บทที่ 4 การเรียงและการค้นหาข้อมูล
(Sorting and Searching)

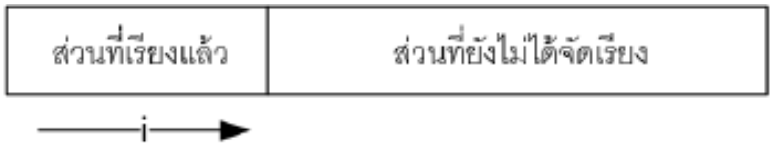
4.1 การเรียงข้อมูล (Sorting)

การเรียงข้อมูล เป็นขั้นตอนการเตรียมข้อมูลที่สำคัญขั้นตอนหนึ่ง เพื่อใช้สำหรับ ประมวลผลเพื่อหาคำตอบ หรือหาผลสรุปทางสารสนเทศ ข้อมูลที่ผ่านการจัดเรียงมาย่อมทำ ให้การดำเนินการในขั้นตอนต่อ ๆ ไปง่าย รวดเร็ว และมีประสิทธิภาพ อย่างไรก็ตามการ จัดเรียงข้อมูลมีหลากหลายวิธี แต่ละวิธีก็มีข้อดีและข้อจำกัดแตกต่างกันไป ดังนั้นการเลือกวิธีการจัดเรียงข้อมูลนอกจากจะเข้าใจวิธีการจัดเรียงแล้ว ยังต้องตระหนักถึงความเหมาะสมสำหรับงานแต่ละลักษณะด้วย

4.1.1 การเรียงแบบแทรก (Insertion sort)

การเรียงลำดับแบบนี้เป็นการเรียงลำดับด้วยการแทรก ซึ่งเป็นวิธีการที่คน ส่วนมากเคยใช้เป็นวิธีง่าย ๆ ไม่ซับซ้อน เช่น ในขณะที่เล่นไพ่มักจะมีการจัดเรียงไพ่ตามเลขโดยการหยิบมาแทรก

การเรียงข้อมูลแบบแทรก ข้อมูลจะจัดเก็บไว้ในอาเรย์ขนาด 1 มิติ ดังรูป 4.2 ขั้นตอนจะเริ่มพิจารณาข้อมูลตั้งแต่ 2 ตัวแรก โดยตัวไหนมีค่าน้อยกว่า (กรณีเรียงจากน้อยไป มาก) จะนำไปแทรกไว้ข้างหน้า ดังนั้นข้อมูล 2 ตัวแรกจึงถูกจัดเรียงกัน จากนั้นจะพิจารณา ข้อมูลตัวที่ 3 ว่ามีค่ามากหรือน้อยกว่าตัวสุดท้าย(ตัวที่ 2) หรือไม่ หากมีค่าน้อยกว่าจะทำการ นำเอาข้อมูลนั้นมาเก็บในตัวแปรชั่วคราวก่อน (Temp) แล้วทำการขยับตัวเลขก่อนหน้าไป ทางขวาจนกว่าข้อมูลในตัวแปรชั่วคราวจะไม่น้อยกว่า แล้วทำการแทรกด้วยข้อมูลที่อยู่ในตัว แปรชั่วคราวนั้น ทำอย่างนี้ไปเรื่อย ๆ จนกว่าจะหมดชุดข้อมูล



รูปที่ 4.1 แสดงส่วนการเรียงแบบแทรก

รูปที่ 4.2 แสดงวิธีการเรียงแบบแทรก

การวางโครงสร้างโปรแกรมการเรียงแบบแทรก

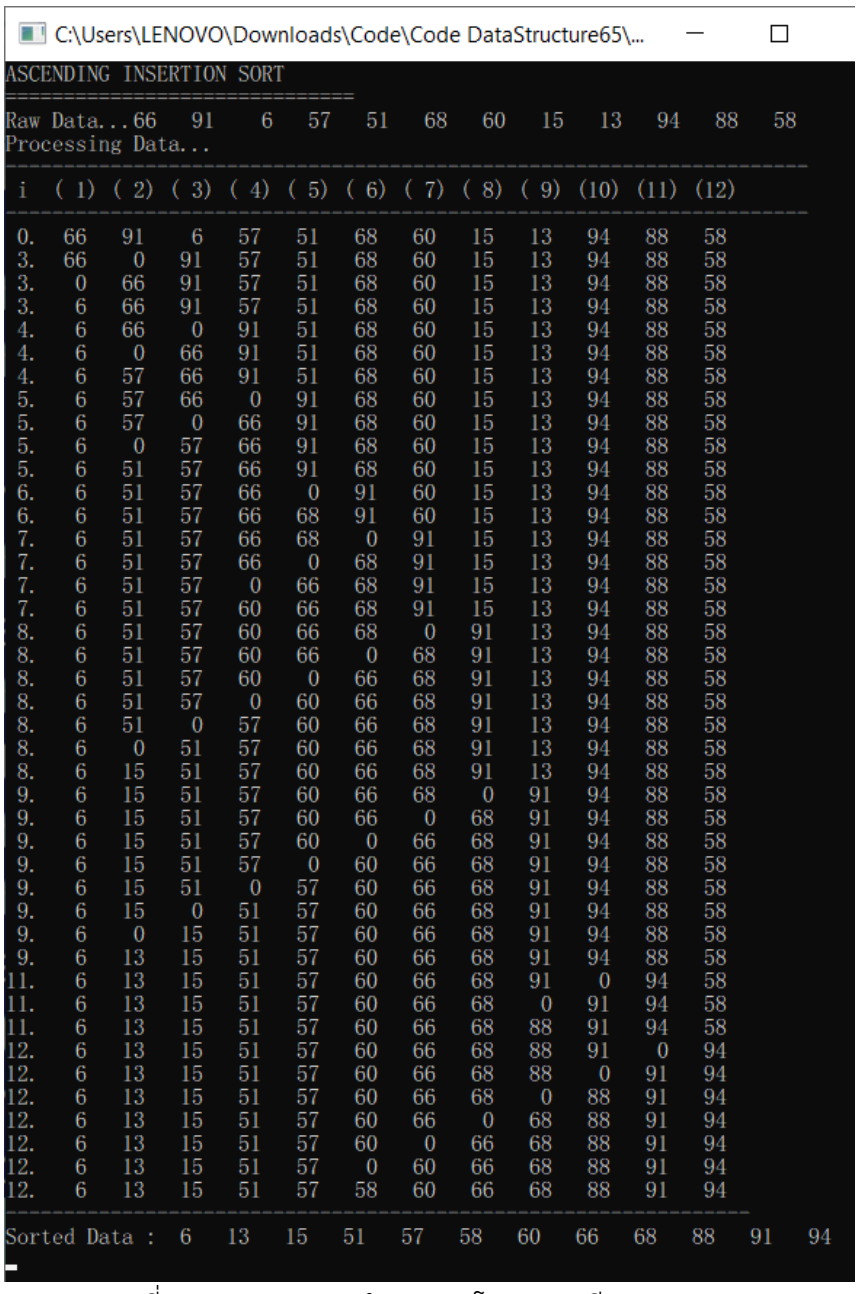
จะใช้คำสั่ง For ในการวนรอบเปรียบเทียบค่า หากทราบว่าตำแหน่งของข้อมูลไม่ ถูกต้องก็จะเก็บข้อมูลนั้นไว้ในตัวแปรชั่วคราวอันหนึ่ง (ในที่นี้จะใช้ temp) จากกนั้นจะทำ การขยับตัวเลขในอาเรย์ ณ จุดที่พบว่ามีค่าไม่ถูกต้อง โดยการขยับไปทางขวาทีละตัว ๆ จนกว่าจะได้ตำแหน่งที่ถูกต้องแล้วนำค่าในตัวแปรชั่วคราว (temp) ไปแทรกในอาเรย์

4.1.1.1 โปรแกรมการเรียงแบบแทรก

```
/* Program ASCENDING INSERTION SORT can..
 1. Random raw data into 1 dimension Array
 2. Sorting and display detail of each result
 3. Display final result
 */
#include <stdio.h> //use printf
#include <conio.h> //use getch
#include <stdlib.h> //use random
#include <time.h> //use time
#define MaxData 100 // Define Max Data
int Data[MaxData];
int N;
void PrepareRawData(int N)
 {
  int i;
  srand(time(NULL)); //for difference random number in rand()
```

```
  for (i=1;i<=N;i++)
    Data[i]=1+rand() % 99; //random difference number 1..99
 }
void DispData(int N)
 {
  int i;
  for(i=1;i<=N;i++)
    printf("%2d   ",Data[i]);
  printf("\n");
 }
void InsertionSort(int N)
 {
  int i,j,temp;
  printf("-------------------------------------------------------------
--------\n");
  printf(" i ");
  for(i=1;i<=N;i++)
    printf(" (%2d)",i);
  printf("\n");
  printf("-------------------------------------------------------------
--------\n");
  printf("%2d.  ",0);
  DispData(N); //Show every step sorting
  for (i=1;i<N;i++) //Count i forward
  {
    if(Data[i+1]<Data[i]) //If next data < previous data
    {
      temp=Data[i+1]; //Keep insert data into temp
      Data[i+1]=NULL;
      j=i; //let counter j loop backward
      while(temp<Data[j]) //loop if temp remain <       Data[j]
      {
        Data[j+1]=Data[j]; //Skip data forward to         next block of
array
        Data[j]=NULL;
        printf("%2d.  ",i+1);
        DispData(N); //Show every time sorting
        j=j-1; //count backward of j
      } //End while
      Data[j+1]=temp; //Put temp into Data[j+1] finally
      printf("%2d.  ",i+1);
      DispData(N); //Show every time sorting
    } //end if
  } //ENd for
 } //End Fn.
```

```
int main()
{
  printf("ASCENDING INSERTION SORT\n");
  printf("==============================\n");
  N=12;
  PrepareRawData(N);
  printf("Raw Data...");
  DispData(N);
  printf("Processing Data...\n");
  InsertionSort(N);
  printf("------------------------------------------------------------
---\n");
  printf("Sorted Data : ");
  DispData(N); //Sorted Data
  getch();
  return(0);
} //End Main
```



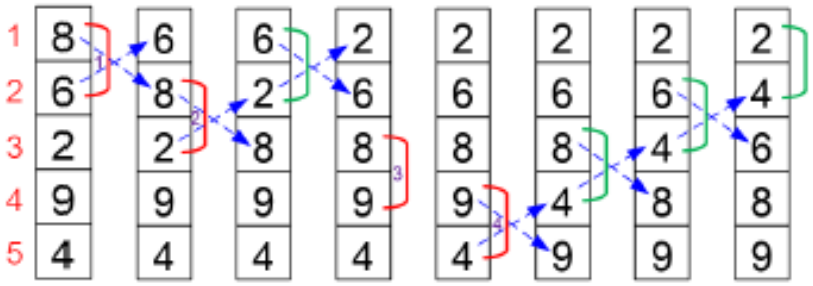
รูปที่ 4.3 แสดงผลการทำงานของโปรแกรมเรียงแบบแทรก

หมายเหตุ –เลข 0 ใส่ไว้เพื่อให้ผู้ศึกษาเข้าใจ และเห็นการเคลื่อนที่ในการขยับของ ตัวเลข เพื่อให้มีช่องว่าง ณ ตำแหน่งที่เหมาะสม แล้วจึงนำค่าที่มีค่าน้อยกว่านั้นใส่ลงในช่องที่ว่าง (ในการเขียนโปรแกรมใช้งานจริง ไม่จำเป็นต้องใส่เลข 0)

4.1.2 การเรียงแบบบับเบิล (Bubble Sort)

บับเบิลซอร์ทเป็นการเรียงข้อมูลด้วยการแลกเปลี่ยน หลักการของการเรียงลำดับวิธี นี้คือ จะเปรียบเทียบค่า 2 ค่าที่ “ติดกัน” ถ้าข้อมูลไม่ได้อยู่ในลำดับที่กำหนด เช่น จากน้อย ไปมาก จะให้แลกเปลี่ยนตำแหน่งของค่าทั้ง 2 ค่านั้น แล้วเอาค่าน้อย (หรือค่ามาก ถ้าเป็นการเรียงจากค่ามากไปหาค่าน้อย) เปรียบเทียบกับค่าถัดไปอีกเป็นเช่นนี้เรื่อยไปจนกว่าอยู่ใน ลำดับที่ถูกต้อง

สมมติว่ามีลิสต์ที่มีข้อมูล 8, 6, 2, 3, 4 จะใช้วิธีการเรียงข้อมูลแบบบับเบิล มาเรียง ข้อมูลจากน้อยไปหามาก



รูปที่ 4.4 แสดงวิธีการเรียงแบบบับเบิล

ชั้นแรกนำเอา 8 เปรียบเทียบกับ 6 (ตำแหน่งที่ 1 กับ 2) เนื่องจาก 8 มากกว่า 6 ซึ่ง ไม่เป็นไปตามลำดับจากน้อยไปมากจึงให้แลกที่ 8 กับ 6 จากนั้นนำ 8 ไปเปรียบเทียบกับ 2 เนื่องจาก 8 มากกว่า 2 จึงให้แลกที่กัน แล้วนำ 2 ไปเปรียบเทียบกับ 6 อีกเนื่องจาก 6 มากกว่า 2 จึงให้แลกที่กัน ต่อมานำ 8 มาเทียบกับ 9 แล้วให้ทำลักษณะเดิมคือถ้าค่าที่นำมา เปรียบเทียบไม่เรียงกันแบบน้อยไปมากก็ให้สลับค่ากัน แต่ถ้าเรียงจากน้อยไปมากแล้วไม่ต้องสลับค่า ในที่สุดค่าที่น้อยที่สุดจะไปอยู่ข้างบน ส่วนค่าที่มากที่สุดจะลงไปอยู่ข้างล่างสุด

สิ่งที่ต้องทำในการเรียงค่า คือ เปรียบเทียบและ แลกค่า จะพิจารณากรณีปลายสุด 2 กรณีคือ

1) ถ้าลิสต์นั้นเป็นลิสต์ที่เรียงเรียบร้อยแล้วจากมากไปน้อย และ บังเอิญต้องการ เรียงจากมากไปน้อยโดยใช้แบบบับเบิล จะสามารถทำได้โดยไม่ต้องมีการแลก ตำแหน่งเลยและต้องทำการเปรียบเทียบ เพียง **“n-1”** ครั้งเท่านั้น (ถ้ามีข้อมูล n ตัว)

2) ถ้าลิสต์นั้นเป็นลิสต์ที่เรียงเรียบร้อยแล้วจากน้อยไปมากและ ต้องการเรียงมันจากมากไปน้อย ต้องทำการแลกเปลี่ยนและเปรียบเทียบเป็นจำนวน 1+2+3+...+(n-1) ครั้งนั่นเอง

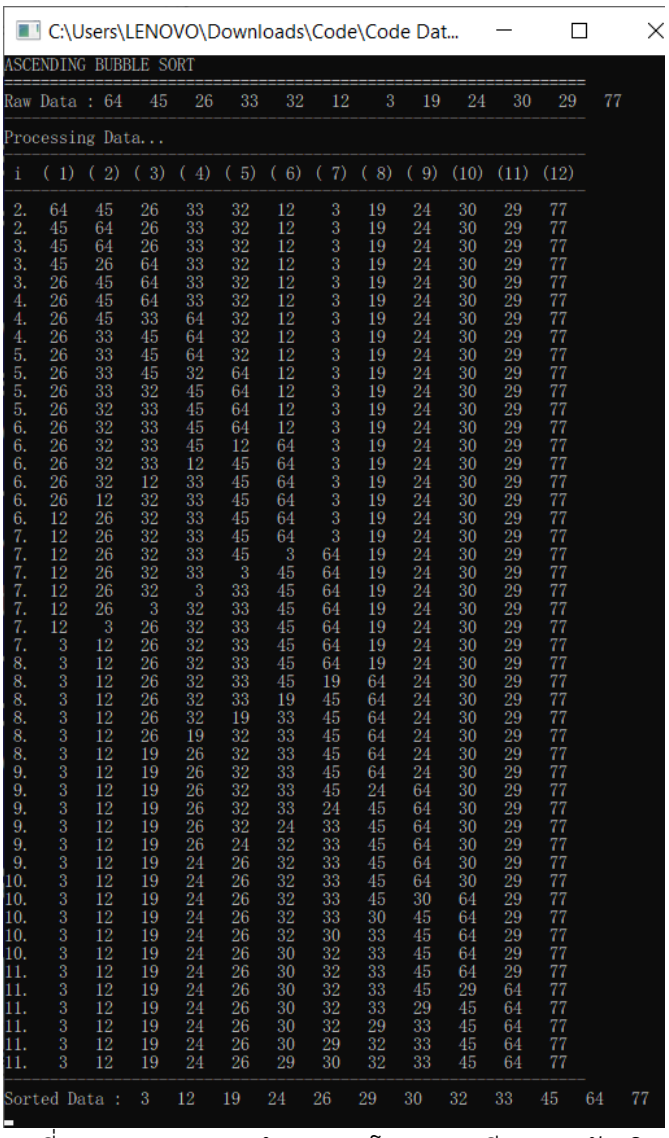
การวางโครงสร้างโปรแกรมการเรียงแบบบับเบิล

จะใช้คำสั่ง For โดยทำการเปรียบเทียบค่าในอาเรย์ตั้งแต่คู่แรก (ตำแหน่งที่ 1 กับ 2) ตัวไหนมีค่าน้อยกว่า (กรณีเรียงจากน้อยไปมาก) จะสลับค่ากัน แล้วจึงพิจารณาข้อมูลคู่ต่อไป หากไม่ถูกตำแหน่งจะสลับค่า โดยค่าที่ถูกสลับขึ้นจะถูกเปรียบเทียบกับค่าที่อยู่ข้างบนอีกไป เรื่อย ๆ โดยใช้คำสั่ง While และถูกสลับที่จนกว่าจะอยู่ถูกตำแหน่ง จึงเสมือนกับฟอง (Bubble) ที่ลอยขึ้นไปข้างบน

4.1.2.1 โปรแกรมการเรียงแบบบับเบิล

```
/* Program ASCENDING BUBBLE SORT can..
 1. Random raw data into 1 dimension Array
 2. Sorting and display detail of each step result
 3. Display final result
 */
#include <stdio.h> //use printf
#include <conio.h> //use getch
#include <stdlib.h> //use random
#include <time.h> //use time
#define MaxData 100 // Define Max Data
int Data[MaxData];
int N;
void PrepareRawData(int N)
{
  int i;
  srand(time(NULL)); //for difference random number in rand()
  for (i=1;i<=N;i++)
    Data[i]=1+rand() % 99; //random difference number 1..99
}
void DispData(int N)
{
  int i;
  for(i=1;i<=N;i++)
    printf("%2d   ",Data[i]);
  printf("\n");
}
void BubbleSort(int N)
{
  int i,j,temp;
  printf("-------------------------------------------------------------
---\n");
  printf(" i ");
  for(i=1;i<=N;i++)
    printf(" (%2d)",i);
  printf("\n");
```

```
  printf("-------------------------------------------------------------
---\n");
  for(i=1;i<=N-1;i++) //loop forward
  {
    if(Data[i]>Data[i+1]) //if not true position
    {
      printf("%2d.  ",i+1);
      DispData(N);
      j=i+1; //loop backward
      while(Data[j]<Data[j-1]) //while if remain bubble
      {
        temp=Data[j-1]; //swap data
        Data[j-1]=Data[j];
        Data[j]=temp;
        j--; //count down j
        printf("%2d.  ",i+1);
        DispData(N);
      } //end while
    } //end if
  } //end for
} //end Fn.

int main()
{
  printf("ASCENDING BUBBLE SORT\n");
  printf("=============================================================
===\n");
  N=12;
  PrepareRawData(N);
  printf("Raw Data : ");
  DispData(N);
  printf("-------------------------------------------------------------
---\n");
  printf("Processing Data...\n");
  BubbleSort(N);
  printf("-------------------------------------------------------------
---\n");
  printf("Sorted Data : ");
  DispData(N); //Sorted Data
  getch();
  return(0);
} //End Main
```



รูปที่ 4.5 แสดงผลการทำงานของโปรแกรมเรียงแบบบับเบิล

4.1.3 การเรียงแบบควิก (Quick Sort)

เป็นวิธีที่เหมาะกับลิสต์ขนาดใหญ่ และเป็นวิธีเรียงข้อมูลที่ให้ค่าเฉลี่ยของเวลาที่ใช้ น้อยที่สุดเท่าที่ค้นพบวิธีหนึ่ง อย่างไรก็ดีต้องใช้สแตกสำหรับเป็นตำแหน่งของลิสต์ย่อยที่ยังไม่ได้เรียง

สมมติ A(K_1, K_2, ..., K_n) เป็นลิสต์ของค่าหรือเรคอร์ดที่ยังไม่ได้เรียง เริ่มจากการหา ค่า K_1 จากนั้นจะแบ่งลิสต์ A นี้ออกเป็น 2 ลิสต์ย่อยคือ s_1 และ s_2 โดยที่

s_1ประกอบด้วยเรคอร์ดที่มีค่าคีย์น้อยกว่า K_1

s_2 ประกอบด้วยเรคอร์ดที่มีค่าคีย์มากกว่า K_1

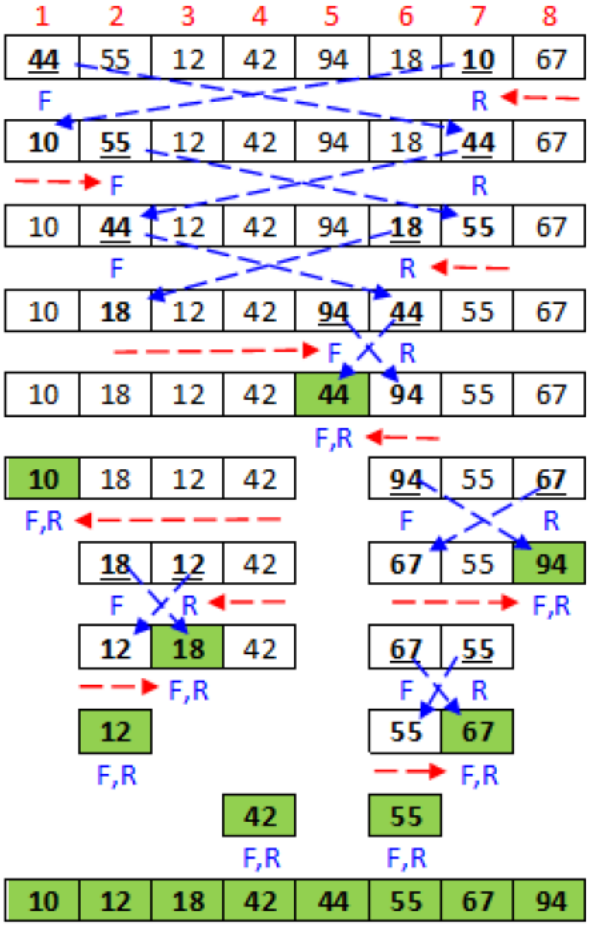
นั่นคือ สามารถเขียนลิสต์ A ได้ดังนี้

$$\{s_1\} < k_1 < \{ s_2 \}$$

โดยที่ลิสต์ { S_1 } และ { S_2 } เป็นค่าต่าง ๆ ที่ยังไม่ได้เรียงจากนั้นก็ทำวิธีดังกล่าว ซ้ำกับ { S_1 } และ { S_2 } ตามลำดับ (อย่างรีเคอร์ซีฟ) ในที่สุดจะได้ลิสต์ที่เรียงตามที่ต้อง

การเรียงข้อมูลจะเริ่มโดยใช้พอยน์เตอร์ 2 ตัวคือ F (Front) และ R (Rear) เริ่มแรก จะให้ F มีค่า 1 ซึ่งคือชี้ไปยังค่าคีย์ตัวแรก ส่วน R มีค่าเท่ากับ N นั่นคือชี้ไปยังค่าคีย์ตัว สุดท้ายในลิสต์ การเปรียบเทียบจะกระทำระหว่างค่าที่ถูกชี้โดย F และ R (อย่าลืมว่า จุดประสงค์ของ คือการแบ่งลิสต์นี้ออกเป็น 2 ลิสต์ย่อย โดยใช้ค่า K_1 เป็นตัวเปรียบเทียบ ฉะนั้นพอยน์เตอร์ที่ชี้ไปยัง K_1 ไม่ว่าจะเป็น F หรือ R จะไม่เป็นตัวเลื่อนไปยังตำแหน่งอื่น) ทุก ครั้งที่มีการเปรียบเทียบจะมีการเลื่อนพอยน์เตอร์ F ไปข้างหน้า นั่นคือจากซ้ายไปขวา หรือ R จะเลื่อนถอยหลัง นั่นคือจากขวาไปซ้าย การจะเลื่อนพอยน์เตอร์ตัวใดให้ใช้กฎต่อไปนี้ “ให้เลื่อนพอยน์เตอร์ตัวที่ไม่ใช่ชี้ไปยังค่า K_1 หรือที่ทำหน้าที่ K_1 ในการเรียงเที่ยวนั้น เมื่อ F พบ R ที่ค่า K_1 เป็นอันว่าเสร็จสิ้นการเรียงเที่ยวนั้น”

ถ้ามีชุดคีย์ที่เรียงมีดังนี้ 44, 55, 12, 42, 94, 18, 10, 67 ต้องการเรียงข้อมูลจาก น้อยไปหามากจะได้ดังรูป 4.6



รูปที่ 4.6 แสดงการเรียงข้อมูลแบบควิก

เริ่มแรกให้ F ชี้ที่ตำแหน่ง 1 และ R ชี้ตำแหน่ง 8 ค่าที่ F ชี้เป็น 44 ค่าที่ R ชี้เป็น 67 ซึ่งถูกต้อง (กรณีเรียงจากน้อยไปมาก) ดังนั้นจึงเลื่อน R ไปทางซ้ายไปชี้ตำแหน่ง 7 ซึ่งมีค่า 10 ทำให้ค่าที่ R ชี้มีค่าน้อยกว่าค่าที่ F ชี้ จึงต้องสลับค่า ณ ตำแหน่งที่ F และ R ชี้ นั้นคือ สลับค่าระหว่าง 44 และ 10

เมื่อมีการสลับค่าให้กลับมาเลื่อน F ไปทางขวาไปชี้ตำแหน่ง 2 ซึ่งมีค่า 55 ทำให้ค่าที่ F ชี้ มีค่ามากกว่าค่าที่ R ชี้ต้องสลับค่าระหว่าง 55 กับ 44

เมื่อมีการสลับค่าให้กลับมาเลื่อน R ไปทางซ้ายมาชี้ตำแหน่งที่ 6 ซึ่งมีค่า 18 ต้องทำ การสลับค่าที่ F และ R ชี้อีก โดยสลับค่าระหว่าง 44 กับ 18

เมื่อมีการสลับค่าให้กลับมาเลื่อน F ไปทางขวาไปเรื่อย ๆ จนกว่าค่าที่ชี้โดย F จะมี ค่ามากกว่าค่าที่ชี้โดย R จึงทำให้ F ชี้ไปที่ตำแหน่ง 5 ซึ่งมีค่า 94 ซึ่งมากกว่า R ต้องสลับ ระหว่าง 94 กับ 44

เมื่อมีการสลับค่าให้กลับมาเลื่อน R โดย R มาชี้ตำแหน่งเดียงกับ F ทำให้ได้ค่า K1 ของครั้งแรก ซึ่งมีค่าเป็น 44 ลิสต์จะถูกแบ่งเป็น 2 ส่วนตามหลักการดังกล่าวข้างต้น

จากนั้นนำลิสต์แต่ละส่วนไปจัดเรียงอีกโดยใช้ F และ R ชี้ จนกว่าสมาชิกทุกตัวของ ข้อมูลจะเป็นค่า K1 การจัดเรียงก็จะแล้วเสร็จ ซึ่งจะเห็นว่าสามารถใช้วิธีการเขียนโปรแกรม แบบเรียกตนเอง (Recursive) มาช่วยให้การทำงานกระชับขึ้นได้ โดยค่า Base Value สำหรับหยุดวนเรียกตนเองคือ F=R

การวางโครงสร้างโปรแกรมการเรียงแบบควิก

ข้อมูลจะจัดเก็บไว้ในอาเรย์ 1 มิติ ใช้ F และ R ชี้ตำแหน่งหัวท้าย และก่อนที่ค่า F และ R จะถูกเปลี่ยนแปลง จะเก็บค่าเดิมไว้ในตัวแปร f1 และ r1 เพื่อเอาไปใช้ในการคำนวณ ค่าตำแหน่งเพื่อวนเรียกตนเอง (Recursive)

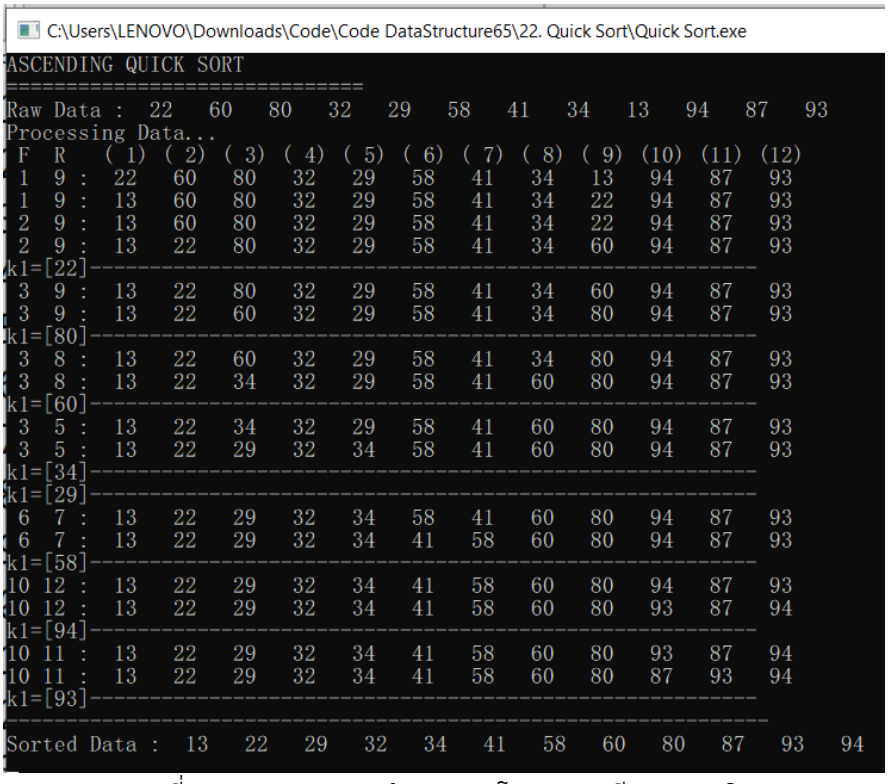
การเรียงในลิสต์ย่อย จะนำเอาลิสต์ส่วนซ้ายมือไปจัดเรียงให้แล้วเสร็จก่อนแล้วจึงค่อยเอาลิสต์ทางขวามือไปจัดเรียงโดยจะกระทำแบบวนเรียกตนเอง ซึ่งค่า Base Criteria ในการหยุดเรียกตนเองแต่ละครั้ง คือ F=R

ส่วนการกลับไป-มาของการเลื่อนค่า F และ R จะใช้ตัวแปร **direction** ซึ่งเป็นตัว แปรแบบตรรกะ (Boolean) โดยจะเปลี่ยนแปลงไปทางตรงข้ามทุกครั้งที่มีการสลับค่า

4.1.3.1 โปรแกรมการเรียงแบบควิก

```
/* Program ASCENDING QUICK SORT can..
 1. Random raw data into 1 dimension Array
 2. Sorting and display detail of each result
 3. Display final result
 */
#include <stdio.h> //use printf
#include <conio.h> //use getch
#include <stdlib.h> //use random
#include <time.h> //use time
#define MaxData 100 // Define Max Data
int Data[MaxData];
int i,N;
void PrepareRawData(int N)
{
  int i;
  srand(time(NULL)); //for difference random number in rand()
  for (i=1;i<=N;i++)
    Data[i]=1+rand() % 99; //random difference number 1..99
}

void DispData(int N)
{
  int i;
  for(i=1;i<=N;i++)
    printf(" %2d  ",Data[i]);
  printf("\n");
}
void swap(int a,int b)
{
  int temp;
  temp=Data[a];
  Data[a]=Data[b];
  Data[b]=temp;
}
void QuickSort(int f, int r) //Recursive Fn.
{
  int f1,r1;
  bool direction;
  f1=f; r1=r; //keep old Front & Rear values
  direction=true;
  while(f!=r)
  {
    if(Data[f]>Data[r]) //Ascending case
    {
```

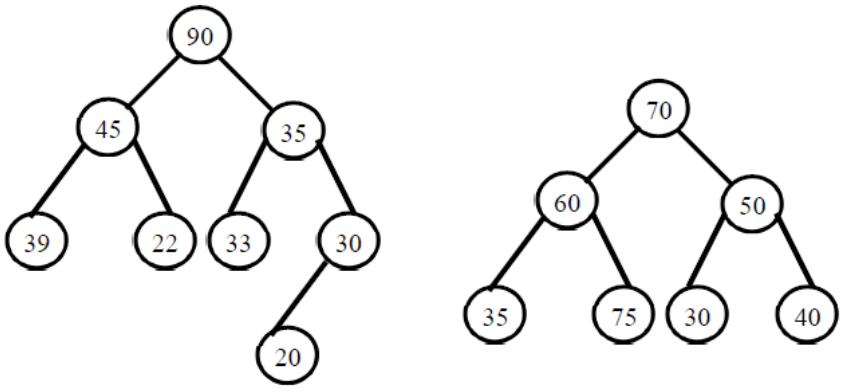
```
      printf("%2d %2d : ",f,r);
      DispData(N);
      swap(f,r);
      printf("%2d %2d : ",f,r);
      DispData(N);
      direction= !direction; //change moving pointer direction
    }
    if (direction) //move r to left if TRUE
      r--;
    else
      f++; //move f right if FALSE
  }
  printf("k1=[%2d]-----------------------------------------------------
---\n",Data[f]); //
  //process in left hand
  if((f>f1) && (f-1 != f1))
    QuickSort(f1,f-1); //recursive new position F&R
  //process in right hand
  if((r<r1) && (r+1 != r1))
    QuickSort(r+1,r1); //recursive set new position F&R
}

int main()
{
  printf("ASCENDING QUICK SORT\n");
  printf("==============================\n");
  N=12;
  PrepareRawData(N);
  printf("Raw Data : ");
  DispData(N);
  printf("Processing Data...\n");
  printf(" F  R  ");
  for(i=1;i<=N;i++)
    printf(" (%2d)",i);
  printf("\n");
  QuickSort(1,N);
  printf("-------------------------------------------------------------
---\n");
  printf("Sorted Data : ");
  DispData(N); //Sorted Data
  getch();
  return(0);
} //End Main
```



รูปที่ 4.7 แสดงผลการทำงานของโปรแกรมเรียงแบบควิก

4.1.4 การเรียงแบบฮีพ (Heap Sort)

การเรียงแบบฮีพ จะอาศัยโครงสร้างต้นไปไม้ฮีพ (Heap Tree) ซึ่งเป็นต้นไม้ไบนารีที่ มีคุณสมบัติว่าโหนดใด ๆ ในต้นไม้นั้นจะมีค่าคีย์ใหญ่กว่าค่าคีย์ที่อยู่ในต้นไม้ย่อยทางซ้าย (Left Sub Tree) และต้นไม้ย่อยทางขวา (Right Sub Tree) ของมัน (ถ้าโหนดนั้นมีลูก) ตัวอย่างดังรูป 4.8



(ก) โครงสร้าง heap (ข) ไม่ใช่โครงสร้าง heap

รูปที่ 4.8 การเปรียบเทียบระหว่างโครงสร้างต้นไม้ฮีพกับโครงสร้างต้นไม้อื่น

จากนิยามของโครงสร้างต้นไม้ฮีพว่ารูทโหนดของฮีพ จะเป็นโหนดที่มีค่าคีย์ใหญ่กว่า ดังนั้น จากชุดอินพุตที่กำหนดให้ ต้องสร้างฮีพขึ้นก่อน แล้วทำการเอาต์พุตรูทโหนดซึ่งจะได้ค่า แรก (ค่าที่

ใหญ่สุด) ของชุดที่เรียงแล้ว ในกรณีนี้จะเรียงจากน้อยไปมาก หลังจากที่เอาต์พุตค่า รูทโหนดไปแล้ว ต้นไม้ที่เหลืออยู่จะไม่เป็นฮีพ ต้องมีวิธีการตกแต่งหรือปรับแต่งให้ต้นไม้ที่ เหลืออยู่นั้นเป็นฮีพจะได้เอาต์พุตค่าถัดไปได้ ดังนั้นกระบวนการใหญ่ของการเรียงแบบชีพ ประกอบด้วย 3 ขั้นตอนดังนี้

ขั้นที่ 1 สร้างโครงสร้างต้นไม้ฮีพ

ขั้นที่ 2 เอาต์พุตคีย์ที่รูทโหนด

ขั้นที่ 3 ปรับแต่งต้นไม้ที่เหลือให้เป็นต้นไม้ฮีพ

การสร้างโครงสร้างต้นไม้ฮีพจากชุดอินพุต

สมมติมีข้อมูล 22, 35, 42, 38, 32, 26, 27, 90 การอินพุตค่าใหม่เข้าไปในชีพ ให้ถูกตำแหน่งค่าตามการแทนต้นแบบเรียงโหนด (เก็บในอาเรย์ 1 มิติ) หลักการมีดังนี้ (ให้ 1 เป็น พอยน์เตอร์ชี้ไปยังโหนด K_{new})

ขั้นที่ 1 : ให้เปรียบเทียบโหนดที่เข้าใหม่กับโหนดที่เป็นพ่อ

IF $K_{new} > K_{FATHER}$ THEN แลกที่กัน เลื่อน 1 ไปยังตำแหน่ง FATHER (นั่นคือ i ติดตาม K_{new}, ขึ้นไป)

ขั้นที่ 2 : ทำขั้นที่ (1) เรื่อย ๆ จนทำไม่ได้

หลังจากที่ข้อมูลในรูปโครงสร้างฮีพแล้ว จะเอาต์พุตค่ารูทโหนดซึ่งอยู่ในตำแหน่งที่ 1 ในอาเรย์ การเอาต์พุต จะให้แลกระหว่างค่า A(1) และ A(8) ค่าที่เอาต์พุตไปแล้วจะถือว่าเป็น เอาต์พุตซึ่งจะไม่นำมาเรียงต่อ นั่นหมายถึงสมาชิกของต้นไม้ฮีพจะลดลง 1 โหนดทุกทั้งที่มี การทำเอาต์พุตจากการเอาต์พุตแต่ละครั้งจะทำให้ต้นไม้ที่เหลือไม่เป็นฮีพ จึงต้องมีการปรับให้เป็นต้นไม้ฮีพ ดังนี้

ขั้นที่ 1 : ให้ตั้งค่าพอยน์เตอร์ 1 ชี้ไปยังรูทโหนด

ขั้นที่ 2 : ให้เลือกแลกค่าที่พอยน์เตอร์ 1 ชี้ กับค่าที่ใหญ่ที่สุดระหว่างลูกทางซ้าย(Left Son) และลูกทางขวา (Right Son) ของโหนด 1 แล้วเลื่อน 1 ไปตามลูก (son) ที่ทำการ แลกค่า ซึ่ง i จะมาอยู่ในตำแหน่งของโหนดล่าง

ขั้นที่ 3 : ทำขั้นที่ (2) จนกว่าจะทำไม่ได้

จากนั้นทำการเอาต์พุตที่รูทโหนด และปรับต้นไม้ที่เหลือให้เป็นฮีพอีก วนอย่างนี้ไปเรื่อย ๆ จนกว่าโหนดทุกตัวจะเป็นเอาต์พุต จะได้ข้อมูลที่เรียงแล้วเสร็จ

Input Seq. 22 35 42 38 32 26 27 90

Swap	Input	1	2	3	4	5	6	7	8
	22	22							
	35	22	35						
1,2	42	**35**	**22**	42					
1,3	38	**42**	22	**35**	38				
2,4	32	42	**38**	35	**22**	32			
	26	42	38	35	22	32	26		
	27	42	38	35	22	32	26	27	
	90	42	38	35	22	32	26	27	90
4,8		42	38	35	**90**	32	26	27	**22**
4,2		42	**90**	35	**38**	32	26	27	22
1,2		**90**	**42**	35	38	32	26	27	22
1,8	Out	**22**	42	35	38	32	26	27	**90**
1,2		**42**	**22**	35	38	32	26	27	**90**
2,4		42	**38**	35	**22**	32	26	27	**90**
1,7	Out	27	38	35	22	32	26	**42**	**90**
1,2		**38**	**27**	35	22	32	26	**42**	**90**
2,5		38	**32**	35	22	**27**	26	**42**	**90**
1,6	Out	**26**	32	35	22	27	**38**	**42**	**90**
1,3		**35**	32	**26**	22	27	**38**	**42**	**90**
1,5	Out	**27**	32	26	22	**35**	**38**	**42**	**90**
1,2		**32**	**27**	26	22	**35**	**38**	**42**	**90**
1,4	Out	**22**	27	26	**32**	**35**	**38**	**42**	**90**
1,2		**27**	**22**	26	**32**	**35**	**38**	**42**	**90**
1,3	Out	26	22	**27**	**32**	**35**	**38**	**42**	**90**
1,2	Out	22	**26**	**27**	**32**	**35**	**38**	**42**	**90**
1,1	Out	**22**	**26**	**27**	**32**	**35**	**38**	**42**	**90**

รูปที่ 4.9 แสดงการเรียงข้อมูลแบบชีพ

จากการที่ต้องใช้วิธีการเก็บโครงสร้างต้นไม้แบบเรียงโหนดข้อมูลทั้งหมดจะจัดเก็บไว้ในอาเรย์ขนาด 1 มิติ ดังนั้นจึงใช้การคำนวณหาความสัมพันธ์พ่อ-ลูกโดยสมการ

Father(i) = int(i/2)

LSon(i) = 2i

RSon(i) = 2i+1

การวางโครงสร้างโปรแกรมการเรียงแบบฮีพ

ด้วยเหตุที่การเรียงแบบฮีพมี 2 ส่วนใหญ่ ๆ คือ

1. สร้างโครงสร้างต้นไม้ฮีพและเอาต์พุตที่รูทโหนด
2. ปรับโครงสร้างต้นไม้ที่เหลือให้เป็นต้นไม้ฮีพ

ดังนั้นจึงแบ่งเป็น 2 โมดูล โดยลักษณะของแต่ละโมดูลเป็นดังนี้

1. โมดูลสร้างต้นไม้ฮีพ (CreateHeapTree) : เริ่มต้นจากการอินพุตข้อมูลทีละตัวตามหลักการของการแทนต้นไม้แบบเรียงโหนด โดยให้ข้อมูลตัวแรกเป็นโหนดที่ 1 ในอาเรย์ขนาด 1 มิติ ตั้งแต่ข้อมูลเป็นตัวที่ 2 เป็นต้นไปจะนำเอาข้อมูลนั้นไปไว้ ณ ตำแหน่ง 2 ของอาเรย์ ตามหลักการแทนแบบเรียงโหนด และทำการเปรียบเทียบค่าของข้อมูลตัวใหม่กับโหนดพ่อ โดยใช้สมการ **Father(i)=int(i/2)** หากโหนดลูกที่เข้ามาใหม่มีค่ามากกว่าโหนดพ่อจะทำการสลับค่าระหว่างโหนดพ่อกับโหนดลูกโดยสลับค่าภายในอาเรย์ และนำโหนดลูกที่ขึ้นไปนั้นเทียบกับโหนดพ่ออีก หากโหนดลูกมีค่ามากกว่าโหนดพ่อก็ทำการสลับอีก ทำไปเรื่อย ๆ จนกว่าจะถึงตำแหน่งที่ถูกต้อง จากนั้นก็อินพุตข้อมูลตัวที่ 3,4,5,... เรื่อย ๆ จนกว่าจะหมดชุดข้อมูล ก็จะได้ต้นไม้ฮีพ และทำการเอาต์พุตที่รูทโหนด (สลับค่าระหว่างรูทโหนดกับโหนดสุดท้าย) ซึ่งจากการทำเอาต์พุตนี้จะทำให้เกิด 2 เหตุกรณ์คือ

1.1 ได้เอาต์พุต 1 ตัว และทำให้จำนวนสมาชิกของต้นไม้หายไป 1 ตัว ดังนั้นทุกครั้งที่ทำการเอาต์พุต ค่า **LastNode (i)** จะลดลง 1 เสมอจนกว่า **LastNode** จะมีค่าเป็น 1 การเรียงจึงจะแล้วเสร็จ

1.2 ต้นไม้ที่เหลือจะไม่เป็นต้นไม้ฮีพ จึงต้องมีการปรับให้เป็นต้นไม้ฮีพในโมดูลถัดไป

2. โมดูลปรับต้นไม้ให้เป็นต้นไม้ฮีพ (AdjustTree) : จากการที่จำนวนสมาชิกของต้นไม้ที่เหลือหายไป 1 ตัว จึงต้องมีการส่งหมายเลขโหนดสุดท้ายของต้นไม้ที่เหลือ ซึ่งในที่นี้จะให้ตัวแปร **LastNode** รับการส่งผ่านค่าให้กับฟังก์ชัน เพื่อใช้ในการคำนวณและปรับต้นไม่ที่เหลือให้เป็นต้นไม้ฮีพโดยไม่ให้มีผลกระทบกับโหนดที่เป็นเอาต์พุตไปแล้ว ในที่นี้จะใช้ตัวแปร **result** ซึ่งเป็นตัวแปรชนิดตรรกในการควบคุมคำสั่ง **while** เพื่อเก็บผลของการปรับต้นไม้ให้เป็นต้นไม้ฮีพ ซึ่งเริ่มต้นจะให้ตัวแปรนี้เป็นเท็จ (false) และหากการปรับเสร็จสิ้นตัวแปรนี้จะมีค่าเป็นจริง (true)

การปรับจะนำเอาข้อมูลของโหนดหมายเลข 1 (ซึ่งได้จากการสลับค่ากับเอาต์พุต) โดยให้ **i=1** และนำมาเปรียบเทียบกับลูกทั้งสอง (หากมี) ซึ่งจะใช้คำสั่ง **Lson=2i** และ **Rson=2i+1** เพื่อหาตำแหน่งของลูกซ้ายและลูกขวา หากลูกไหนมีค่ามากกว่าจะทำการสลับค่ากับลูกนั้น และให้ค่า 1 ตามตำแหน่งที่ทำการสลับลงไปและทำการเปรียบเทียบกับลูกซ้าย-ขวาอีกเรื่อย ๆ จนกว่าจะอยู่ตำแหน่งที่ถูกต้อง

การปรับในขั้นตอนนี้ต้องแยกประเด็นการพิจารณาให้ดีว่าโหนดนั้นมีลูกกี่ตัว ซึ่งในที่นี้จะใช้ตัวแปร **son** เก็บจำนวนลูกโดยจะมีค่า **0** หรือ **1** หรือ **2** การเปรียบเทียบกรณีลูก 1 หรือ 2 นั้นจะมีวิธีการเปรียบเทียบต่างกัน อีกทั้งยังต้องตรวจสอบว่าโหนดนั้นเป็นโหนด

สุดท้ายหรือไม่ด้วย หากเป็นเป็นโหนดสุดท้ายก็จะยุติการทำงานโดยให้ค่า **result=true** หรือหาก **son=0** ก็จะยุติการปรับและให้ค่า **result=true** เช่นกัน ซึ่งจะเป็นการหยุดการ ทำงานของคำสั่ง **while** การปรับก็จะแล้วเสร็จ

จากนั้นจะทำการเอาต์พุตที่รูทโหนด จำนวนโหนดสุดท้าย (LastNode) ลดลง 1 และปรับอีกไปเรื่อย ๆ จนกว่าโหนดทุกตัวจะเป็นเอาต์พุตการเรียงก็จะแล้วเสร็จ

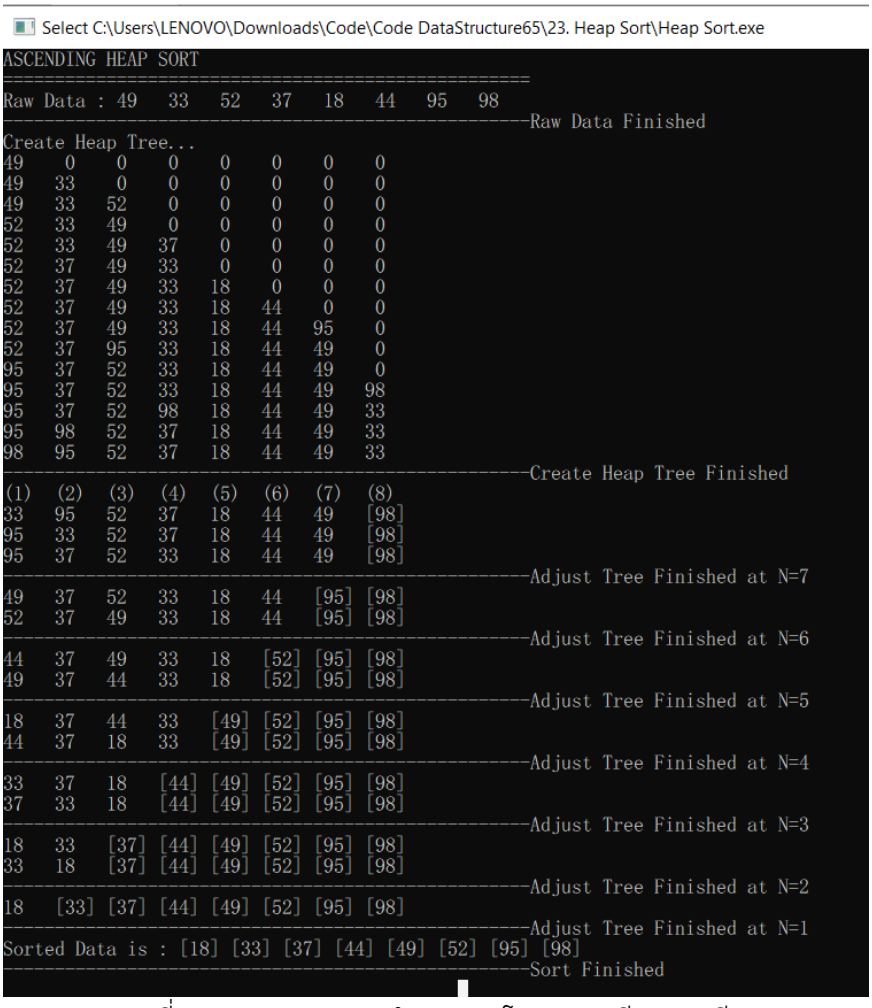
4.1.4.1 โปรแกรมการเรียงแบบฮีพ

```
/* Program ASCENDING HEAP SORT can..
 1. Random raw data into one dimension Array
 2. Sorting and display detail of each step result
 3. Display final result
 */
#include <stdio.h> //use printf
#include <conio.h> //use getch
#include <stdlib.h> //use random
#include <time.h> //use time
#define MaxData 100 // Define Max Data
int Data1[MaxData], Data2[MaxData];
int N;
void PrepareRawData(int N)
{
  int i;
  srand(time(NULL)); //for difference random number in rand()
  for (i=1;i<=N;i++)
    Data1[i]=1+rand() % 99; //random difference number 1..99
}
void DispData(int Data[],int out) //Out is point of Outputted Number
backward
{
  int i;
  for(i=1;i<=N;i++)
  {
    if(i<out)
      printf("%2d   ",Data[i]); //Show 2 width of number
    else
      printf("[%2d] ",Data[i]); //Show [ ] if it's Output
  }
  printf("\n");
}
void swap(int a,int b)
{
  int temp;
  temp=Data2[a];
```

```
  Data2[a]=Data2[b];
  Data2[b]=temp;
}
int Maximum(int a, int b) //Fine Maximum from 2 Data
{
  if(a>b)
    return(a);
  else
    return(b);
}
void AdjustTree(int LastNode)
{
  int i,Max,lson,rson,son;
  bool result;
  i=1;
  result=false; // False is NOT Finish Adjustment yet
  while(!result)
  {
    lson=(2*i); //Calculate LSon
    rson=(2*i)+1; //Calculate RSon
    son=0; //Set default Son
    if(lson==LastNode)
    {
      son=1;
      if(Data2[i]<Data2[lson]) //Check Father Data <    LSon data ?
      {
        swap(i,lson);
        DispData(Data2,LastNode+1); //Show each step result
      }
      result=true; //Finish Adjustment
    }
    if(rson<=LastNode)
    {
      son=2;
      Max=Maximum(Data2[lson],Data2[rson]); //Find Maximum Data
      if(Data2[i]<Max) //Check Father Data < Max ?
      {
        if(Max==Data2[lson]) //Max == Data Lson?
        {
          swap(i,lson);
          DispData(Data2,LastNode+1); //Show each step result
          if(rson==LastNode) //Check for Last Node
            result=true; //Finish Adjustment
          else
            i=lson; //Let i follow to LSon
        }
```

```
        else //if Data RSon is Maximum
        {
          swap(i,rson);
          DispData(Data2,LastNode+1); //Show each step
          if(rson==LastNode) //Check for Last Node
            result=true; //Finish Adjustment
          else
            i=rson; //Let i follow to RSon
        }
      }
      else
        result=true; //Finish Adjustment
    }
    if(son==0)
      result=true; //Finish Adjustment
  } //End While
  printf("-------------------------------------------------Adjust
Tree Finished at N=%d \n",LastNode);
} //End Fn.
void CreateHeapTree() // Create form Data1 into Data2
{
  int i,j,k,father;
  bool result;
  //Craete Heap Tree
  Data2[1]=Data1[1]; //First node of Heap Tree
  DispData(Data2,N+1); //Show each step result
  for(i=2;i<=N;i++)
  {
    Data2[i]=Data1[i];
    DispData(Data2,N+1); //Show each step result
    result=true;
    j=i; //set backward counter start here
    while(result)
    {
      father=j/2; //Calculate Father
      if((Data2[j]>Data2[father]) && (j>1)) //Heap tree adjusting
      {
        swap(j,father);
        DispData(Data2,N+1); //Show each step result
        j=father; //Let j follow to new Father
        result=true;
      }
      else
        result=false;
    } //End While
  } //End for
```

```
  printf("-------------------------------------------------Create
Heap Tree Finished \n");
  for(k=1;k<=N;k++) //Display Array subscript
    printf("(%d)  ",k);
  printf("\n");
  for(i=N;i>1;i--)
  {
    swap(1,i); //Output Root Node
    DispData(Data2,i); //Show each step result
    AdjustTree(i-1); //Call Adjust Heap Tree
  } //End for
} //End Fn.

int main()
{
  printf("ASCENDING HEAP SORT\n");
  printf("================================================\n");
  N=8;
  PrepareRawData(N);
  printf("Raw Data : ");
  DispData(Data1,N+1);
  printf("-------------------------------------------------Raw Data
Finished \n");
  printf("Create Heap Tree...\n");
  CreateHeapTree();
  printf("Sorted Data is : ");
  DispData(Data2,1); //Sorted Data
  printf("-------------------------------------------------Sort
Finished \n");
  getch();
  return(0);
} //End Main
```



รูปที่ 4.10 แสดงผลการทำงานของโปรแกรมเรียงแบบฮีพ