บทที่9 การค้นหาและการเรียงลำดับ (Searching and Sorting)

9.1 การค้นหาแบบลำดับ

9.2 การปรับปรุงการกระทำของการค้นหาแบบลำดับ

9.3 การค้นหาแบบทวิภาค

9.4 แนะนำการเรียงลำดับ

- การเรียงลำดับแบบเลือก
- การเรียงลำดับเลือกแบบสับเปลี่ยน
- การเรียงลำดับแบบใส่
- การเรียงลำดับแบบสับเปลี่ยน : การเรียงลำดับแบบฟอง
- การเรียงลำดับแบบสับเปลี่ยนบางส่วน(การเรียงลำดับอย่างเร็ว)
- การเรียงลำดับแบบฮีป
- การเรียงลำดับแบบแข่งขัน

บทสรุป

แบบฝึกหัด

IT 204 279

บทที่ 9
การค้นหาและการเรียงลำดับ
(Searching and Sorting)

ในบทนี้เราจะพักการเรียนรู้เกี่ยวกับโครงสร้างข้อมูลชนิดใหม่และศึกษาเทคนิคต่างๆ สำหรับการหาค่าข้อมูลเฉพาะในโครงสร้างและการเรียงลำดับค่าข้อมูล การค้นหาและการเรียงลำดับเป็นกระบวนการที่เกี่ยวข้องใกล้ชิดกันมาก จงพิจารณากลุ่มของระเบียนหนึ่งชุด ระเบียนแต่ละตัวมีคีย์ (key) ซึ่งสามารถนำมาใช้แสดง ถึงการเป็นระเบียน

คีย์ ประกอบด้วย หนึ่งเซตข้อมูลหรือมากกว่าหนึ่งเซตข้อมูล (A key is composed of one or more data fields) ค่าคีย์อาจเป็นไอเดนติไฟเออร์ซึ่งเป็นเพียงหนึ่งเดียวเท่านั้นของระเบียน หรืออาจจะเป็นค่าซ้ำได้

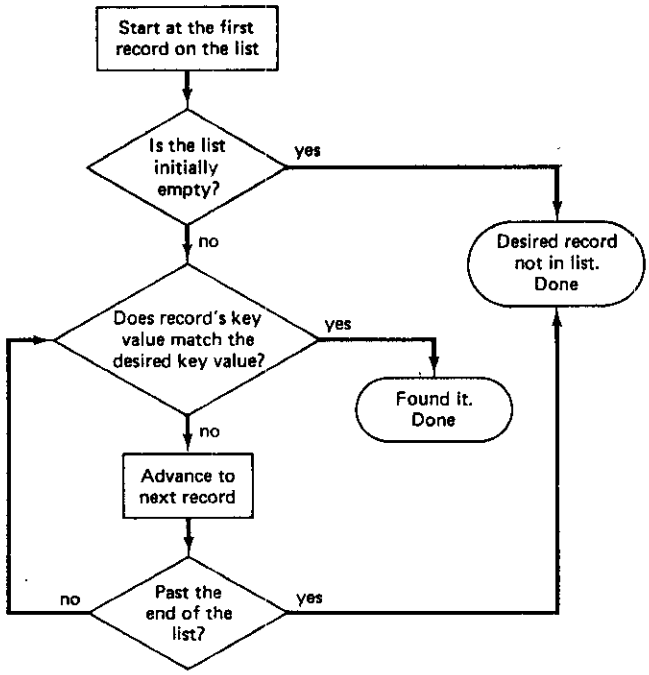
ถ้ายอมให้มีค่าคีย์ซ้ำกันอาจเป็นการพิจารณาเพื่อให้ความแตกต่างระหว่างระเบียนซึ่งมีค่าคีย์เหมือนกัน โดยอันดับการปรากฏของมันในกลุ่ม แนวคิดของคีย์จะขยายไปตลอดในส่วนที่เหลือของหนังสือเล่มนี้

การค้นหา หมายถึง กระบวนการของการหาตำแหน่งของระเบียน ซึ่งมีค่าคีย์เฉพาะ (**Searching** is the process of locating a record with a particular key value)

การเรียงลำดับ หมายถึง กระบวนการของการจัดการระเบียนเพื่อให้มันเรียงลำดับโดยค่าคีย์ (**Sorting** is the process of arranging records so that they are in order by key value)

9.1 การค้นหาแบบลำดับ (Sequential Searches)

อัลกอริทึมการค้นหา หมายถึง เทคนิคการหาระเบียนซึ่งมีค่าคีย์บางอย่าง เราจะเรียกค่าคีย์ว่า k ซึ่งเป็นอาร์กิวเมนต์ของการค้น การค้นหาอาจจบแบบประสบผลสำเร็จ เมื่อพบระเบียนที่มีค่าคีย์เป็น k หรือการค้นหาอาจจะไม่ประสบผลสำเร็จ เมื่อตรวจพบว่าไม่มีระเบียนใดๆเลยที่มีค่าคีย์เท่ากับ k



รูป 9-1 อัลกอริทึมการค้นแบบลำดับ

อัลกอริทึมการค้นมีหลายวิธี ในหัวข้อนี้การค้นหาแบบลำดับซึ่งเรากำลังจะศึกษานี้เรียกว่า **การค้นหาแบบเชิงเส้น** (linear searches) ในตอนท้ายของบทนี้ เราจะพิจารณาการค้นแบบไม่ใช่เชิงเส้น (nonlinear searches) ซึ่งมีแนวโน้มที่พิจารณาแล้วว่ากระทำงานได้ดีกว่าเทคนิคแบบลำดับ อย่างไรก็ตาม การค้นหาแบบลำดับค่อนข้างง่ายและจัดว่าเป็นการวางที่ดีสำหรับการเริ่มต้นศึกษาการค้นหาและการเรียงลำดับ

สมมติว่ากลุ่มของระเบียนซึ่งเรากำลังค้นหา มีการจัดระเบียบแล้วเป็นรายการเชิงเส้น ทั้งนี้รายการเชิงเส้นอาจถูกแทนที่โดยแถวลำดับหรือโดยรายการโยงก็ได้ ในที่นี้เราจะใช้การแทนที่ของแถวลำดับนักศึกษาควรจะตรวจดูว่าการใช้การแทนที่ของแถวลำดับนักศึกษาควรจะตรวจดูว่าการใช้การแทนที่โดยรายการโยงจะเปลี่ยนอัลกอริทึมอย่างไร

ให้ key(i) เป็นค่าคีย์ในระเบียนที่ i ของรายการ อัลกอริทึมการค้นหาแบบลำดับพื้นฐาน คือ เริ่มต้นที่ตอนต้นของรายการและผ่านไปแต่ละระเบียน จนกระทั่งมีหนึ่งระเบียนซึ่งค่าคีย์ (k) ที่ต้องการถูกพบหรือมาจนจบรายการ

ผังงานของอัลกอริทึมอยู่ในรูป 9-1 ข้างต้นนี้ ถูกทำให้เกิดผลในทางปฏิบัติ ด้วย Pascal ดังนี้ สมมติว่ามี n ระเบียนในรายการและ key , k , n เป็นตัวแปรส่วนกลาง

```
procedure findk ;
var     foundit : boolean;  i : 1..n;
begin
        foundit :=  false;
            i :=  1;
          while (not foundit) and i <= n
          do  if (k = key[i])
              then foundit := true
              else i := i +1;
          if (foundit)
          then writeln('record with key', k ,'found at position', I)
          else writeln('record with key',k,'not found')
end;
```

ถ้าอัลกอริทึมจบด้วย foundit = true เพราะฉะนั้นระเบียนที่มีค่าคีย์ k คือ ระเบียนที่ i ในรายการ ถ้าอัลกอริทึมจบด้วย foundit = false เพราะฉะนั้นค่าคีย์ k ไม่มีอยู่ในรายการ

ประสิทธิภาพของโปรซีเดอร์นี้เป็นอย่างไร ให้ prob(i) คือความน่าจะเป็นซึ่งระเบียนที่ i คือสิ่งที่กำลังค้นหา ดังนั้น

$$\sum_{i=1}^{n} prob(i) + Q = 1$$

เมื่อ Q = ความน่าจะเป็นซึ่งไม่มีคีย์อยู่ในรายการ

สถานะการณ์ดีที่สุด คือ ระเบียนซึ่งกำลังค้นหาเป็นระเบียนแรกในรายการ กรณีแย่ที่สุดคือเมื่อคีย์ของระเบียนทั้งหมด n ตัว ถูกเปรียบเทียบกับ k และไม่ว่า key[n] = k หรือระเบียนที่ต้องการค้นหาไม่มีอยู่ในรายการ

ค่าเฉลี่ย , หรือค่าคาดคะเน , จำนวนการเปรียบเทียบ คือ

$$\sum_{i=1}^{n} i * prob(i) + Q * n$$

ด้วยความน่าจะเป็น prob(1) , ต้องเปรียบเทียบ 1 ครั้ง

ด้วยความน่าจะเป็น prob(2) , ต้องเปรียบเทียบ 2 ครั้ง ;

. . .

ด้วยความน่าจะเป็น prob(n)+q,ต้องเปรียบเทียบ n ครั้ง

ถ้าระเบียนทั้งหมดต้องถูกค้นคืนและคีย์ทุกตัวที่ค้นหามีอยู่ในรายการ ดังนั้น

$$prob(i) = \frac{1}{n} \qquad \text{for all } i$$

และ

$$\sum_{i=1}^{n} i * prob\ (i) = \sum_{i=1}^{n} \frac{i}{n} \quad , \text{ซึ่งคือ} \quad \frac{n+1}{2}$$

นั่นคือ ค่าเฉลี่ยประมาณครึ่งหนึ่งของคีย์จะถูกเปรียบเทียบกับอาร์กิวเมนต์ k

ในบางสถานะการณ์ ถ้าคีย์ไม่เป็นเพียงหนึ่งเดียวเท่านั้น(key values are not unique) การหาระเบียนทั้งหมดด้วยค่าคีย์เฉพาะ เพราะฉะนั้นการกวาดตรวจรายการเต็มค่าคีย์ทั้งหมดต้องถูกเปรียบเทียบกับอาร์กิวเมนต์ที่ต้องการค้น อัลกอริทึมการค้นหาแบบ

ลำดับเรียกว่า ต้องใช้การเปรียบเทียบ "on the value of N" นั่นคือจำนวนคาดคะเนของการเปรียบเทียบเป็นฟังก์ชันเชิงเส้นของจำนวนระเบียนในกลุ่ม " on the value of N"ปกติใช้สัญลักษณ์ O(N) จำนวนคีย์เป็นสองเท่า หมายความว่า กระบวนการจะใช้เวลานานเป็นสองเท่า

สิ่งนี้ไม่ใช่วิธีที่ดีของการค้นหา ตัวอย่างเช่น เราน่าจะไม่เคยใช้เทคนิคค้นหาชื่อในสารบบโทรศัพท์ ซึ่งเป็นรายการเชิงเส้นจริงๆ เราจะปรับปรุงสถานการณ์อย่างไร

9.2 การปรับปรุงการกระทำของการค้นหาแบบลำดับ

(Improving Sequential Search Performance)

จงพิจารณานิพจน์สำหรับจำนวนการเปรียบเทียบคาดคะเนอีกครั้งหนึ่ง ด้วยการค้นหาแบบลำดับ

$$\sum_{i=1}^{n} i * prob(i) + Q * n$$

ถ้าเรามีอิสระที่จะจัดการระเบียนต่างๆในรายการเชิงเส้นให้เรียงลำดับอย่างใดอย่างหนึ่งซึ่งเราเลือก ดังนั้นนิพจน์ข้างต้นถูกทำให้ต่ำสุดเมื่อ

$$prob(1) \geq prob(2) \geq \ldots \geq prob(n)$$

นั่นคือ

$$prob(I) \geq prob(J) \qquad \text{for } I < J$$

โดยการจัดการระเบียนต่างๆด้วยความถี่การเข้าถึงจากมากไปหาน้อย เราปรับปรุงความน่าจะเป็นซึ่งการเปรียบเทียบจะถูกกระทำน้อยลง มันอาจจะยังคงเป็นกรณีที่การเปรียบเทียบ n ครั้งยังจำเป็นอยู่(นั่นคือเรากำลังมองหาคีย์ซึ่งไม่มีอยู่ในรายการ) แต่จำนวนของการเปรียบเทียบที่คาดคะเนจะถูกทำให้ต่ำสุด

การเข้าถึงตัวอย่าง (Sampled Accesses)

ขณะนี้จะพิจารณาสถานการณ์ปกติมากขึ้น ซึ่งไม่ทราบความถี่สัมพัทธ์ของการเข้าถึงระเบียน มีโครงร่างหลายชนิดที่จัดกระทำกับปัญหานี้ โครงร่างหนึ่งคือสังเกตการณ์ ร้องขอกับรายการผ่านช่วงเวลาหนึ่ง เก็บการนับจำนวนการเข้าถึงแต่ละระเบียน เมื่อ ตัวอย่างตัวแทนของกิจกรรมได้ถูกรวบรวมไว้ ระเบียนอาจถูกบันทึกใหม่เป็นไปตามการตรวจสอบพบความน่าจะเป็นของการเข้าถึง

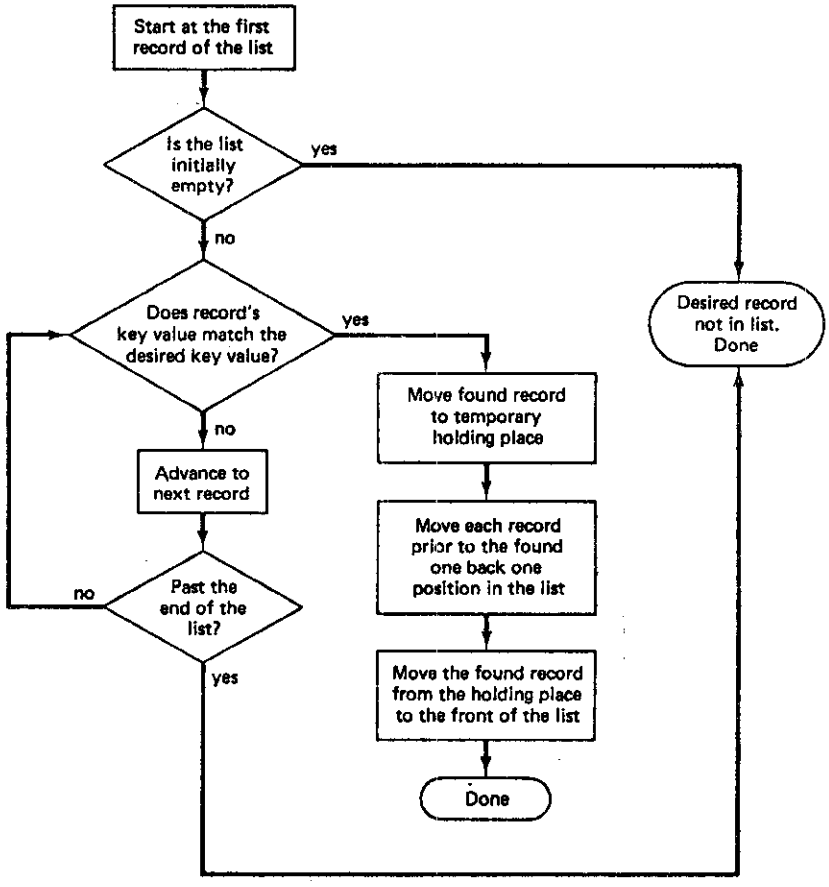
ย้ายไปส่วนหน้า(Move to the Front)

โครงร่างชนิดที่สองคือให้รายการของระเบียนต่างๆจัดระเบียบใหม่ด้วยตัวเองอย่างพลวัตด้วยวิธีย้ายไปส่วนหน้า เมื่อใดก็ตามที่การค้นหาคีย์ประสบผลสำเร็จ ระเบียนที่สมนัยกัน จะย้ายไปตำแหน่งแรกในรายการ ดังนั้นระเบียนซึ่งเป็นเลขที่หนึ่งจะกลายเป็นเลขที่สองเช่นนี้เรื่อยไป ผังงานของอัลกอริทึมย้ายไปส่วนหน้าอยู่ในรูป 9-2 อัลกอริทึมจะถูกทำให้เกิดผลใน Pascal เป็นดังนี้ สมมติว่าในรายการมี n ระเบียนและ rec(ซึ่งเก็บ key) , k และ n เป็นตัวแปรส่วนกลาง

```
procedure movetofront;
var     foundit : boolean;
        i : 1 . . n;
        j : 0 . . n;
        temprec : rectype;
begin
        foundit := false;
              i := 1;
         while (not foundit) and i <= n
         do if  k =key[i]
            then foundit := true
            else i := i +1;
         if (foundit)
         then    begin  writeln('record with key',k, 'found');
                        temprec  := rec[i];
                              j  := i - 1;
                             while j > 0
                             do begin  rec[j + 1] := rec[j];
                                                j  := j -1
                                end;
                             rec[1] := temprec
                 end;
```

```
            else  writeln('record with key',k, 'not found')
        end;
```

วิธีนี้กระทำได้บนรายการเชิงเส้นซึ่งแทนที่โดยรายการโยงดีกว่าการแทนที่โดยแถวลำดับ ทำไม ?



รูป 9-2 " Move-to-the-front " sequential search algorithm

การสับเปลี่ยนที่กัน(Transposition)

โครงร่างการจัดระเบียบใหม่แบบพลวัตอีกวิธีหนึ่งคือวิธีสับเปลี่ยนที่กัน เมื่อใดก็ตามที่การค้นค่าคีย์ประสบผลสำเร็จ ระเบียนที่สมนัยกันจะสับเปลี่ยน(interchange)กับระเบียนที่อยู่ก่อนหน้าทันทีใน Pascal ข้อสมมติฐานเกี่ยวกับการประกาศตัวแปรเหมือนกับในโปรซีเดอร์ movetofront

```
procedure transpose;
var     foundit : boolean;
        i : 1 . . n;
        temprec : rectype;
begin
        foundit := false;
            i   := 1;
           while (not foundit) and i <= n
           do if (k = key[i])
               then foundit := true
                else i := i +1;
            if (foundit and i >1)
            then begin writeln('record with key',k,found');
                        temprec := rec[i];{transpose adjacent records}
                        rec[i] := rec[I - 1];
                        rec[i-1] := temprec
                   end;
             else writeln('record with key',k, 'not found')
end;
```

การค้นหาระเบียนยิ่งมีความถี่มากขึ้น ความเร็วที่มันไหล(percolates)ไปหนึ่งตำแหน่งยิ่งเร็วมากขึ้น เปรียบเทียบกับวิธี move-to-the-front วิธีสับเปลี่ยนที่กันจะใช้เวลาของกิจกรรมเพื่อจัดระเบียบใหม่ให้กลุ่มของระเบียนนานกว่า ข้อดีของวิธีสับเปลี่ยนที่กันคือ

มันไม่ยอมให้การร้องขอครั้งเดียวสำหรับระเบียนเพื่อย้ายกลุ่มทั้งหมดของระเบียน ผลกระทบระเบียนต้องไปทางขวาของมันสู่ตอนบนตามประวัติของความต้องการ

การเรียงลำดับ(Sorting)

วิธีหนึ่งเพื่อลดจำนวนของการเปรียบเทียบคาดคะเน เมื่อมีความถี่อย่างมีนัยสำคัญของการค้นหาที่ไม่ประสบผลสำเร็จ คือ การเรียงอันดับระเบียนโดยค่าคีย์ นั่นคือ

key(I) <= key(J) for I < J

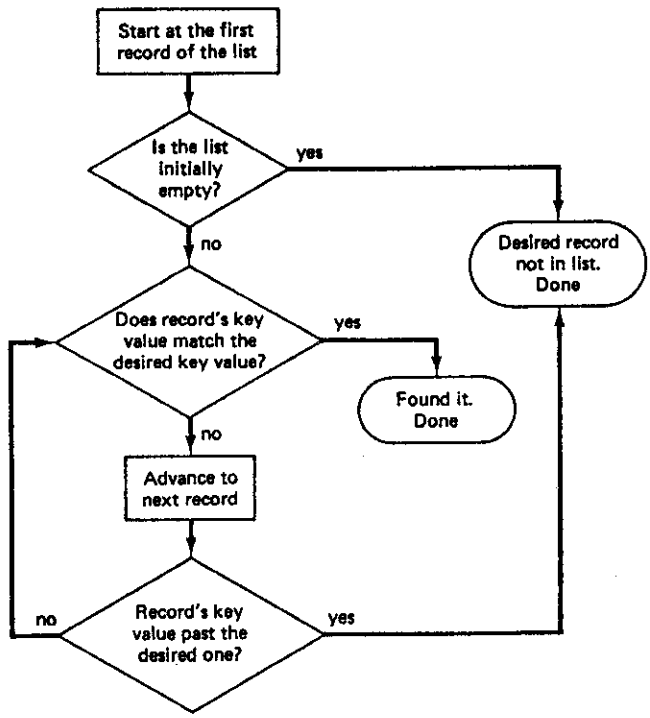
หรือ key(I) >= key(J) for I < J ในการเรียงลำดับลง(descending order)

เทคนิคนี้มีประโยชน์เมื่อรายการ คือ รายการของข้อยกเว้น(exceptions) เช่น เลขบัตรเครดิตไม่ถูกต้อง(bad credit card numbers) ถูกเปรียบเทียบอีกครั้ง ส่วนใหญ่ของการค้นหาคือไม่ประสบผลสำเร็จ ขณะนี้การค้นหาไม่ประสบผลสำเร็จ จบเมื่อคีย์ตัวแรกมีค่ามากกว่าอาร์กิวเมนต์ ไม่ใช่ที่ตอนจบของรายการ เช่นที่แสดงให้เห็นในผังงานรูป 9-3

อัลกอริทึมทำให้เกิดผลใน Pascal เขียนดังนี้

```
foundit := false;
  i  := 1;
while (not foundit) and key[i] <= k
do if (k=key[i])
    then foundit := true
    else i:= i+1;
```

ในตอนท้ายของบทนี้ จะพิจารณาเทคนิคต่าง ๆสำหรับการเรียงลำดับสมาชิกทั้งหมดในรายการ



รูป 9-3 Sequential search algorithm for a sorted list.

9.3 การค้นหาแบบทวิภาค(The Binary Search)

เทคนิคการค้นหาแบบลำดับทั้งหมดต้องการการเปรียบเทียบ O(n) การกระทำ(performance)ของการค้นหาแบบลำดับพื้นฐาน สามารถปรับปรุงให้ดีขึ้นได้ โดยการทำตำแหน่งที่เร็วของระเบียน(โดยการทราบความน่าจะเป็นของการเข้าถึง , โดยการย้ายระเบียนเป้าหมายทันที หรือทำให้ช้ามากขึ้นไปยังส่วนหน้าของรายการ , หรือโดยการรักษาระเบียนในการเรียงตามลำดับ) แต่พฤติกรรม O(n) ยังคงอยู่ จริงๆแล้วการปรับปรุงการกระทำต้องใช้วิธีการเข้าถึงที่แตกต่างเพื่อการค้นหา - เทคนิคแบบไม่ใช่เชิงเส้น (nonlinear techniques)

ในบทที่ 7 และ 8 เราเริ่มต้นศึกษาโครงสร้างข้อมูลแบบไม่ใช่เชิงเส้น เราค้นพบว่าโดยการทำโครงสร้างข้อมูลในต้นไม้แบบทวิภาค , เวลาการค้นหาระเบียนเฉพาะของกลุ่มระเบียนอาจลดลงอย่างมีนัยสำคัญ ผ่านการทำโครงสร้างซึ่งข้อมูลในโครงสร้างข้อมูลแบบเชิงเส้น สำหรับกลุ่มระเบียน n ตัว ความยาวการค้นเฉลี่ยคาดคะเนของต้นไม้ได้ดุลคือ $O(\log_2 n)$ ในขณะที่ความยาวการค้นเฉลี่ยของรายการเชิงเส้นคือ $O(n/2)$ สำหรับ n ที่มีขนาดใหญ่ต้นไม้แบบทวิภาคแสดงให้เห็นว่า การกระทำดีกว่ามากเพียงแค่เป็นโครงสร้างการแตกกิ่งของโครงสร้างข้อมูลแบบไม่ใช่เชิงเส้นช่วยลดความยาวการค้นหาลง ดังนั้นเทคนิคการค้นหาและการเรียงลำดับแบบไม่ใช่เชิงเส้นจึงปรับปรุงการกระทำให้เหนือกว่าวิธีแบบลำดับ เราได้รวบรวมยุทธวิธีการค้นหาต้นไม้ไม่ใช่เชิงเส้น ในการอภิปรายของต้นไม้ค้นแบบทวิภาค เมื่อเรามองหาโหนดที่มีค่าคีย์เฉพาะ , การเปรียบเทียบที่โหนดแต่ละตัวทันที ทำให้เราลงยังกิ่งที่ถูกต้องของต้นไม้ไปยังจุดหมายปลายทางในต้นไม้ได้ดุจแบบบริบูรณ์ การเปรียบเทียบแต่และครั้งลบครึ่งหนึ่งของโหนดที่เหลือบนต้นไม้ออกไป ดังนั้น ความยาวการค้นเฉลี่ยคาดคะเนในต้นไม้ค้นทวิภาคแบบได้ดุลคือ $O(\log_2 n)$ นี่คือการกระทำที่ดีเมื่อเปรียบเทียบกับวิธีการค้นแบบลำดับ

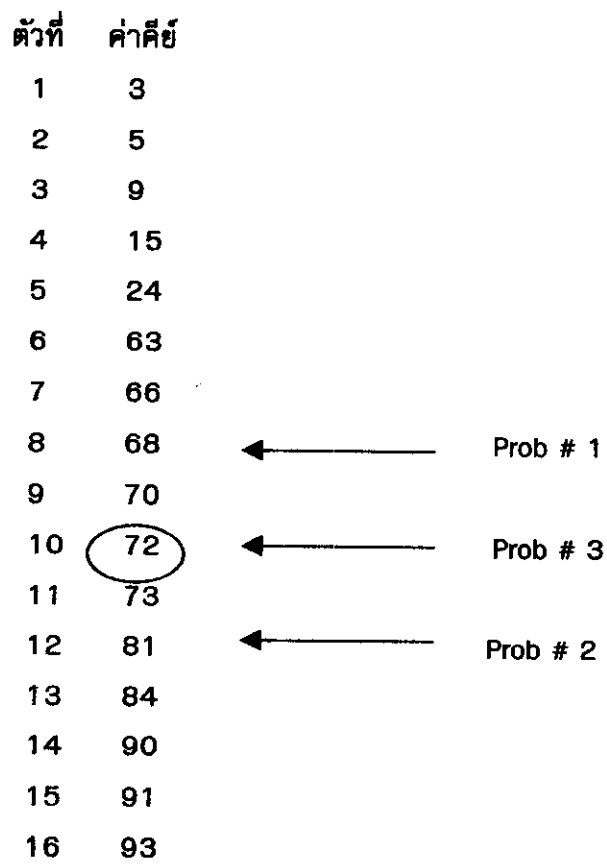
เทคนิคการค้นแบบทวิภาคถูกนำมาประยุกต์ใช้กับข้อมูลในรายการเชิงเส้น เช่นเดียวกับข้อมูลในต้นไม้ค้นแบบทวิภาค สิ่งสำคัญที่ต้องมาก่อนคือ ระเบียนในรายการ(list) ต้องเรียงลำดับ โดยค่าคีย์ที่จะถูกค้นหาและต้องทราบจำนวนของระเบียน เทคนิคการค้นแบบทวิภาคไม่สามารถประยุกต์ใช้ได้ ถ้ายังไม่มีสิ่งที่ต้องมีมาก่อนที่กล่าวข้างต้น

กระบวนการค้นหาโดยการตรวจ(probes)อย่างสืบเนื่องในรายการ การตรวจสอบครั้งแรกเปรียบเทียบค่าคีย์ในตำแหน่งตรงกลางของรายการเรียงซึ่งลำดับแล้วกับค่าที่ค้นหา (sought value) ถ้าค่าที่ค้นหามีค่าน้อยกว่า ดังนั้นครึ่งหนึ่งของรายการตามหลังตัวตรงกลางถูกตัดออกไป ในทางตรงกันข้ามถ้าค่าที่ค้นหามีค่ามากกว่าค่าคีย์ตรงกลาง ดังนั้นครึ่ง

หนึ่งของรายการก่อนตัวตรงกลางไม่จำเป็นต้องพิจารณาอีกต่อไป การตรวจสอบครั้งที่สอง คือ คีย์ในตำแหน่งตรงกลางของครึ่งที่เหลือของรายการเดิม อีกครั้งหนึ่งถ้าค่าที่ค้นหาน้อยกว่าค่าของคีย์ตรวจสอบนี้ ดังนั้นครึ่งของ(ครึ่ง-)รายการตามหลังการตรวจสอบจะถูกลบออกไปจากการพิจารณา ถ้าค่าที่ค้นหามากกว่าค่าของคีย์ตรวจสอบนี้ ดังนั้นครึ่งของ (ครึ่ง-)รายการก่อนการตรวจสอบไม่จำเป็นต้องพิจารณาต่อไปและดังนั้น กระบวนการตรวจสอบทำต่อไปจนกระทั่งค่าคีย์ที่ค้นหาถูกพบหรือค่าคีย์ที่ค้นหาไม่ได้อยู่ในรายการ

ตัวอย่างเช่น รูป 9-4 แสดงให้เห็นลำดับของกิจกรรมตรวจสอบเกี่ยวข้องในการค้นแบบทวิภาคสำหรับค่าคีย์ 72 ในรายการเชิงเส้นตัวอย่าง เริ่มต้นมีคีย์ 16 ตัว ให้พิจารณาการตรวจสอบครั้งแรก คือ คีย์ตัวที่ 8 (16/2 = 8) เนื่องจาก 68 < 72 การค้นหาต่อไปต้องการเฉพาะที่ครึ่งที่สองของรายการ การตรวจสอบครั้งที่สองคือ คีย์ตัวที่ 12 ((8+16)/2 = 12) : 81 เนื่องจาก 81 > 72 ขณะนี้เราตัดทนึ่งในสี่สุดท้ายของรายการทิ้ง การตรวจสอบครั้งที่สามคือคีย์ตัวที่ 10 ((8+12)/2=10) : 72 ซึ่งเป็นค่าคีย์ตัวที่ค้นหา ในที่นี้จึงต้องใช้การตรวจสอบสามครั้ง ; ถ้าเป็นการค้นหาแบบเชิงเส้นจะต้องเปรียบเทียบถึง 10 ครั้ง สำหรับรายการขนาดใหญ่การปรับปรุงการกระทำคือ ทำให้รวดเร็วมากขึ้น

รายการถูกตัดออกทีละครึ่ง ครั้งแล้วครั้งเล่าจนกระทั่งค่าคีย์ตรงกลาง คือ ค่าคีย์ที่ต้องการ(ค่าที่ค้นหา)หรือขนาดของรายการที่เหลือเป็นศูนย์ แสดงว่าค่าคีย์ที่ค้นหาไม่อยู่ในรายการ



รูป 9-4 Binary search probes

เทคนิคการค้นหาแบบทวิภาคนำมาโปรแกรมเป็นอัลกอริทึมแบบวนซ้ำ หรืออัลกอริทึมแบบเรียกซ้ำได้ทั้งคู่ โปรซีเดอร์การค้นหาแบบทวิภาคเขียนด้วย COBOL เป็นดังนี้ รายการเชิงเส้นของคีย์ N ตัวเก็บในแถวลำดับชื่อ KEYLIST ค่าคีย์ค้นหาคือ SKEY , ส่วน FOUNDIT กำหนดให้เท่ากับดรรชนีล่างของสมาชิกที่มีค่า SKEY ถ้ามีอยู่หรือเท่ากับ 0 ถ้าไม่พบ SKEY

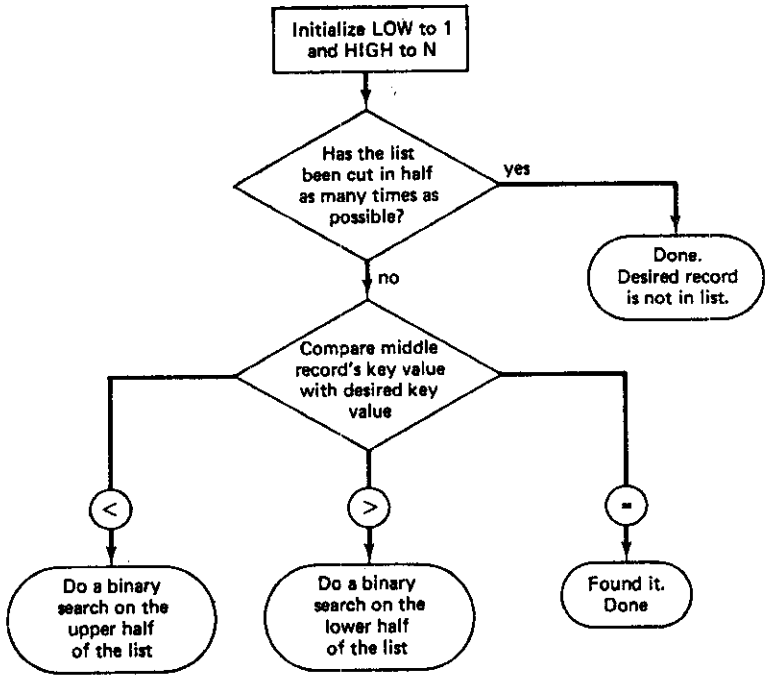
```
01    KEYLIST.
      02    KEYVALUE OCCURS N TIMES PICTURE S9(5).
01    AUX-VARIABLES.
      02    LOW       PICTURE    9(4).
      02    HIGH      PICTURE    9(4).
      02    MID       PICTURE    9(4).
      02    SKEY      PICTURE    S9(5).
```

```
        02    FOUNDIT      PICTURE     9(4).
```

เมื่อใน PROCEDURE DIVISION เขียนดังนี้

```
BINARY-SEARCH.
        COMPUTE   FOUNDIF  = 0.
        COMPUTE   LOW      = 1.
        COMPUTE   HIGH     = N.
        PERFORM   PROBE    UNTIL  LOW > HIGH
                           OR  FOUNDIF > 0
PROBE.
        COMPUTE   MID = (LOW+HIGH)/2
        IF SKEY = KEYVALUE(MID)
           COMPUTE FOUNDIT = MID
        ELSE IF SKEY < KEYVALUE(MID)
                   COMPUTE  HIGH = MID + 1
                ELSE  COMPUTE  LOW = MID +1 .
```

ในรูทีนนี้ตรรชนีล่าง LOW , MID และ HIGH ถูกป้องกันจากการมีค่าเป็นเศษส่วน เพราะว่า ประกาศแล้วให้เป็นจำนวนเต็ม



รูปที่ 9-5 Recursive binary search algorithm

ผังงาน(flowchart)สำหรับอัลกอริทึมการค้นหาทวิภาคแบบเรียกซ้ำ กำหนดให้แล้วในรูป 9-5 อัลกอริทึมนี้เขียนด้วย Pascal เป็นดังนี้ สมมติว่าชื่อตัวแปรเหมือนกับในโปรซีเดอร์ของ COBOL แถวลำดับ keylist และตัวแปรที่ค้นหา skey เป็นตัวแปรส่วนกลาง พิสัยของดรรชนี key เริ่มจาก 1 ถึง n

```
procedure binasrch(var low,high,foundit : integer);
var     mid : integer;
begin
        if (low > high) then foundit :=0
        else  begin  {prob}
                        mid :=  (low+high) div 2;
                        if (skey = key[mid])
                        then   foundit :=  mid
                        else if (skey < key[mid])
                            then  binsrch(low,mid – 1 ,foundit)
                            else  binsrch(mid + 1 ,high,foundit)
                    end {prob}
end {binsrch};
```

ในโปรซีเดอร์นี้ถูกเรียกโดยข้อความสั่ง เช่น binsrch(1,n,foundit) ตัวแปร foundit มีค่าเป็นดรรชนีของสมาชิกของแถวลำดับด้วยค่า skey หรือ foundit มีค่าเป็น 0 ถ้า skey ไม่อยู่ในแถวลำดับ

จำนวนการเปรียบเทียบมากที่สุดสำหรับการค้นหาแบบทวิภาคของรายการเชิงเส้นซึ่งเรียงลำดับแล้วของสมาชิกเท่ากับ $\log_2 n$ จำนวนการเปรียบเทียบน้อยที่สุดคือ 1 และจำนวนการเปรียบเทียบคาดคะเนเท่ากับ $1/2\log_2 n$ เพราะฉะนั้น การค้นหาแบบทวิภาคเป็น $O(\log_2 n)$

การค้นหาแบบทวิภาคปกติควรจะถูกนำมาใช้ไม่เพียงแต่ใช้กับการค้นหาแบบลำดับบนกลุ่มข้อมูลขนาดใหญ่ ซึ่งมีการเรียงอันดับโดย search key ที่จริงโดยปกติจะเป็นข้อดีที่สร้างรายการเรียงตามอันดับ โดยมีวัตถุประสงค์เพื่อให้สามารถใช้เทคนิคการค้นหาแบบทวิภาคได้ สิ่งนี้ปกติใช้การค้นหาแบบทวิภาคควรถูกดัดแปร บางอย่างเพื่อแสดงถึงความสามารถของมัน เพื่อการค้นหากลุ่มของข้อมูลทั้งหมดที่เก็บในหน่วยความจำหลัก การค้นหาแบบทวิภาคเป็นวิธีการค้นหาภายใน (The binary search is an internal search method) แนวคิดของการค้นหาแบบไม่ใช่เชิงเส้น สามารถประยุกต์ใช้กับกลุ่มข้อมูลที่เก็บ

ในหน่วยเก็บรอง(secondary storage)ซึ่งจะได้พิจารณาการจัดระเบียบข้อมูลเช่นนี้ในตอนท้ายของหนังสือนี้

9.4 แนะนำการเรียงลำดับ(Introduction to Sorting)

ขณะนี้ให้ดูเทคนิคสำหรับการเก็บระเบียนแบบเรียงตามอันดับซึ่งเราจะเห็นความสำคัญสำหรับปรับปรุงการกระทำของการค้นหา

เทคนิคการเรียงลำดับแบ่งออกเป็นสองชนิดคือ การเรียงลำดับภายในและการเรียงลำดับภายนอก **วิธีการเรียงลำดับภายใน** ถูกนำมาใช้เมื่อกลุ่มข้อมูลทั้งหมดซึ่งต้องการเรียงลำดับมีปริมาณน้อยที่ทำให้การเรียงลำดับสามารถเกิดขึ้นได้ ภายในหน่วยความจำหลัก(Internal sorting methods are applied when the entire collection of data to be sorted is small enough that the sorting can take place within main memory) **วิธีการเรียงลำดับภายนอก** ถูกนำมาใช้กับกลุ่มของข้อมูลที่มีปริมาณมากกว่าซึ่งส่วนใหญ่จะเป็นกลุ่มข้อมูลที่เก็บในอุปกรณ์หน่วยความจำช่วย เช่น เทปแม่เหล็กหรือดิสก์ (External sorting methods are applied to larger collections of data where some (frequently most) of the collection resides on an auxiliary memory device such as magnetic tape or disk)

ในที่นี้ช่วงเวลาเข้าถึงอ่านและเขียนเป็นสิ่งเกี่ยวข้องสำคัญในการคำนวณหาการกระทำของการเรียงลำดับ การศึกษาวิธีการเรียงลำดับภายนอกจะอยู่ในส่วนของการประมวลผลแฟ้มของหนังสือเล่มนี้

ด้วยเหตุที่ความสนใจอยู่ที่ให้ได้มาซึ่งรายการของระเบียนที่เรียงลำดับ เราจึงไม่สนใจระเบียนและสนใจเฉพาะการได้มาซึ่งรายการของคีย์ที่เรียงลำดับ อย่าเพิ่งสับสนกับวิธีลัดนี้ ถ้าเราชอบเมื่อเราระบุว่า " a key " ควรถูกใส่ในรายการหรือ " a key " ควรถูกเลือกจากรายการ หรือ " two keys " ในรายการควรจะสับเปลี่ยนตำแหน่งกัน " a key " หมายถึง " a key and its accompanying record " หรือ " a key and the pointer to its corresponding record " โดยตลอด สมมติว่า ผลลัพธ์ของการเรียงลำดับคือ คีย์ถูกเรียงจากน้อยไปหามาก คือ การเรียงลำดับขึ้น(ascending order)

นักศึกษาควรจะสามารถเปลี่ยนแปลงบางสิ่งได้เพื่อให้อัลกอริทึมให้ผลลัพธ์เป็นการเรียงลำดับลง(descending order)ของคีย์

เราจะไม่กำหนดว่าค่าคีย์ต้องเป็นไอเดนติไฟเออร์ของระเบียนที่เป็นเพียงหนึ่งอย่าง (unique) อัลกอริทึมยังคงเสถียร(stable) ถ้าการเรียงลำดับเดิมของระเบียนด้วยค่าคีย์เท่า

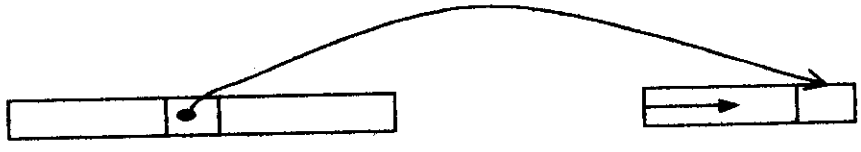
กันถูกรักษาไว้ในการเรียงตามอันดับของระเบียนเหล่านั้น เป็นสิ่งสำคัญที่ใช้วิธีการเรียงลำดับที่เสถียร ถ้าการเรียงอันดับระเบียนเดิมมีความหมาย ตัวอย่างเช่น การเรียงอันดับนั้นอาจถูกสร้างขึ้นในเวลาของข้อมูลของระเบียนในรายการเดิม วิธีการเรียงลำดับไม่ใช่ทั้งหมดซึ่งเสถียร(Not all sort methods are stable) ในการอภิปรายเทคนิคการค้นหาแบบเชิงเส้น เราสมมติว่ารายการเชิงเส้นซึ่งเรียงลำดับถูกแทนที่ด้วยแถวลำดับ นักศึกษาควรจะตรวจสอบว่า การใช้แทนที่แบบรายการโยงจะต้องเปลี่ยนจะอัลกอริทึมอย่างไร

9.4.1 การเรียงลำดับแบบเลือก(A Selection Sort)

ครอบครัวหนึ่งของอัลกอริทึมการเรียงลำดับภายในได้แก่ กลุ่มของการเรียงลำดับแบบเลือก

ความคิดพื้นฐานของการเรียงลำดับแบบเลือก คือ เลือกคีย์ตัวที่มีค่าน้อยที่สุดที่เหลืออยู่ในรายการข้อมูลซึ่งไม่เรียงลำดับเป็นคีย์ตัวถัดไปในรายการข้อมูลซึ่งเรียงลำดับและโตขึ้นเรื่อยๆโดยทำซ้ำๆกัน (รูป 9-6)

(The basic idea of a selection sort is to repeatedly select the smallest key remaining in the unsorted list of data as the next key in a growing sorted list of data.)



รูปที่ 9-6 การเรียงลำดับแบบเลือก

รายการทั้งหมดของคีย์ซึ่งไม่เรียงลำดับต้องมีอยู่ให้สามารถเลือกคีย์ตัวที่มีค่าน้อยที่สุดในรายการ อย่างไรก็ตามรายการซึ่งเรียงลำดับแล้วจะเป็นเอาต์พุต

ตัวอย่างเช่น จงพิจารณารายการของคีย์ซึ่งยังไม่เรียงลำดับข้างล่างนี้

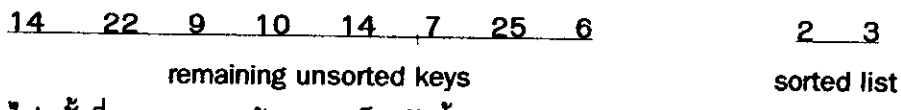
14 3 22 9 10 14 2 7 25 6

ผ่านการเลือกครั้งแรก แสดงว่า 2 เป็นสมาชิกตัวที่มีค่าน้อยที่สุดและลบมันออกจากรายการไม่เรียงลำดับ

14 3 22 9 10 14 7 25 6 2

remaining unsorted keys sorted list

ผ่านการเลือกครั้งสอง แสดงว่า 3 เป็นสมาชิกตัวที่มีค่าน้อยที่สุดและลบมันออกจากรายการไม่เรียงลำดับ



หลังจากผ่านไปครั้งที่หก รายการข้อมูลจะเป็นดังนี้

ใน COBOL การเรียงลำดับแบบเลือกโดยตรงคือ

```
01    UNSORTED-TABLE.
      02    UNSORTED-ENTRY    OCCURS N TIMES PICTURE S99.
01    SORTED-TABLE.
      02    SORTED-ENTRY      OCCURS N TIMES PICTURE S99.
01    AUX-ITEMS.
      02    CURR-MIN-KEY      PICTURE S99 VALUE HIGH-VALUES.
      02    NEXT-IN-SORT      PICUTURE S99 VALUE 1.
      02    I                 PICTURE 9(3).
      02    MIN-POS           PICTURE 9(3).
```

ใน Procedure division เขียนดังนี้

```
      PERFORM     SELECT-NEXT N  TIMES
```

เมื่อ

```
SELECT-NEXT.
      PERFORM  PICK-MIN   VARYING I FORM 1 BY 1 UNTIL I > N.
      MOVE  UNSORTED-ENTRY(MIN-POS) TO
            SORTED-ENTRY(NEXT-IN-POST).
      MOVE  HIGH-VALUES TO UNSORTED-ENTRY(MIN-POS).
      MOVE  HIGH-VALUES TO CURR-MIN-KEY.
```

และ

```
PICK-MIN.
      IF    UNSORTED-ENTRY(I) < CURR-MIN-KEY
            MOVE  I TO MIN-POS
```

IT204

MOVE UNSORTED-ENTRY(I) TO CURR-MIN-KEY.

สำหรับรายการที่มี n ระเบียน อัลกอริทึมนี้ต้องเขียนผ่านรายการซึ่งไม่เรียงลำดับ n ครั้ง ใน pass ที่ I , กระทำการเปรียบเทียบค่าของคีย์ n-I ครั้ง ดังนั้นจำนวนการเปรียบเทียบทั้งหมดคือ :

$$\sum_{i=1}^{n}(n-i)$$

ซึ่งเท่ากับ

$n(n-1)/2$

การเรียงลำดับนี้เรียกว่า ต้องเปรียบเทียบ $O(n^2)$ เพราะว่า n^2 เป็นเทอมที่สำคัญในนิพจน์ จำนวนของการเปรียบเทียบเป็นสัดส่วนกับกำลังสองของจำนวนคีย์ในกลุ่มข้อมูล จำนวนคีย์เป็นสองเท่า หมายความว่า กระบวนการจะใช้เวลานานเป็นสี่เท่า

9.4.2 การเรียงลำดับเลือกแบบสับเปลี่ยน(An Exchange Selection Sort)

โปรแกรมข้างต้นจะใช้เนื้อที่หน่วยความจำมากเกือบจะเป็นสองเท่า การดัดแปรการเรียงลำดับแบบเลือกโดยตรง คือ การเรียงลำดับเลือกแบบสับเปลี่ยน ซึ่งค่าคีย์ที่ถูกเลือกจะถูกย้ายไปยังตำแหน่งสุดท้ายของมันโดยการสับเปลี่ยนกับคีย์ที่ตอนแรกอยู่ในตำแหน่งนั้น จงพิจารณารายการไม่เรียงลำดับอีกครั้งหนึ่ง

14 3 22 9 10 14 2 7 25 6

หลังจากผ่านครั้งแรก , 2 ถูกเลือก

2 | 3 22 9 10 14 14 7 25 6

sorted | unsorted

หลังจากผ่านครั้งที่สอง

2 3 | 22 9 10 14 14 7 25 6

sorted | unsorted

หลังจากผ่านครั้งที่หก

2 3 6 7 9 10 | 14 14 25 22

sorted | unsorted

โปรดสังเกตว่า นี่ไม่ใช่การเรียงลำดับซึ่งเสถียร ตัวอย่างเช่น ในที่นี้ผ่านครั้งที่หนึ่งใส่ 14 ตัวแรก หลัง 14 ตัวที่สอง ; ผ่านครั้งที่เจ็ดและครั้งที่แปด , เลข 14 ทั้งสองตัวจะใส่ต่อท้ายรายการซึ่งเรียงลำดับแล้ว ในลักษณะย้อนลำดับ(reversed order) ใน COBOL การเรียงลำดับเลือกแบบสับเปลี่ยนเขียนดังนี้

```
01    KEY-TABLE.
      02    KEY-ENTRY    OCCURS N TIMES PICTURE S99.
01    AUX-ITEMS.
      02    CURR-MIN-KEY        PICTURE S99 VALUE HIGH-VALUES.
      02    NEXT-IN-SORT        PICTURE  9(3) VALUE 1.
      02    MIN-POS             PICTURE  9(3).
      02    TEMP-KEY            PICTURE  S99.
```

ใน Procedure division เขียนดังนี้

```
EXCHANGE-DRIVER.
      PERFORM     SELECT-NEXT  VARYING  NEXT-IN-SORT
                        FROM 1 BY 1 UNTIL NEXT-IN-SORT = N.
```

เมื่อ

```
SELECT-NEXT.
      PERFORM     PICK-MIN  VARYING  I FROM
                  NEXT-IN-SORT BY 1 UNTIL I > N.
      MOVE     KEY-ENTRY(NEXT-IN-SORT) TO TEMP-KEY.
      MOVE     KEY-ENTRY(MIN-POS) TO KEY-ENTR(NEXT-IN-SORT).
      MOVE     TEMP-KEY TO KEY-ENTRY(MIN-POS).
      MOVE     HIGH-VALUES TO CURR-MIN-KEY.
```

และ

```
PICK-MIN.
      IF KEY-ENTRY(I) < CURR-MIN-KEY
          MOVE  I TO MIN-POS
          MOVE KEY-ENTRY(I) TO CURR-MIN-KEY.
```

การเรียงลำดับเลือกแบบสับเปลี่ยนมีความสำคัญ ต้องการการเปรียบเทียบเหมือนกับการเรียงลำดับแบบเลือกโดยตรง นั่นคือ $O(N^2)$ ต่อไปจะพิจารณาสมาชิกอื่นๆของครอบครัวการเรียงลำดับแบบเลือก ซึ่งกระทำงานได้ดีกว่านี้

9.4.3 การเรียงลำดับแบบใส่(An Insertion Sort)

อีกครอบครัวหนึ่งของอัลกอริทึมการเรียงลำดับภายใน คือ กลุ่มของการเรียงลำดับแบบใส่ ความคิดพื้นฐานของการเรียงลำดับแบบใส่คือ ให้เอาคีย์ตัวถัดไปของรายการซึ่งยังไม่เรียงลำดับและใส่มันในตำแหน่งสัมพัทธ์ที่ถูกต้องของมัน การโตขึ้นของรายการข้อมูลซึ่งเรียงลำดับ(ดูรูป 9-7)

(The basic idea of an insertion sort is to take the next key of the unsorted list and insert it in its proper relative position in a growing sorted list of data)

Unsorted keys Sorted keys

รูป 9-7 การเรียงลำดับแบบใส่

รายการทั้งหมดของคีย์ซึ่งเรียงลำดับต้องอยู่ตลอดกระบวนการเพื่อให้สามารถใส่คีย์ในตำแหน่งสัมพัทธ์ที่ถูกต้องของมันได้ อย่างไรก็ตาม รายการซึ่งยังไม่เรียงลำดับจะเป็นอินพุต

ให้เปรียบเทียบเทคนิคการเรียงลำดับนี้กับวิธีการเรียงลำดับแบบเลือกให้หัวข้อที่แล้ว เมื่อเราเรียงอันดับไพ่บนมือ ถ้าเราหยิบไพ่แต่ละใบของเราเปิดดูและใส่ลงในช่อง(slot)ที่ถูกต้องสัมพันธ์กับไพ่อื่นๆที่มีอยู่แล้วในมือ นั่นคือ เรากำลังใช้การเรียงลำดับแบบใส่ ในทางตรงกันข้าม ถ้าเราคอยจนกระทั่งในมือได้ไพ่ครบทุกใบแล้ว จากนั้นจึงหยิบไพ่ซึ่งควรจะอยู่ซ้ายมือสุด ไปวางตำแหน่งซ้ายมือสุด จากนั้นหยิบไพ่ซึ่งควรจะเป็นใบที่สองไปไว้ในตำแหน่งที่สอง เช่นนี้เรื่อยไป วิธีนี้คือ ใช้การเรียงลำดับแบบเลือก

เทคนิคทั้งสองวิธีนี้ค่อนข้างแย่(relatively poor) เพราะว่าเป็น $O(N^2)$ แต่เป็นวิธีที่ค่อนข้างง่ายในการทำความเข้าใจและเขียนโปรแกรม จึงใช้กันอย่างแพร่หลายทั้งคู่

ให้พิจารณาตัวอย่างรายการซึ่งไม่เรียงอันดับของคีย์อีกครั้งหนึ่ง

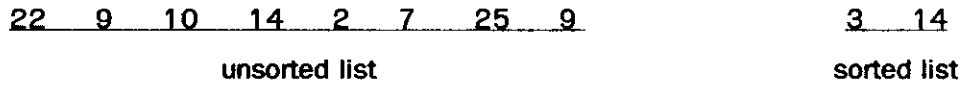
14 3 22 9 10 14 2 7 25 6

ผ่านครั้งแรก พิจารณาคีย์ตัวแรก , คือ 14 , ผลลัพธ์คือ

3 22 9 10 14 2 7 25 6 14

unsorted list sorted list

หลังจากผ่านครั้งที่สอง



หลังจากผ่านครังที่หก

คล้ายกับการเรียงลำดับเลือกแบบสับเปลี่ยน , การเรียงลำดับแบบใส่เขียนเป็นโปรแกรมภาษา Pascal ดังนี้

```
procedure insertsort;
var     key : array[1. . n] of integer;
        tempkey : integer;
        i , j , jj  : 1. . n;
        foundpos  : boolean;
begin
   for  i := 2  to  n
      do begin  j := 1
            foundpos :=  false;
            while (not foundpos) and  j < i
            do if (key[i] < key[j])
               then  foundpos := true
               else  j :=  j+1;
               if (foundpos)
               then  begin
                        tempkey := key[i];
                            jj   := i -1
                        while (jj > j - 1)
                        do  begin
                              key[jj + 1] := key[jj];
```

```
                    jj:= jj - 1
                end;
            key[j]:= tempkey;
        end;
    end;
end;
```

ทำไมการเรียงลำดับนี้เป็นโปรแกรมจึงเสถียรเมื่อเปลี่ยนแปลงกับรหัสอะไรซึ่งจะทำให้มันเป็นการเรียงลำดับที่ไม่เสถียร

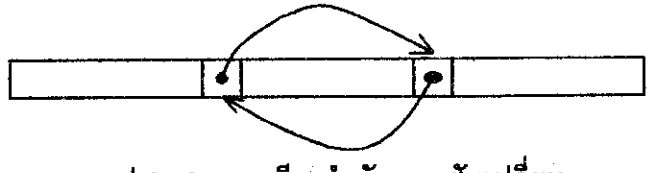
ความหลากหลายของการเรียงลำดับแบบใส่โดยตรงขึ้นอยู่กับการใช้ความรู้ก่อนหน้านั้นของการแจกแจงของการแทนที่ค่าคีย์ในรายการข้อมูล ความพยายามที่จะลดจำนวนการเคลื่อนย้ายข้อมูล , คีย์ถูกใส่ในตำแหน่งการเรียงลำดับซึ่งสัมพันธ์กัน โดยทิ้งให้มีช่องว่างบางช่องระหว่างคีย์เพื่อครอบคลุมคีย์ ที่จะถูกใส่ใน pass ที่ตามมา

ต่อไปจะพิจารณาสมาชิกอื่นๆของครอบครัวของการเรียงลำดับแบบใส่ซึ่งแสดงให้เห็นว่าดีกว่าพฤติกรรม $O(N^2)$

9.4.4 การเรียงลำดับแบบสับเปลี่ยน : การเรียงลำดับแบบฟอง

(An Exchange Sort : The Bubble Sort)

ครอบครัวที่สามของอัลกอริทึมการเรียงลำดับภายใน คือ กลุ่มของการเรียงลำดับแบบสับเปลี่ยน ความคิดพื้นฐานของการเรียงลำดับแบบสับเปลี่ยน คือ ให้เปรียบเทียบคู่ของค่าคีย์และสับเปลี่ยนกัน ถ้ามันไม่ได้อยู่ในตำแหน่งสัมพัทธ์ถูกต้อง(ดูรูป 9-8)



รูป 9-8 การเรียงลำดับแบบสับเปลี่ยน

(The basic idea of an exchange sort is to compare pairs of key values and exchange them if they are not in the proper relative position)

พฤติกรรมการเรียงลำดับดีมาก ให้ผลลัพธ์จากข้อกำหนดที่ฉลาดของคู่ที่ถูกเปรียบเทียบ อย่างไรก็ตามในหัวข้อนี้

ขั้นแรก พิจารณาการเรียงลำดับแบบสับเปลี่ยน ซึ่งฉลาดไม่มาก(not-so-clever) : การเรียงลำดับแบบฟอง คล้ายกับการเรียงลำดับแบบเลือกและแบบใส่ ซึ่งนำเสนอไปแล้วในบทนี้ การเรียงลำดับแบบฟองต้องการการเปรียบเทียบ $O(N^2)$ แต่กระนั้นการเรียงลำดับแบบฟองมีการใช้กันปกติ ซึ่งเราจะตรวจหาการเรียงลำดับแบบสับเปลี่ยนที่ดีกว่าภายหลัง

ความคิดพื้นฐานของการเรียงลำดับแบบฟอง คือ ยอมให้คีย์แต่ละตัวลอยไปยังตำแหน่งถูกต้องของมัน ผ่านชุดของการเปรียบเทียบทีละคู่และสับเปลี่ยนกันกับค่าคีย์ซึ่งประชิดกัน การผ่านแต่ละครั้งให้ผลลัพธ์คือหนึ่งคีย์ ลอยไปยังตำแหน่งสุดท้ายของมัน ในรายการซึ่งเรียงลำดับ

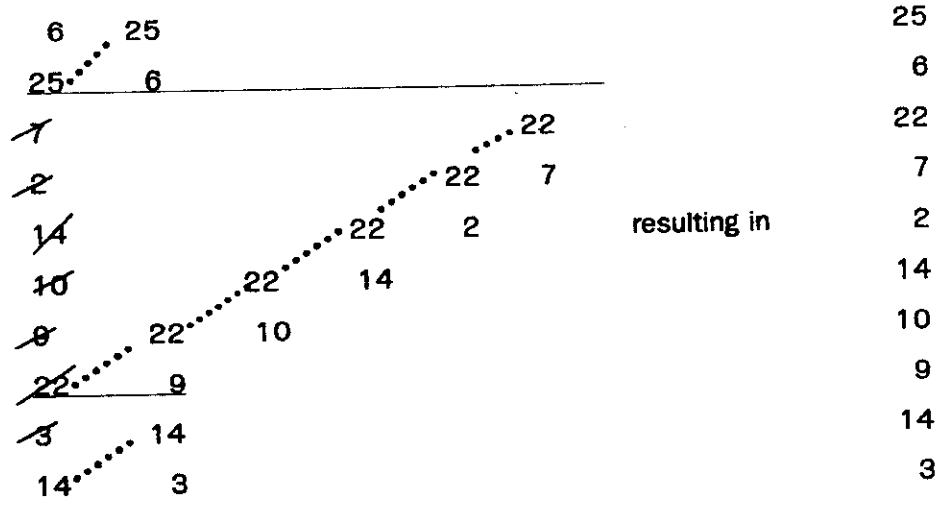
(The basic idea of the bubble sort is to allow each key to float to its proper position through a series of pairwise comparisons and exchanges with adjacent key values. Each pass results in bubbling one key to its find position in the sorted list.)

จงพิจารณารายการไม่เรียงลำดับตัวอย่างของคีย์อีกครั้งหนึ่ง

6
25
7
2
14
10
9
22
3
14

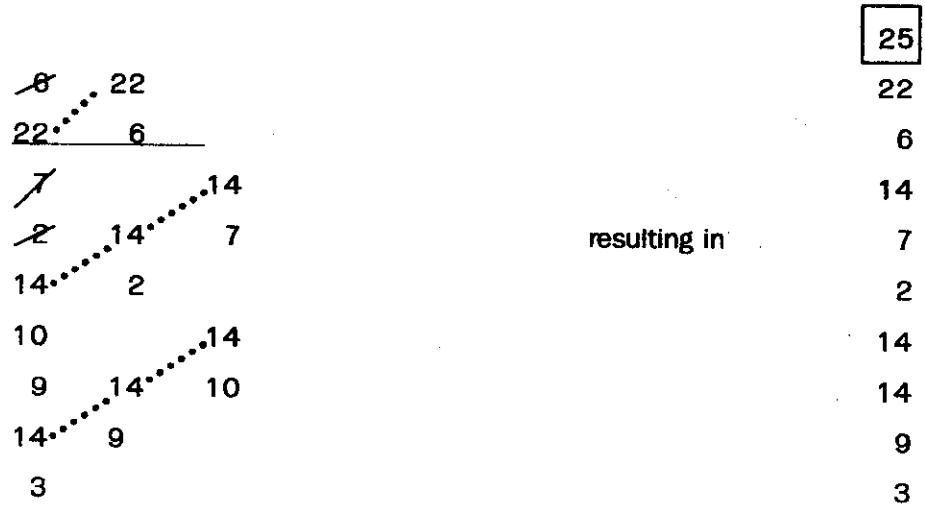
เกิดฟองขึ้น ; ดังนั้นเราจัดรายการให้เป็นแนวตั้ง โดยให้คีย์ตัวแรกอยู่ล่างสุด(at the bottom) คีย์แต่ละตัวเปรียบเทียบกับคีย์ตัวบนของมันและสับเปลี่ยนกัน ถ้าคีย์ตัวบนมีค่าน้อยกว่า เมื่อพบคีย์ซึ่งมีค่าใหญ่กว่า Subject key ตัว Subject key กลายเป็นตัวบนของคู่นั้นและกระบวนการทำต่อเนื่องไป หลังจากการผ่านคีย์ทั้งหมดซึ่งอยู่เหนือตัวสุดท้าย ถูกสับเปลี่ยนต้องเป็นในตำแหน่งสุดท้ายของมัน ซึ่งไม่จำเป็นต้องตรวจสอบใน pass ต่อมา

กิจกรรมของ pass แรก ย้ายขึ้น 14 , 22 และ 25



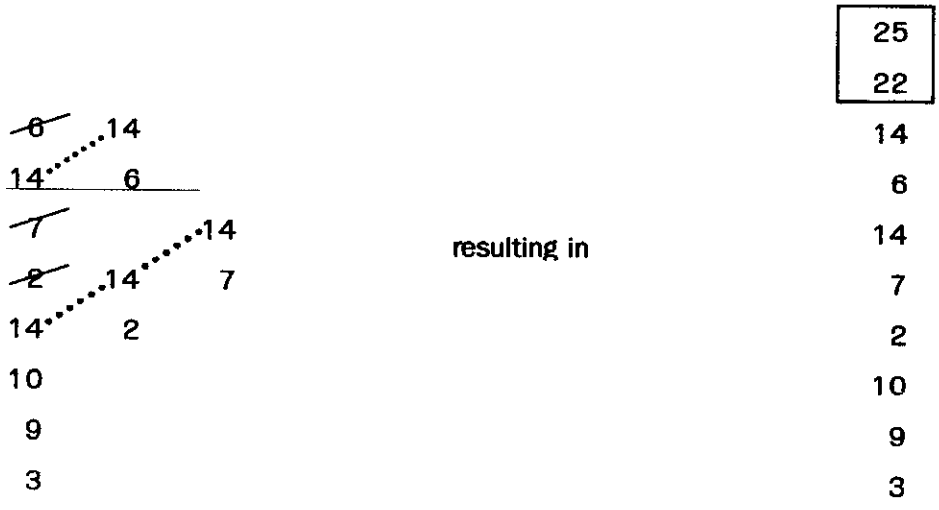
เลข 25 อยู่ในตำแหน่งสุดท้ายของมัน

กิจกรรมของ pass ที่สอง ย้ายขึ้น 14 , 14 และ 22



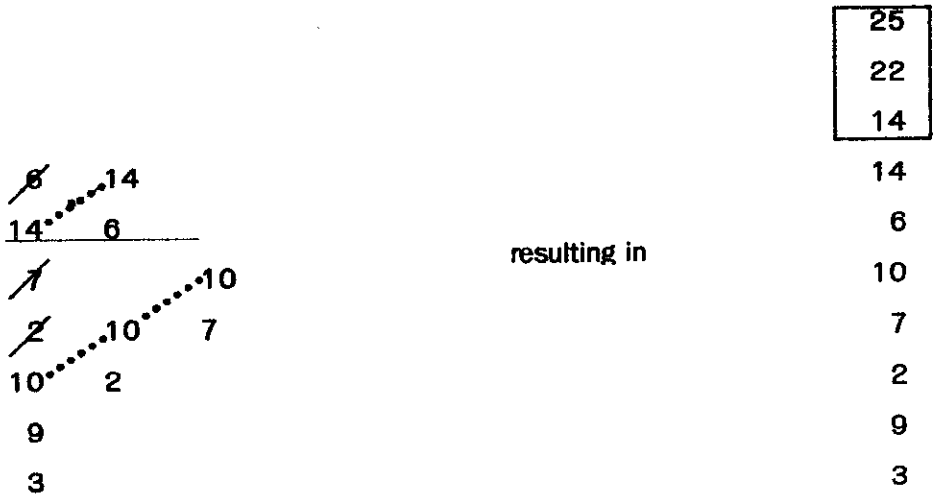
เลข 22 อยู่ในตำแห่งสุดท้ายของมัน

กิจกรรม pass ที่สาม ย้ายขึ้น 14 และ 14



เลข 14 อยู่ในตำแหน่งสุดท้ายของมัน

กิจกรรมของ pass ที่สี่ ย้ายขึ้น 10 และ 14



เช่นนี้เรื่อยไป

การเรียงลำดับแบบฟองจริงๆแล้วมีข้อสังเกตน้อยมาก ยกเว้นเป็นที่รู้จักกันดี(น่าจะเป็นเพราะชื่อของมัน)และโชคไม่ดีใช้กันบ่อย(น่าจะเป็นเพราะว่าค่อนข้างง่ายต่อการนำไปปฏิบัติให้เกิดผล) การกระทำคล้ายการเรียงลำดับเลือกแบบสับเปลี่ยนเล็กน้อย ซึ่งคีย์ตัวเล็กจะจบลงไปตอนล่างของรายการ

9.4.5 การเรียงลำดับแบบเปลี่ยนบางส่วน(การเรียงลำดับอย่างเร็ว)

(The Partition-Exchange Sort Quicksort)

ขณะนี้ให้พิจารณาการเรียงลำดับสับเปลี่ยนที่ดีกว่ามาก : การเรียงลำดับแบบสับเปลี่ยนบางส่วนหรือเรียกว่า Hoare's quicksort หรือ quicksort วิธีนี้เชื่อว่าเป็นของ C.A.R. Hoare (1962)

ความคิดพื้นฐานเบื้องหลังการเรียงลำดับอย่างเร็ว คือ ใช้ผลลัพธ์ของการเปรียบเทียบแต่ละ pass เพื่อเป็นแนวทางของการเปรียบเทียบ pass ถัดไป ระหว่าง comparison pass , คีย์มีการสับเปลี่ยนในลักษณะซึ่งเมื่อเสร็จสิ้น pass รายการมีการแบ่งเป็นส่วนๆเพื่อให้ค่าคีย์ในหนึ่งส่วน(ไม่เรียงลำดับ) ทั้งหมดมีค่าน้อยกว่าค่าคีย์เฉพาะ(a particular key) และค่าคีย์ในส่วนอื่นๆ(ไม่เรียงลำดับ) ทั้งหมดมีค่ามากกว่าค่าคีย์เฉพาะ จากนั้นการเปรียบเทียบ pass ถัดไปจะดำเนินการด้วยส่วนผลลัพธ์สองชุด ปฏิบัติอย่างเป็นอิสระต่อกัน pass ของการเปรียบเทียบคือเฉพาะที่ทำสืบเนื่องเพื่อให้ส่วนต่างๆเล็กลงไปเรื่อยๆ

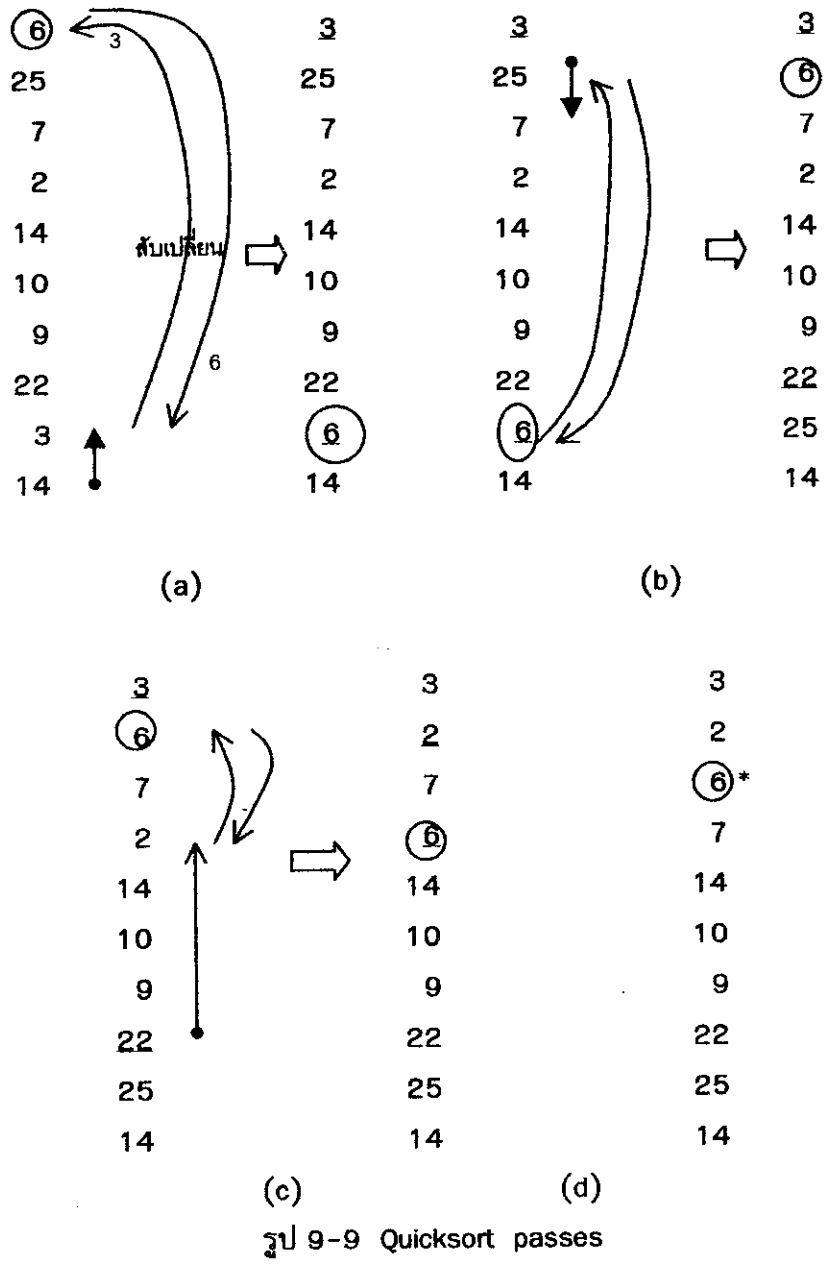
เทคนิคการเรียงลำดับนี้ อธิบายได้ดีที่สุดโดยตัวอย่าง จงพิจารณารายการไม่เรียงอันดับตัวอย่างของคีย์ข้างล่างนี้

6
25
7
2
14
10
9
22
3
14

การเปรียบเทียบ pass แรก มีวัตถุประสงค์ดังนี้

1. แสดงตัวของตำแหน่งสุดท้ายของ subject key ตัวแรก(ในที่นี้คือ 6) ในรายการที่เรียงลำดับ
2. แบ่งรายการออกเป็นสองส่วน ส่วนหนึ่งมีเฉพาะค่าคีย์ที่เล็กกว่า subject key ตัวแรก , อีกส่วนหนึ่งมีเฉพาะค่าคีย์ที่ใหญ่กว่า subject key ตัวแรก

เพื่อให้วัตถุประสงค์ประสบผลสำเร็จ ครั้งแรก เปรียบเทียบค่า subject key กับสมาชิกตัวสุดท้าย(ในที่นี้คือ 14) จากนั้น สมาชิกถัดจากตัวสุดท้าย(ในที่นี้คือ 3) จนกระทั่งพบค่าคีย์ซึ่งควรจะอยู่ก่อนไม่ใช่อยู่หลังค่า subject key จากนั้นให้สับเปลี่ยนตำแหน่งคีย์ตัวนี้กับค่า subject key(รูป 9-9(a)) หลังจากนั้น การเปรียบเทียบดำเนินต่อไปกับ subject key value ตัวเดิม จากทิศทางอื่น : ในที่นี้เปรียบเทียบ 6 กับ 25 เนื่องจากคีย์ทั้งสองตัวนี้ไม่อยู่ในตำแหน่งสัมพันธ์ที่ถูกต้อง ดังนั้นการสับเปลี่ยนเกิดขึ้น(รูป 9-9(b)) ขณะนี้การเปรียบเทียบสวิตช์ทิศทางอีกครั้งหนึ่งจากล่างขึ้นบน , 6 เปรียบเทียบกับ 22 , กับ 9 ,กับ 10 , กับ 14 , และ 1 กับ 2 ซึ่งอยู่ผิดด้านของ 6 การสับเปลี่ยนเกิดขึ้น(รูป 9-9(c)) เปรียบเทียบอีกครั้งสวิตช์ทิศทางจากบนลงล่าง , 6 เปรียบเทียบกับ 7 , ซึ่งอยู่ผิดด้านของ 6 การสับเปลี่ยนเกิดขึ้น ผลลัพธ์ใน pass ที่สมบูรณ์ (รูป 9-9(d)) ค่าคีย์ทุกตัวก่อน 6 มีค่าน้อยกว่า 6 ; ค่าคีย์ทุกตัวหลัง 6 มีค่ามากกว่า 6 ขณะนี้ผลลัพธ์สองส่วนถูกเรียงลำดับเป็นอิสระกัน



รูป 9-9 Quicksort passes

ให้พิจารณาส่วนที่สอง (second partition)

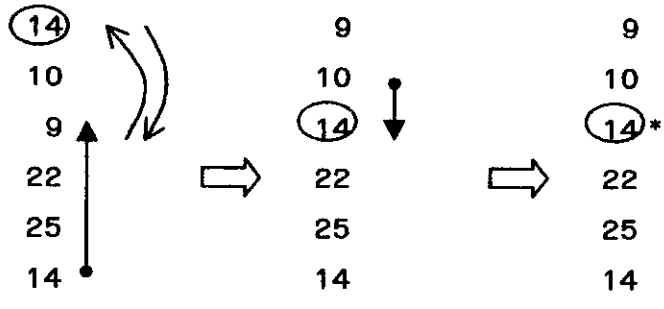
7
14
10
9
22
25
14

Comparison pass ส่วนนี้แสดงว่า 7 อยู่ในตำแหน่งถูกต้องแล้ว ส่วนถัดไปที่จะพิจารณาคือ

14
10
9
22
25
14

ดำเนินการต่อไปโดยมี 14 เป็น subject key(รูป 9-10)

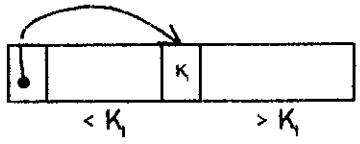
ขณะที่ขนาดของส่วนรายการย่อยหดลง , อัลกอริทึมการเรียงลำดับที่ง่ายกว่า เช่น การใส่หรือการเลือกโดยตรงสามารถนำมาประยุกต์ใช้เพื่อ reduce overhead



รูป 9-10 Final quicksort pass

การกระทำ(Performance)

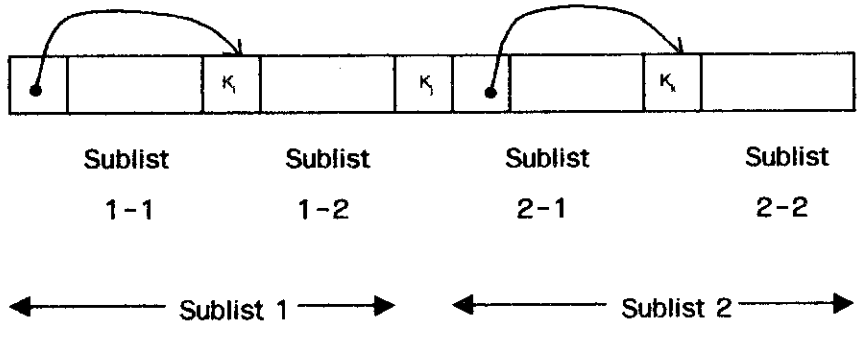
การวิเคราะห์อัลกอริทึม quicksort ค่อนข้างตรงไปตรงมา ขั้นแรกพิจารณากรณีเฉลี่ย สมมติว่า ตำแหน่งสุดท้ายที่ถูกต้องของสมาชิกตัวแรก K_1 อยู่ตรงกลาง unsorted list (รูป 9-11)



รูป 9-11 Quicksort pass

ผลลัพธ์ของ pass แรก ในการแบ่งรายการออก เป็นรายการย่อยสองชุด ขนาดรายการย่อยแต่ละชุด โดยประมาณมี n/2 คีย์ นิยาม " superpass" ให้เป็นการประมวลผลชนิดหนึ่งของรายการทั้งหมด ซึ่ง superpass ครั้งแรกต้องการการเปรียบเทียบ n-1 ครั้ง

superpass ครั้งที่สองเกี่ยวกับการประมวลผลรายการย่อยทั้งคู่ (รูป 9-12)



รูป 9-12 Second quicksort superpass

การประมวลผลของ Sublist 1 ต้องการการเปรียบเทียบโดยประมาณ (1/2)(n-3) ครั้ง เช่นเดียวกับการประมวลผลของ Sublist 2 ที่จริงการประมวลผลของ superpass แต่ละชุดต้องการการเปรียบเทียบ O(n)

โดยเฉลี่ย ต้องการ $\log_2 n$ superpasses เพื่อเรียงลำดับรายการอย่างบริบูรณ์ ดังนั้น quicksort ต้องการเปรียบเทียบโดยเฉลี่ย $O(n\log_2 n)$ ครั้ง นี่คือการกระทำดีที่สุดที่พบในเทคนิคการเรียงลำดับ ข้อควรจำวิธีเชิงเส้นทั้งหมดต้องการการเปรียบเทียบ $O(n^2)$

ในบางกรณี quicksort ต้องการมากกว่า $\log_2 n$ superpasses ในกรณีแย่ที่สุด (worst case) ต้องการ n superpasses กรณีแย่ที่สุดคือ เมื่อรายการเริ่มต้นเรียงลำดับเรียบร้อยแล้ว ไม่ใช่แนวโน้มที่จะลด ครึ่งหนึ่งของจำนวนการเปรียบเทียบที่จำเป็นที่

superpass แต่ละครั้ง การเรียงลำดับ sorted list ที่เริ่มต้นผลลัพธ์คือ การลดการเปรียบเทียบลงหนึ่งครึ่งที่แต่ละ superpass กรณีแย่ที่สุดพฤติกรรมของ $O(n^2)$ คือแย่เท่ากับการกระทำอัลกอริทึมการเรียงลำดับแบบเชิงเส้น อย่างไรก็ตามมันเป็นไปได้ที่ การจัดการ quicksort ไม่เคยใกล้พฤติกรรมกรณีแย่ที่สุดของมัน ดังนั้นวิธีนี้จึงเป็นการเรียงลำดับที่ดีมาก

9.4.6 การเรียงลำดับแบบฮีป (The Heapsort)

การเรียงลำดับแบบควิกซอร์ต (quicksort) โดยเฉลี่ยมีการกระทำ (performance) ที่ปรับดีขึ้นเหนืออัลกอริทึมการเรียงลำดับแบบเชิงเส้น ในหัวข้อนี้จะพิจารณาการเรียงลำดับแบบไม่ใช่เชิงเส้นอีกวิธีหนึ่งเรียกว่า ฮีปซอร์ต(heapsort) ซึ่งการกระทำโดยเฉลี่ยดีเท่ากับควิกซอร์ตและการกระทำดีกว่าควิกซอร์ตในกรณีแย่ที่สุด การเรียงลำดับแบบฮีปมีการกระทำโดยตลอด ดีกว่าการเรียงลำดับภายในวิธีอื่น ๆ ซึ่งได้นำเสนอมาแล้ว อย่างไรก็ตาม วิธีนี้เมื่อเป็นโปรแกรมค่อนข้างจะซับซ้อน การเรียงลำดับแบบฮีปถูกพัฒนาในปี ค.ศ 1964 โดย j.w.j. williams

ฮีปซอร์ตมีหลักการโดยใช้ต้นไม้แบบทวิภาคชนิดพิเศษ (เรียกว่า ฮีป) เพื่อเป็นโครงสร้างกระบวนการของการเรียงลำดับ โครงสร้างการแตกกิ่งของต้นไม้ เก็บ (keep) จำนวนของการเปรียบเทียบที่ต้องใช้อยู่ที่ $O(n\log_2 n)$

ฮีปซอร์ตมีสองขั้นตอนดังนี้ :

1. การสร้างฮีป (Creation of the heap)
2. การประมวลผลของฮีป (Processing of the heap)

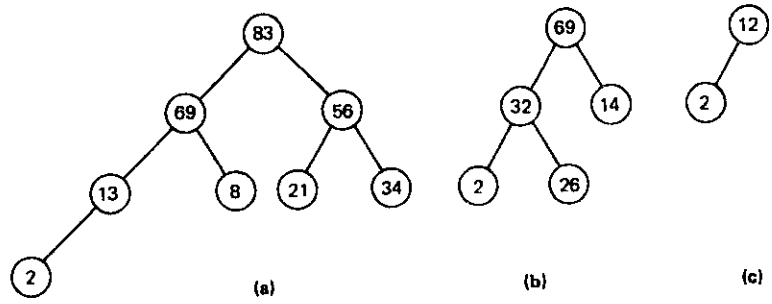
ในขั้นตอนแรก คีย์ซึ่งยังไม่เรียงลำดับ (unsorted keys) ถูกนำมาใส่ในต้นไม้แบบทวิภาคในวิธีซึ่งมันประกอบขึ้นเป็นฮีป

โครงสร้างฮีป (heap Structure)

ฮีปขนาด n หมายถึงต้นไม้แบบทวิภาคที่มีโหนด n ตัวซึ่งเกาะติดกับข้อบังคับ สองข้อข้างล่างนี้

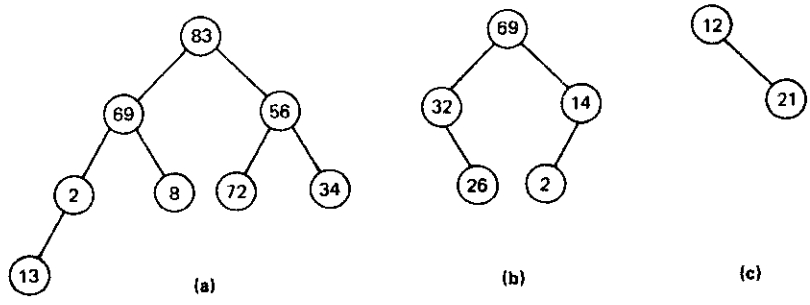
1. ต้นไม้แบบทวิภาคเป็นต้นไม้เกือบจะบริบูรณ์ นั่นคือ มี จำนวนเต็ม k ซึ่ง :
   (a) จุดแตกกิ่งทุกตัวของต้นไม้อยู่ที่ระดับ k หรือ ระดับ k+1 และ
   (b) ถ้าโหนดใดมี โหนดล่างทางขวา (right descendant) อยู่ที่ระดับ k+1 ดังนั้นโหนดตัวนี้มีโหนดล่างซ้าย (left descendant) อยู่ที่ระดับ k+1 ด้วย
2. คีย์ในโหนดถูกจัดการโดยที่ โหนด i แต่ละตัว $k_i \leq k_j$ เมื่อโหนด j คือ father ของโหนด i

ข้อบังคับข้อแรกมีหมายความว่า ระดับของฮีปถูกใส่จากซ้ายไปขวาและโหนดจะไม่ถูกใส่ที่ระดับใหม่ จนกว่าระดับก่อนหน้านั้นเต็ม ต้นไม้แบบทวิภาคของรูป 9-13 ทั้งหมดนี้มีคุณสมบัติเป็นฮีป โปรดสังเกตว่าต้นไม้เหล่านี้ไม่มีคุณสมบัติเป็นต้นไม้ค้นหาแบบทวิภาค (binary search trees) เพราะว่าการแวะผ่านแบบตามลำดับจะไม่เป็นลำดับซึ่งโหนดเรียงค่าคีย์จากน้อยไปมาก



รูป 9-13 ตัวอย่างฮีป (Example heaps)

ต้นไม้แบบทวิภาคของรูป 9-14 ไม่ใช่ฮีป เพราะว่าข้อบังคับการใส่คีย์ถูกฝ่าฝืนที่โหนด 2-13 และโหนด 56-72 ในต้นไม้ (a); ข้อบังคับเป็นต้นไม้เกือบจะบริบูรณ์ถูกฝ่าฝืนบนระดับ 2 ของต้นไม้ (b); ข้อบังคับการใส่คีย์และข้อบังคับการเป็นต้นไม้เกือบจะเต็มถูกฝ่าฝืนทั้งคู่ในต้นไม้ (c)



รูป 9-14 ต้นไม้แบบทวิภาคซึ่งไม่ใช่ฮีป

การสร้างฮีป (Creating a Heap)

ขั้นแรกจงพิจารณาขั้นตอนการสร้างฮีปของฮีปซอร์ต ในขั้นตอนนี้เราผ่านแบบลำดับตลอดคีย์ซึ่งยังไม่เรียงลำดับ ใส่คีย์ในฮีป ขนาดของฮีปใดขึ้นเมื่อใส่คีย์ตัวใหม่แต่ละตัว การสร้างฮีป ขนาด i , คีย์ (k_i) คือตัวที่ i ถูกใส่ในฮีปที่มีอยู่แล้วขนาด i-1 โหนดตัวแรกจะอยู่ในตำแหน่งโดยที่เป็นไปตามข้อบังคับต้นไม้เกือบจะบริบูรณ์ จากนั้นค่าของ k_i เปรียบเทียบกับค่าคีย์ของnode' s father ถ้า k_i มีค่ามากกว่าดังนั้นให้สลับที่กันระหว่าง content ของโหนดใหม่ (new node) กับโหนดพ่อ (father node)

กระบวนการเปรียบเทียบ-สลับที่นี้ ทำซ้ำๆ กันจนกระทั่งไม่มีค่าคีย์ของ father node น้อยกว่า k_i หรือ k_i เป็นรากของต้นไม้ หลังจากนั้นต้นไม้จะมีคุณสมบัติของการเป็นฮีปขนาด i พูดเป็นทางการมากขึ้น ใน Pascal การสร้างฮีปขนาด i โดยการใส่คีย์ (ชื่อ newkey) ให้กับฮีป (ในที่นี้รู้จักว่าเป็นตัวแปรส่วนกลางชื่อ key และเก็บในแถวลำดับ) ของขนาด i-1 (เมื่อ $i \geq 1$)

โปรซีเดอร์เขียนดังนี้

```
procedure crheapi (i, newkey : integer);
var father, temp, next : integer;
 begin
        next := I ;
        father := next div 2;
        key[next] := newkey ;
        while (next <> 1 and key[father] <= key[next])
        do begin { interchange father and son}
                temp := key[father] ;
                key[father] := key[next] ;
                key[next] := temp ;
                { advance up tree }
                  next := father;
                  father := next div 2
            end ;
end;
```

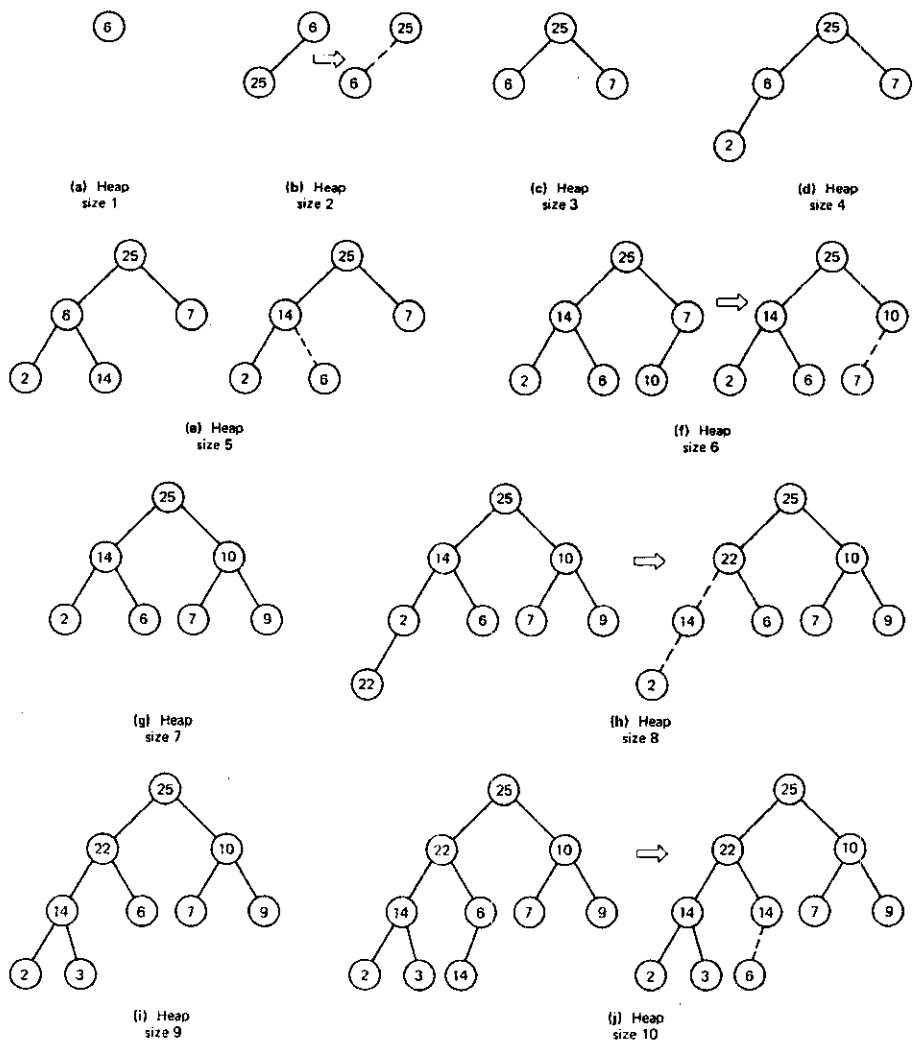
โปรซีเดอร์นี้ถูกเรียกสำหรับการใส่ในแต่ละครั้งให้กับฮีป รูป 9-15 แสดงการสร้างฮีป หนึ่งรูปเพื่อให้ประกอบด้วยรายการของคีย์ที่ยังไม่เรียงลำดับ :

6 , 25 , 7 , 2 , 14 , 10 , 9 , 22 , 3 , 14

เส้นประ (dashed line) ในรูปแสดงว่าโหนดที่ปลายสุดของด้าน (edge) มีการสับเปลี่ยน

ฮีปนี้ถูกแทนที่ในแถวลำดับ ดังที่แสดงในรูป 9-16 โปรดสังเกตว่าโหนด i คือ father ของโหนด 2i และโหนด 2i+1 ดังนั้น เพราะว่าต้นไม้คือฮีป

$key(i) \leq key(\frac{i}{2})$ เมื่อใช้การหารจำนวนเต็ม



รูป 9-15 ขั้นตอนต่าง ๆ ในการสร้างฮีปตัวอย่าง

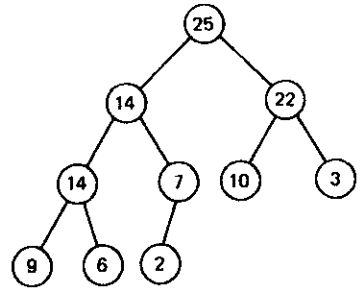
Subscript	1	2	3	4	5	6	7	8	9	10
Key	25	22	10	14	14	7	9	2	3	6

รูป 9-16 การแทนที่แถวลำดับของฮีปในรูป 9-15 (j)

ถ้าเริ่มต้นด้วย keys ในอันดับแตกต่างกันในรายการไม่เรียงลำดับ (unsorted list) ดังนั้นฮีปผลลัพธ์จะแตกต่างกันด้วย ตัวอย่างเช่น รายการไม่เรียงลำดับ

9 , 14 , 10 , 22 , 7 , 25 , 3 ,14 , 6 , 2

ผลลัพธ์ คือฮีปซึ่งแสดงในรูป 9-17



รูป 9-17 ฮีปอีกชุดหนึ่งสำหรับเซตของคีย์ชุดเดียวกับคีย์ในรูป 9-15 นำเสนอในอันดับแตกต่างกัน

การประมวลผลฮีป (Processing the Heap)

ข้อควรจำวัตถุประสงค์ของฮีปซอร์ต คือการสร้างรายการที่เรียงลำดับของคีย์ และณ. จุดนี้คือฮีปของคีย์ ขั้นตอนการประมวลผลของฮีปซอร์ต คือแวะผ่านฮีปในวิธีซึ่งผลลัพธ์จะได้คีย์ที่เรียงลำดับ โปรดสังเกตว่าคีย์ตัวที่มีค่าใหญ่สุด ในฮีปปกติจะอยู่บนสุด ขั้นตอนการประมวลผลคือสร้างตามความเป็นจริงนี้ หลังจากสร้างฮีปแล้วสมาชิกตัวบนสุดจะถูกลบออกอีกครั้งหนึ่งและทำเรื่อยไปจนกระทั่งฮีปผลลัพธ์มีขนาดเท่ากับ 0

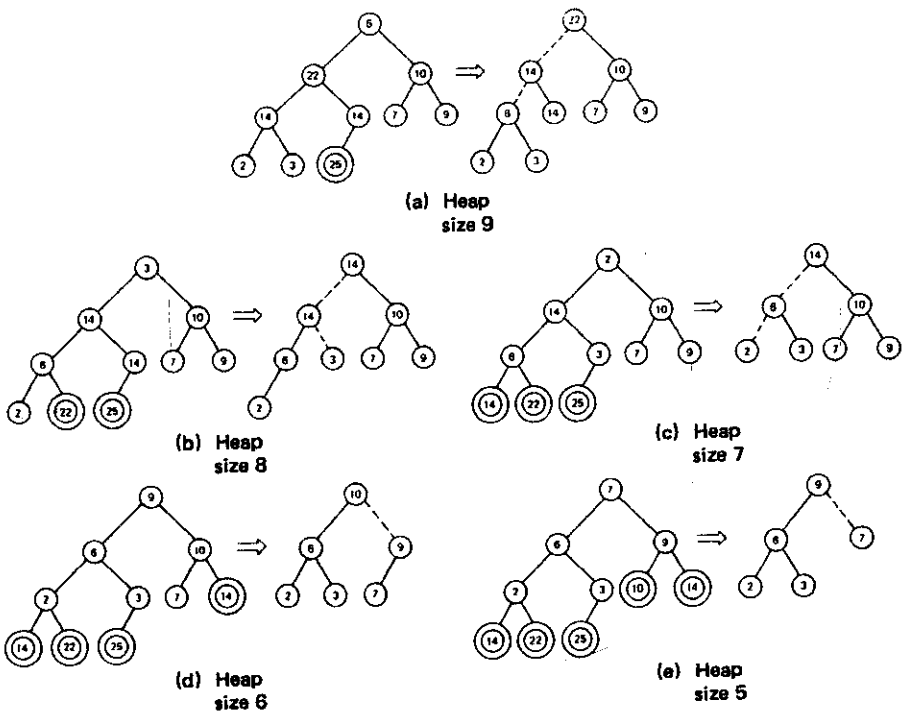
พูดเป็นทางการมากขึ้น โปรซีเดอร์ Pascal เพื่อประมวลผลฮีปขนาดเท่ากับ n จะเป็นดังนี้เราเอาข้อดีของความจริงที่ว่า ฮีปยังคงเก็บในแถวลำดับเรียกว่า key (ผลลัพธ์จากการใช้โปรซีเดอร์ crheapi) ; ค่าคีย์ตัวใหญ่สุดอยู่ที่บนสุดของ heap : key[1] เราย้ายค่านั้น

อย่างสืบเนื่องไปตอนท้ายสุดของแถวลำดับ (เป็นครั้งแรก key[n] ผ่านส่วนวนซ้ำ) จากนั้น ปรับแถวลำดับให้เป็นฮีปขนาด n-1 ที่ตอนจบคีย์ในแถวลำดับเรียงลำดับ ; key[1] คือค่าเล็กสุดและ key[n] คือค่าใหญ่สุด

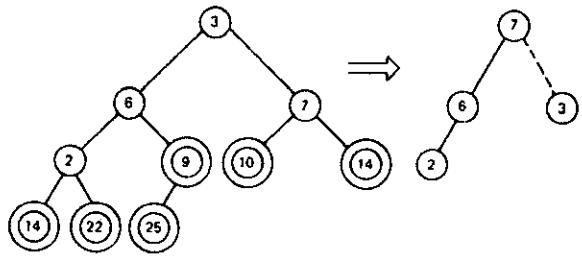
```
procedure prheapn ;
 var father, son, last , waslastkey : integer;
 for last := n downto 2
 do begin { move root key down to last place}
        waslastkey := key[last];
        key[last] := key[1];
        { adjust tree to heap of size last-1}
         father := 1;
        { find larger of root's sons}
         if (last - 1 >= 3 ) and (key[3] > key[2])
         then son := 3
         else son := 2 ;
        { move keys upward until find place}
        { for saved waslastkey}
         while (son <= last - 1 and key[son] > waslastkey )
         do begin key[father] := key[son];
                  father := son ;
                  son := father * 2;
                  { find larger of father 's sons}
                 if (son + 1 <= last - 1) and (key[son + 1] > key[son])
                 then son := son + 1
           end;
          key[father] := waslastkey
end;
```

รูป 9-18 แสดงให้เห็นการประมวลผลของการสร้างฮีปในรูป 9-15 โหนดที่มีวงกลมสองวงล้อมรอบ หมายถึงโหนดซึ่งมีการย้ายไปยังตำแหน่งสุดท้ายของมันในแถวลำดับ และไม่มีส่วนยาวกว่าของฮีป เส้นประแสดงว่าโหนดที่ตอนท้ายสุดของด้านมีการสับเปลี่ยน ขณะการปรับต้นไม้ให้เป็นฮีปอีกครั้งหนึ่ง หลังจากผ่าน n-1 ครั้ง (ในที่นี้คือครั้งที่ 9) คีย์ถูกอ่านแบบลำดับ (1 ถึง n) จากแถวลำดับที่เก็บฮีปและจะเป็นการเรียงลำดับอย่างถูกต้อง

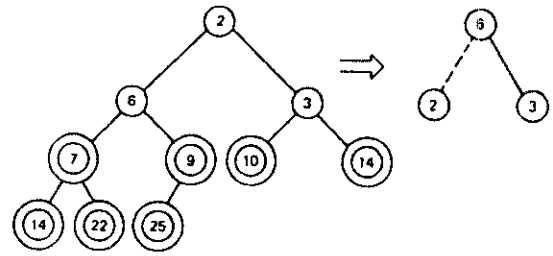
ด้วยอัลกอริทึมควิกซอร์ตการวิเคราะห์ของอัลกอริทึมฮีปซอร์ต ค่อนข้างตรงไปตรงมา ระหว่างขั้นตอนการสร้างฮีป การใส่ของคีย์ตัวที่ i ต้องเปรียบเทียบ O ($\log_2 i$)และสับเปลี่ยนในกรณีแย่ที่สุดเมื่อคีย์มาถึงการเรียงอันดับเรียบร้อยแล้ว



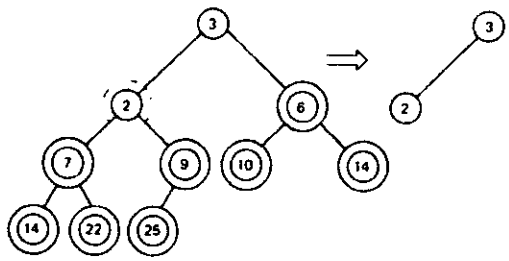
รูป 9-18 การประมวลผลฮีป รูป 9-15



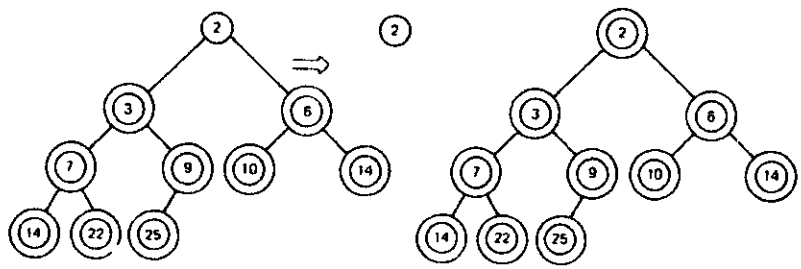
(f) Heap size 4



(g) Heap size 3



(h) Heap size 2



(i) Heap size 1

รูป 9-18 (ต่อ)

การใส่ของคีย์บนระดับที่ k สามารถบังคับการเปรียบเทียบและสับเปลี่ยนกับค่ามากที่สุดของ k คีย์อื่น ๆ ไปตามกิ่งไปยังรากของฮีป ในทำนองเดียวกันระหว่างขั้นตอนการประมวลผลฮีป การปฏิบัติของฮีปต้องใช้O (log $_2$ i) การเปรียบเทียบและสับเปลี่ยนในกรณีแย่ที่สุด การเคลื่อนย้ายสามารถถูกกระทำเฉพาะหนึ่งของกิ่งของฮีป เพราะฉะนั้นบนค่าเฉลี่ย จำนวนการเปรียบเทียบและสับเปลี่ยนที่ต้องใช้คือ

$$\frac{1}{2}\sum_{i=2}^{n} \log_2 i \quad + \quad \frac{1}{2}\sum_{i=2}^{n} \log_2 i$$

ซึ่งคือ (n-1)log $_2$ n กรณีแย่ที่สุดคือไม่เป็นกรณีแย่มากกว่ากรณีเฉลี่ย เพราะฉะนั้นฮีปซอร์ตจึงถูกรับประกันเป็น O (n log $_2$ n) สำหรับ n ขนาดใหญ่ ความซับซ้อนของอัลกอริทึมคือมีน้ำหนักกว่า (outweighed) โดยประสิทธิภาพของการเรียงลำดับ

9.4.6 การเรียงลำดับแบบแข่งขัน (The Tournament Sort)

การเรียงลำดับ tree-based ที่สำคัญอีกวิธีหนึ่งคือการเรียงลำดับแบบแข่งขัน การเรียงลำดับนี้บางครั้งเรียกว่าการเรียงลำดับแบบ tree-selection ผู้พัฒนาคือ E. H. Friend (1956) กระบวนการเรียงลำดับแบบแข่งขันดูคล้ายกับต้นไม้การแข่งขันแบบแพ้คัดออกปกติใช้จัดการกีฬาจับคู่ (โดยเฉพาะกีฬาเทนนิส ปิงปอง เป็นต้น)

แข่งขันทีละคู่ (pair-wise competition) ระหว่างผู้เล่นจนกระทั่งได้ผู้ชนะคนสุดท้าย

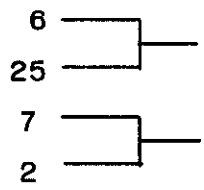
การเรียงลำดับแบบแข่งขัน ใช้บ่อยกว่าการเรียงลำดับภายในอื่น ๆ ผู้ผลิตซอฟต์แวร์จำนวนมากรวมสิ่งนี้ไว้ในโปรแกรมสำเร็จการเรียงลำดับของแฟ้ม (file-sorting packages) เราจะอภิปรายการทำงานของการเรียงลำดับแบบแข่งขันในที่นี้

จงพิจารณารายการไม่เรียงลำดับของคีย์ซึ่งเราได้เคยใช้ในตัวอย่าง :

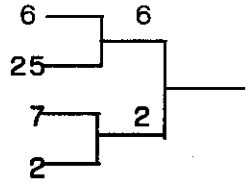
6 , 25 , 7 , 2 , 14 , 10 , 9 , 22 , 3 , 14

เพิ่มเติมด้วย 8 , 12 , 1 , 30 , 13

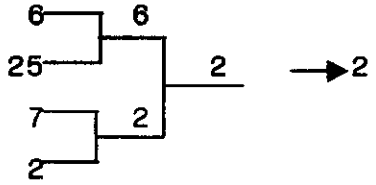
สมมติว่าเนื้อที่มีขีดจำกัดบังคับให้ในแต่ละครั้งในหน่วยความจำมีเพียงสี่คีย์เท่านั้น การเรียงลำดับแบบแข่งขันจับคู่คีย์สี่ตัวนี้เป็น 2 matches ดังนี้



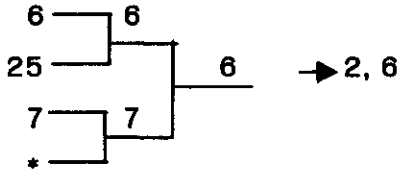
สมมติว่าเราต้องการกระทำงานเรียงลำดับขึ้น (ascending order) ผู้ชนะของ matches เหล่านี้คือ 6 และ 2 ซึ่งจะจับกันเป็นคู่สุดท้าย



ครั้งนี้ 2 เป็นผู้ชนะและเป็นเอาต์พุต คีย์ตัวแรกจากการเรียงลำดับ



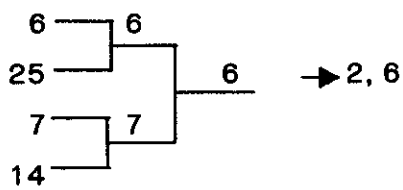
Pass ถัดไปของการเรียงลำดับแบบแข่งขันจะให้คีย์ตัวที่สองของรายการเรียงลำดับ 2 ไม่สามารถมีส่วนเข้าร่วมใน pass ครั้งที่สองนี้ เราจะเล่นได้กับ 7 ใครซึ่งชนะใน match แรกจากนั้นจะแข่งกับ 6



สิ่งนี้จะให้ผลลัพธ์ในการเรียงลำดับ tree-selection บริสุทธิ์ อย่างไรก็ตาม เรานำผู้สมัครอีกชุดหนึ่งเข้าสู่การแข่งขันจากรายการไม่เรียงลำดับส่วนที่เหลือ :

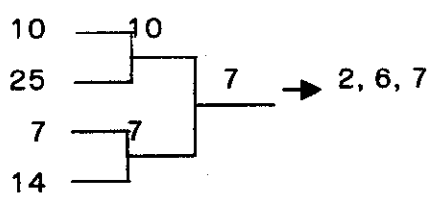
14, 10, 9, 22, 3, 14

จากนั้นเราเล่น pass ที่สองของการแข่งขัน

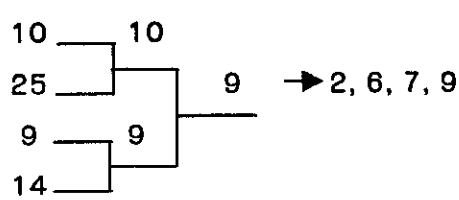


นโยบายการแทนที่นี้จะทำให้การเรียงลำดับสร้างสายอักขระเรียงลำดับยาวขึ้น ถ้าเราไม่ได้นำผู้เล่นคนใหม่เข้ามา ขณะนี้การเรียงลำดับของเราถูกต้องมากขึ้น ชื่อ tree replacement selection sort

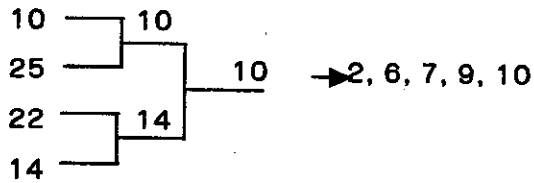
จากนั้น pass ที่สามคือ



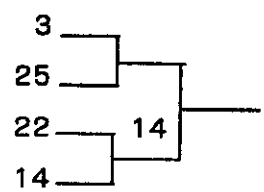
Pass ที่สี่คือ



และ pass ที่ห้าคือ



Pass ที่หกนำ 3 เข้ามาในต้นไม้



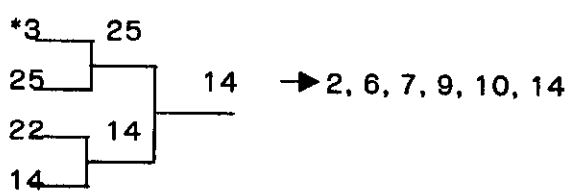
ถ้า 3 เข้ามาเล่น ลำดับที่เรียงแล้วของรายการเอาต์พุตจะต้องถูกแยกออกจากกัน เพื่อป้องกันสิ่งนี้การเรียงลำดับแบบ replacement – selection เรียกกฎต่อไปนี้ :

If $key_{new} < Key_{lastout}$

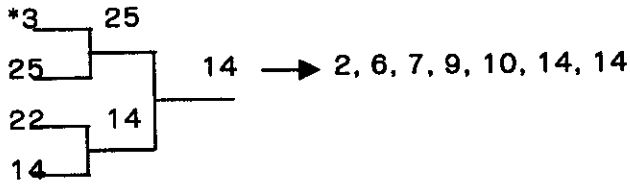
Then Key_{new} ถูกใส่เข้าไปในต้นไม้แต่เป็น

Temporarity disqualified

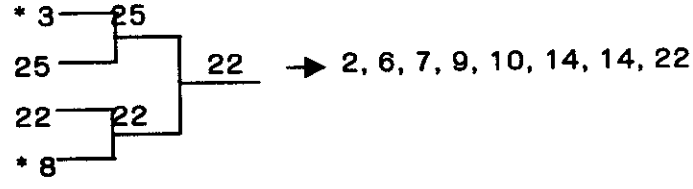
เราทำเครื่องหมายสมาชิกที่ disqualified โดยใส่เครื่องหมาย * ดังนั้นผลลัพธ์ pass ที่หก คือ



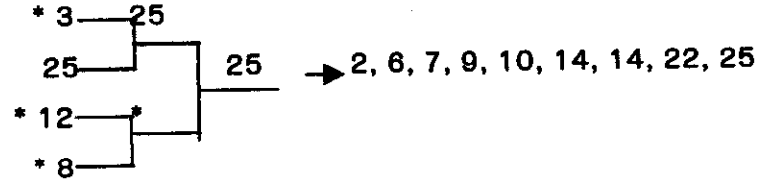
คีย์ตัวถัดไปของรายการไม่เรียงลำดับ (14) ถูกนำเข้ามา และสามารถเล่นได้เพราะว่ามันมีค่าไม่น้อยกว่า 14



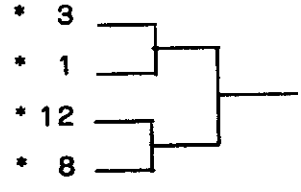
Pass 8 นำคีย์ที่มีค่าเท่ากับ 8 ซึ่งเป็น disqualified เพราะว่ามันน้อยกว่า 14 แต่ผู้เข้ามา (entrants) อื่น ๆ สามารถเล่นได้



Pass 9 ผลลัพธ์คือ



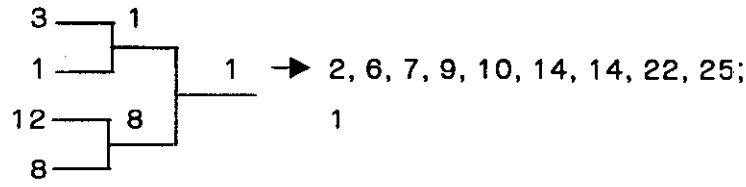
Pass 10 นำ 1 เข้ามา



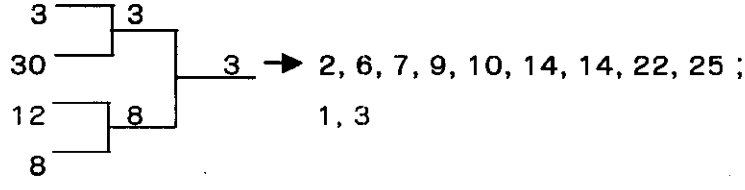
ขณะนี้ entries ทั้งหมดถูกตัดสิทธิ์ (disqualified) โปรดสังเกตว่าถึงแม้ว่าจะยอมให้เพียงผู้เล่นสี่คนเข้าในการแข่งขัน ณ เวลาที่กำหนดให้ใด ๆ ก็ได้สายอักขระเรียงลำดับที่เป็น

ผลลัพธ์ประกอบด้วยคีย์ 9 ตัว ขณะนี้เราลบ disqualifications และเริ่มต้นอีกครั้งหนึ่ง ครั้งนี้เราจะสร้างสายอักขระเรียงลำดับชุดที่สอง

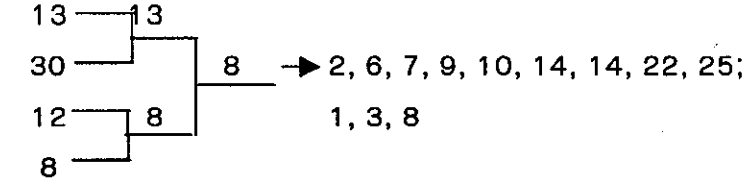
Pass 10 จะเป็นดังนี้



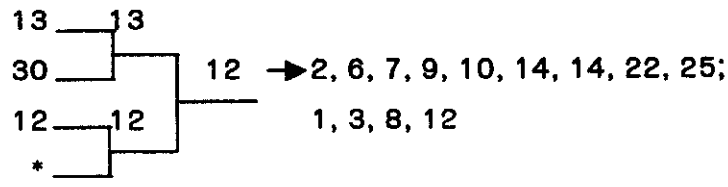
Pass 11 นำ 30 เข้ามา



Pass 12 คือ



ขณะนี้รายการอินพุตเป็น empty ดังนั้นเราต้องใส่ placeholders เข้าไปในต้นไม้ และการแข่งขันเป็นการเล่นที่จะเสร็จสิ้น Pass 13 ผลลัพธ์เป็นดังนี้



Pass 14 และ 15 ต้นไม้เป็น empty
ตัวอย่างนี้ให้ผลลัพธ์เป็นรายการเรียงลำดับสองชุด :

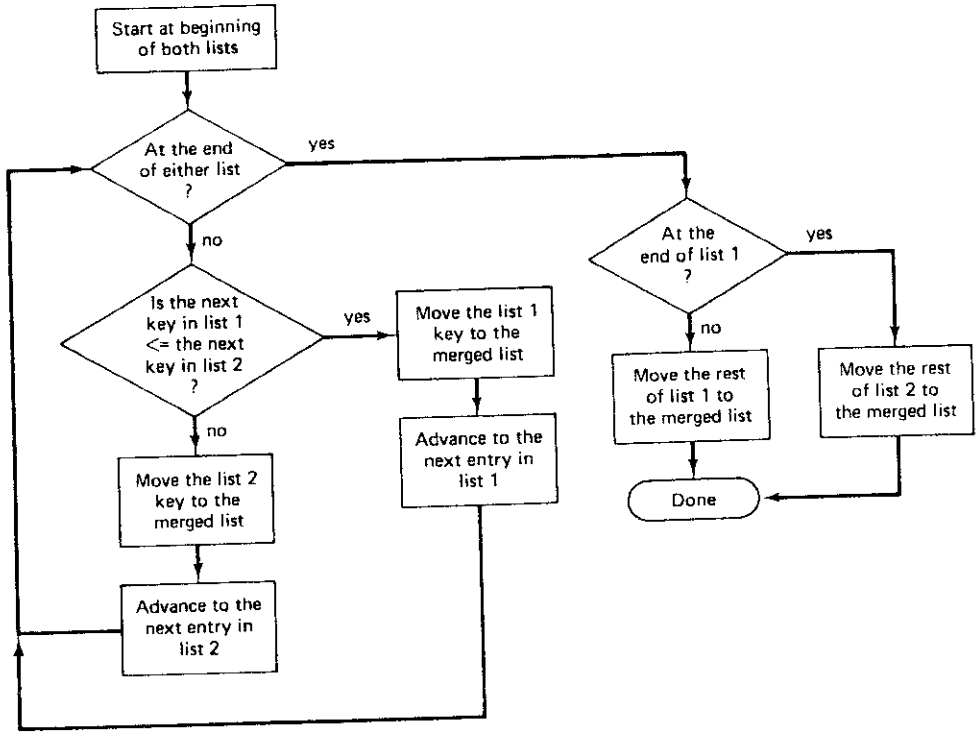
2, 6, 7, 9, 10, 14, 14, 22, 25 :
1, 3, 8, 12, 13, 30

ขณะนี้สายอักขระสองชุดนี้สามารถนำมาผสาน(merged) กันให้เป็นสายอักขระเอาต์พุต เรียงลำดับหนึ่งชุด สมาชิกตัวแรกของสายอักขระแต่ละชุดนำมาเปรียบเทียบกันและสมาชิกตัวที่มีค่าต่ำกว่าจะเป็นเอาต์พุตและลบออกจากการพิจารณาต่อไป การผสานสองทาง (two-way merge) อย่างง่ายนี้แสดงให้เห็นในผังงานรูป 9-19 ซึ่งเขียนเป็นภาษา Pascal ดังนี้แถวลำดับ Key1 และ Key2 เก็บรายการเรียงลำดับแล้วตอนเริ่มต้น ; แถวลำดับ mergedเก็บรายการผสานที่เป็นผลลัพธ์ n1 และ n2 เป็นดรรชนีล่างของสมาชิกตัวถัดไปใน key1 และ key2 ตามลำดับ next คือดรรชนีล่างแสดงตำแหน่งสำหรับสมาชิกตัวถัดในรายการที่ผสานแล้ว (merged list)

```
begin
        next := 1;
        n1 := 1;
        n2 := 1;
        while (n1 <= size1) and (n2 <= size2)
        do begin if (key1[n1] <= key2[n2])
                then begin
                        merged[next] := key1[n1];
                                n1 := n1 + 1
                        end;
                else begin
```

```
                    merged[next]:= key2[n2];
                        n2 := n2 + 1
                end;
            next := next + 1
        end;
    while (n1 <= size1) {fill with remaining list 1}
    do begin
            merged[next] := key1[n1];
                n1 := n1 + 1;
            next := next + 1
        end;
    while (n2 <= size2) {fill with remaining list 2}
    do begin
            merged[next] := key2[n2];
                n2 := n2 + 1;
            next := next + 1
        end;
end;
```

ตัวอย่างถูกจัดการเพื่อให้รายการเรียงลำดับสองชุดสะดวกที่จะให้ผลลัพธ์ในทาง เป็นจริงรายการเรียงลำดับหลายชุดอาจเป็นผลลัพธ์ รายการเหล่านี้สามารถนำมาผสานกัน โดยลำดับของการผสานสองทาง (two-way merges) หรือโดยการผสานอันดับที่สูงกว่าครั้งหนึ่งมี รายการ 3 ชุด หรือ รายการ 4 ชุด (tree or four lists at a time)



รูป 9-19 อัลกอริทึมสำหรับ a two-way merge

การกระทำ (Performance)

การเรียงลำดับแบบแข่งขันกระทำงานได้ดีอย่างไร ขั้นแรกจงพิจารณากรณีซึ่งต้นไม้การแข่งขันใหญ่มากพอที่จะเก็บคีย์ทั้งหมด n ตัวเพื่อถูกเรียงลำดับให้ระบุผู้ชนะคนแรก

$$\frac{n}{2} + \frac{n}{4} + \cdots + \frac{n}{2^k} = n \sum_{i}^{k} \frac{1}{2^i}$$

การเปรียบเทียบต้องถูกกระทำ เมื่อ k หมายถึงจำนวนระดับในต้นไม้การแข่งขัน $k = \log_2 n$ นิพจน์นี้คือ O(n) หลังจากการ setup เริ่มต้นนี้ของต้นไม้, การเปรียบเทียบ k ต้องใช้เพื่อปรับต้นไม้และระบุถึงผู้ชนะคนถัดไป ดังนั้นจำนวนของการเปรียบเทียบที่ต้องใช้ประมวลผลต้นไม้ทั้งหมดคือ $O(n\log_2 n)$ โปรดสังเกตว่าผลลัพธ์คือสายอักขระหนึ่งชุดเท่านั้นเมื่อผู้เล่นทั้งหมดเหมาะสมเข้าไปในต้นไม้การแข่งขัน

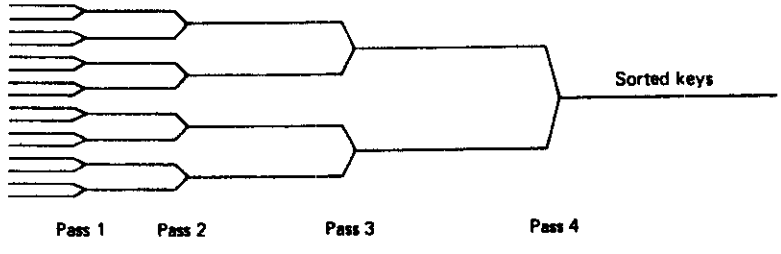
ขณะนี้จงพิจารณากรณีซึ่งต้นไม้การแข่งขันสามารถเก็บคีย์ได้เพียงP ตัวในแต่ละเวลาที่กำหนดให้ , $p << n$ ในแต่ละ pass ผ่านตลอดต้นไม้ต้องใช้การเปรียบเทียบ $\log_2 P$ เพื่อระบุถึงผู้ชนะคนถัดไป ด้วยเหตุนี้การเปรียบเทียบ $n \log_2 p$ ต้องใช้เพื่อสร้างรายการย่อยเรียงลำดับ

จากนั้นรายการย่อยเหล่านี้ต้องถูกผสานกันซึ่งสามารถแสดงให้เห็นได้ว่า expected length ของรายการเรียงลำดับแต่ละชุด เป็นผลลัพธ์จากการเรียงลำดับแบบแข่งขัน ด้วยการแทนที่ของ entries เท่ากับ 2 * P ดังนั้นคีย์ n ตัวถูกจัดสรรให้เป็น $\frac{n}{2p}$ lists เพื่อใช้ two-way merge ต้องใช้ $\log_2\left(\frac{n}{2}\right)$ passes ตัวอย่างเช่น ถ้า n = 256 และ p = 8 ดังนั้น expected number ของรายการย่อยคือ 16 การผสาน 16 รายการ, ครั้งละ 2 ชุด ต้องใช้ $\log_2 16$ นั่นคือ 4 passes (ดูรูป 9-20) แต่ละ pass ต้องเปรียบเทียบ n ครั้งดังนั้น ต้องเปรียบเทียบ $n \log_2(n/2p)$

ในขั้นตอนผสาน รวมทั้งสิ้นโดยเฉลี่ยต้องใช้จำนวนการเปรียบเทียบเท่ากับ

$$n \log_2 p + n \log_2(n/2p)$$

ซึ่งคือ $O(n \log_2 n)$ สำหรับ $n>>p$



รูป 9-20 ตัวอย่าง passes สำหรับการผสานรายการย่อย 16 ชุด

บทสรุป (Summary)

บทนี้เริ่มต้นด้วยการอภิปรายเทคนิคการค้นหาแบบเชิงเส้น เราแสดงให้เห็นว่าการค้นหาแบบลำดับพื้นฐานสามารถปรับปรุงให้ดีขึ้น โดยการเรียงอันดับคีย์ในรายการของความถี่การเข้าถึงที่ลดลง รายการเชิงเส้นอาจจัดระเบียบใหม่ตัวมันเองอย่างพลวัตโดย (1) การย้ายคีย์ไปข้างหน้ารายการเมื่อมันถูกร้องขอ(requested) หรือโดย (2) สับเปลี่ยนคีย์กับตัวหน้าที่ใกล้กับมันเมื่อถูกร้องขอ ดังนั้นการย้ายทีละเล็กทีละน้อยไปข้างหน้าจำนวนครั้งของการเปรียบเทียบซึ่งใช้สำหรับการค้นหาที่ซึ่งไม่ประสบผลสำเร็จอาจทำให้ลดลงโดยรายการที่เรียงลำดับด้วยค่าคีย์ การค้นหาแบบลำดับทั้งหมดเป็น O(n)

จากนั้นเทคนิคการค้นหาแบบทวิภาคถูกอภิปรายขึ้นเป็นวิธีของการได้ $O(\log_2 N)$ การกระทำการค้นหาจากรายการเชิงเส้น สิ่งซึ่งต้องได้มาก่อนสำหรับการประยุกต์ใช้เทคนิคการค้นแบบทวิภาคคือระเบียนในรายการต้องเรียงลำดับโดยค่าคีย์ที่ค้นหา (search key value) การค้นหากระทำต่อไปโดยการตรวจสอบอย่างสืบเนื่องในรายการ ; การตรวจสอบแต่ละครั้งจะตัดหน่วยข้อมูล (entries) ส่วนที่เหลือออกไปทีละครึ่งจากการพิจารณาต่อไป เทคนิคการค้นหาแบบทวิภาคถูกนำเสนอเวอร์ชันการเรียกซ้ำ (recursive) และเวอร์ชันการทำซ้ำ (iterative) ทั้งคู่

ส่วนที่เหลือของบทนี้เน้นที่เทคนิคการเรียงลำดับรายการของระเบียน การเรียงลำดับปกติต้องทำให้เสร็จสิ้นก่อน เพื่อเตรียมรายการสำหรับการค้นหาภายหลัง

ขั้นแรกวิธีเรียงลำดับภายในขั้นพื้นฐานหลายวิธีได้ถูกแนะนำ ทุกวิธีต้องใช้จำนวนของการเปรียบเทียบโดยประมาณเป็นสัดส่วนกันจำนวนคีย์ในรายการยกกำลังสอง ; นั่นคือต้องใช้การเปรียบเทียบ $O(n^2)$

การเรียงลำดับแบบเลือกโดยตรงทำซ้ำ ๆ กันโดยเลือกคีย์ที่มีค่าน้อยที่สุดที่เหลืออยู่ในรายการซึ่งยังไม่เรียงลำดับให้เป็นคีย์ตัวถัดไปในรายการเรียงลำดับ การเรียงลำดับเลือกแบบสับเปลี่ยนเรียงรายการให้ถูกตำแหน่ง การเรียงลำดับแบบใส่โดยตรง เอาคีย์ตัวถัดไปของรายการซึ่งยังไม่เรียง และใส่มันในตำแหน่งสัมพันธ์ที่ถูกต้องของมัน ในการโตขั้นของรายการที่เรียงลำดับ การเรียงลำดับแบบฟอง(หรือการเรียงลำดับแบบสับเปลี่ยน) เปรียบเทียบคู่ของค่าคีย์ที่ติดกันและสับเปลี่ยนกันถ้ามันไม่อยู่ในอันดับสัมพันธ์ที่ถูกต้อง ทำซ้ำ ๆ กัน

การเรียงลำดับแบบสับเปลี่ยน-บางส่วนหรือที่เรียกว่า ควิกซอร์ตของ Hoare ถูกนำเสนอให้เป็นวิธีของการได้การกระทำงานที่ดีในการเรียงลำดับ ระหว่าง pass การเปรียบ

เทียบแต่ละครั้งของควิกซอร์ต คีย์ถูกสับเปลี่ยนในวิธีซึ่งเมื่อเสร็จสิ้น pass รายการถูกแบ่งเป็นส่วนและสองรายการย่อยภายหลังจากนั้นจะถูกปฏิบัติอย่างเป็นอิสระกันของแต่ละชุด ควิกซอร์ตต้องใช้การเปรียบเทียบเฉลี่ย $O(n \log_2 N)$ แต่ในกรณีแย่ที่สุดต้องใช้การเปรียบเทียบ $O(n^2)$

ฮีปซอร์ตปรับปรุงการกระทำงานดีขึ้นเหนือควิกซอร์ต ; มันรับประกันการเปรียบเทียบ $O(n\log_2 n)$ แม้กระทั่งในกรณีแย่ที่สุด ฮีปซอร์ตมีสองขั้นตอนในขั้นตอนแรก subject keys ถูกทำโครงสร้างให้เป็นฮีป ซึ่งเป็นกรณีพิเศษของต้นไม้แบบทวิภาค ในขั้นตอนที่สอง ฮีปถูกประมวลผลและทำให้เป็นเชิงเส้นในวิธีซึ่งให้ผลลัพธ์เป็นรายการเรียงลำดับของคีย์

การเรียงลำดับแบบแข่งขัน (หรือเรียกว่าการเรียงลำดับแบบ tree-selection) และความหลากหลาย , การเรียงลำดับแบบ tree replacement-selection ถูกนำเสนอเป็นเทคนิคที่สำคัญสำหรับการเรียงลำดับกลุ่มข้อมูลขนาดใหญ่ คีย์ถูกส่งเข้าไปยังต้นไม้การแข่งขันและเปรียบเทียบในลักษณะทีละคู่เพื่อสร้างสายอักขระที่เรียงลำดับ ถ้าต้นไม้ไม่ใหญ่พอที่จะเก็บกลุ่มของคีย์ทั้งหมด ดังนั้นการเรียงลำดับจะสร้างสายอักขระที่เรียงลำดับแล้วหลายชุด จากนั้นสายอักขระต้องถูกนำมาผสานกันเพื่อให้รายการเรียงลำดับชุดสุดท้าย สำหรับ replacement-selection , การเรียงลำดับของคีย์ p ตัวจะสร้างสายอักขระเรียงลำดับแล้วที่คาดคะเนความยาวเท่ากับ 2 * p นี่คือคุณสมบัติที่ทำให้การเรียงลำดับแบบแข่งขันเป็นการเลือกที่แพร่หลายสำหรับขั้นตอนภายใน ของการเรียงลำดับแฟ้มภายนอก การเรียงลำดับแบบแข่งขันคลายกับการเรียงลำดับเชิงต้นไม้วิธีอื่น ๆ ต้องใช้การเปรียบเทียบ $O(n\log_2 n)$ โดยเฉลี่ย

การค้นหาและการเรียงลำดับเป็นกิจกรรมของการประมวลผลข้อมูลที่มีความสำคัญมากเทคนิคที่ใช้เป็นปัจจัยสำคัญในการกำหนดการกระทำงาน (performance) ของระบบสารสนเทศ

แบบฝึกหัด

1. จงแสดงให้เห็นโดยการยกตัวอย่างความแตกต่างระหว่างการเรียงลำดับที่เสถียร กับการเรียงลำดับซึ่งไม่เสถียร (Show by example the distinction between a stable sort and an unstable sort.)
2. จงพิจารณาความน่าจะเป็นข้างล่างนี้ของการเป็นอาร์กิวเมนต์การค้นหา

key(i)	prob(i)
8	.05
2	.26
10	.21
4	.15
12	.32

   (a) จงหาความน่าจะเป็นของการค้นหาที่ไม่ประสบผลสำเร็จ

   (b) ถ้าคีย์ถูกเรียงอันดับเช่นที่แสดงไว้ข้างต้นจงหาจำนวนคาดคะเนของการเปรียบเทียบสำหรับ การค้นหาแบบลำดับ

   (c) จงจัดระเบียบใหม่ให้กับคีย์เพื่อให้จำนวนคาดคะเนของการเปรียบเทียบสำหรับการค้นหาแบบลำดับมีค่าน้อยที่สุด

   (d) จงหาจำนวนคาดคะเนของการเปรียบเทียบสำหรับการค้นหาแบบลำดับสำหรับคำตอบในข้อ (c)
3. จงอธิบายเทคนิคสองวิธีสำหรับการจัดระเบียบใหม่ของรายการเชิงเส้นอย่างพลวัต เพื่อปรับปรุงเวลาค้นหาคาดคะเนที่ดีขึ้น
4. จงเปรียบเทียบการใช้วิธี "move - to - the - front " บนรายการเชิงเส้นเก็บในแถวลำดับและเก็บในรายการโยง
5. จงเปรียบเทียบการใช้วิธี "transposition" บนรายการเชิงเส้นเก็บในแถวลำดับและเก็บในรายการโยง
6. ทำไมจึงต้องมีการเรียงลำดับกระทำ "in place"
7. ให้ใช้ไพ่หรือบัตรอื่น ๆ เก็บค่าคีย์ จงแสดงให้เห็นความแตกต่างระหว่าง การเรียงลำดับแบบใส่, การเรียงลำดับแบบเลือกและการเรียงลำดับแบบสับเปลี่ยน
8. จงเขียนโปรแกรมเพื่อ implement การเรียงลำดับแบบฟอง

9. จงเขียนอัลกอริทึมเพื่อค้นหาแบบลำดับสำหรับระเบียนทั้งหมดด้วยค่าคีย์เฉพาะในรายการโยง
10. ในหนังสือได้อภิปรายเทคนิคการค้นหาแบบทวิภาคสำหรับประยุกต์ใช้เพื่อเรียงลำดับรายการเชิงเส้น จงพัฒนาและวิเคราะห์เทคนิคการค้นหาแบบไตรภาค (a ternary search technique)
11. ในหนังสือมีอัลกอริทึมสำหรับการค้นแบบทวิภาค สมมติว่ารายการเชิงเส้นเก็บในแถวลำดับซึ่งดรรชนีล่างมีพิสัยจาก 1 ถึง n จงดัดแปร (modify) อัลกอริทึมให้จัดการกับดรรชนีล่างมีพิสัยจาก a ถึง b เมื่อ a และ b ไม่จำเป็นต้องเป็นจำนวนเต็มบวก
12. ในหนังสือมีอัลกอริทึมสำหรับการค้นหาแบบทวิภาค สมมติว่ารายการเชิงเส้นเก็บในแถวลำดับ จงบอกข้อดีสัมพัทธ์และข้อไม่ดีสัมพัทธ์ของการประยุกต์เทคนิคการค้นหาแบบทวิภาคใช้กับรายการเชิงเส้นเก็บในรายการโยง
13. จงอธิบายการปรับปรุงโปรซีเดอร์ซึ่งการเรียงลำดับแตกต่างกันที่ต้องใช้การปฏิบัติการ $O(n\log_2 n)$ จากการเรียงลำดับที่ต้องใช้การปฏิบัติการ $O(n^2)$
14. กราฟ n^2 และ $n\log_2 n$ (หรือ n และ $\log_2 n$) กับ n และจงพิสูจน์ extent ของการปรับปรุงความสำเร็จในการเรียงลำดับแบบไม่ใช่เชิงเส้น
15. จงเขียนอัลกอริทึมที่กำหนดให้ในบทนี้ใหม่ สำหรับการเรียงลำดับลง (descending) ไม่ใช่การเรียงลำดับขึ้น (ascending sorts)
16. จงเขียนโปรแกรมเพื่อกระทำ(perform)การเรียงลำดับแบบ tree-selection
17. จงเขียนโปรแกรมเพื่อกระทำการเรียงลำดับแบบ tree replacement-selection
18. จงเขียนโปรแกรมเพื่อกระทำควิกซอร์ต วิ่งโปรแกรมกับเซตของ input data หลายชุด และให้เปรียบเทียบ performance คาดคะเนและกรณีแย่ที่สุด
19. จงเขียนโปรแกรมกระทำ ฮีปซอร์ต วิ่งโปรแกรมด้วยอินพุต data ชุดเดียวกับที่ใช้ในแบบฝึกหัดข้อ 18 และให้เปรียบเทียบ performance คาดคะเนและกรณีแย่ที่สุด
20. จงพัฒนาอัลกอริทึมเรียงลำดับที่รับประกัน performance $O(n\log_2 n)$ จงบอกคุณสมบัติอัลกอริทึมการเรียงลำดับซึ่งรับประกัน performance $O(n\log_2 n)$
21. มีอัลกอริทึมการเรียงลำดับใด ๆ หรือไม่ที่ให้ performance $O(n\log_2 n^2)$
22. จงพัฒนาแนวทาง (guidelines) สำหรับผลกระทบของค่าสัมพัทธ์ต่าง ๆ ของ n และ p บน tree replacement-selection performance ควรจะเลือก p อย่างไร? จะเกิดอะไรขึ้นถ้า p=n จะเกิดอะไรขึ้นถ้า p เกือบจะเท่ากับ n จะเกิดอะไรขึ้น ถ้า p=1

23. จงพิจารณารายการโยงสองทิศทาง (bidirectional linked list) ของสมาชิกข้อมูลซึ่งมีการ implement โดยใช้ตัวแปรพอยน์เตอร์ เมื่อใดเราจึงจะใช้เทคนิคการค้นหาแบบลำดับเพื่อหาสมาชิกเฉพาะหนึ่งตัว
24. จงวิเคราะห์อัลกอริทึมการเรียงลำดับแต่ละชนิดที่นำเสนอในบทนี้ สำหรับความเที่ยง (stability) ถ้าอัลกอริทึมการเรียงลำดับไม่เสถียรจะสามารถดัดแปรเพื่อทำให้มันเสถียร (stable) ได้อย่างไร
25. จงอธิบายว่านักศึกษาเลือกชนิดของการเรียงลำดับ เพื่อใช้บนเซตหนึ่งของระเบียนอย่างไรอะไรเป็นปัจจัยที่จำเป็นที่ต้องถูกพิจารณา
26. จงอธิบายว่านักศึกษาเลือกชนิดของการค้นหา เพื่อใช้บนเซตของระเบียนอย่างไร อะไรเป็นปัจจัยที่จำเป็นซึ่งต้องถูกพิจารณา
27. จงอ่านงานที่ดีเยี่ยมของ knuth เรื่องการเรียงลำดับและการค้นหา