รายงาน

เรื่อง โปรแกรม HEAP SORT

จัดทำโดย

นายพงษ์พันธุ์ เลาวพงศ์

รหัสนักศึกษา 66543206019-2

เสนอ

อาจารย์ปิยพล ยืนยงสถาวร

รายงานนี้เป็นส่วนหนึ่งของวิชา

ENGCE124

โครงสร้างข้อมูลและขั้นตอนวิธี

(Data Structures and Algorithms)

หลักสูตร วศ.บ.วิศวกรรมคอมพิวเตอร์

สาขาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์

มหาวิทยาลัยเทคโนโลยีราชมงคลล้านนา ภาคพายัพ เชียงใหม่

ภาคเรียนที่ 1 ปีการศึกษา 2567

รายงาน

เรื่อง โปรแกรม HEAP SORT

จัดทำโดย

นายพงษ์พันธุ์ เลาวพงศ์

รหัสนักศึกษา 66543206019-2

เสนอ

อาจารย์ปิยพล ยืนยงสถาวร

รายงานนี้เป็นส่วนหนึ่งของวิชา

ENGCE124

โครงสร้างข้อมูลและขั้นตอนวิธี

(Data Structures and Algorithms)

หลักสูตร วศ.บ.วิศวกรรมคอมพิวเตอร์

สาขาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์

มหาวิทยาลัยเทคโนโลยีราชมงคลล้านนา ภาคพายัพ เชียงใหม่

ภาคเรียนที่ 1 ปีการศึกษา 2567

คำนำ

รายงานฉบับนี้เป็นส่วนหนึ่งของรายวิชา ENGCE124 โครงสร้างข้อมูลและขั้นตอนวิธี (Data Structures and Algorithms) หลักสูตร วศ.บ.วิศวกรรมคอมพิวเตอร์ สาขาวิศวกรรมไฟฟ้า คณะวิศวกรรมศาสตร์มหาวิทยาลัยเทคโนโลยีราชมงคลล้านนา ภาคพายัพ เชียงใหม่ ในระดับปริญญาตรีปีที่ 2 โดยมีจุดประสงค์ในการอธิบายโค้ดของโปรแกรม HEAP SORT รวมถึงอธิบายหลักการทำงานของโปรแกรม HEAP SORT และอธิบายผลลัพธ์การใช้งานโปรแกรม HEAP SORT

ผู้จัดทำรายงานหวังว่า รายงานฉบับนี้จะเป็นประโยชน์กับผู้ที่สนใจ หรือนักศึกษาทุกท่าน ที่กำลังหาศึกษาในหัวข้อของโปรแกรม HEAP SORT หากมีข้อแนะนำหรือข้อผิดพลาดประการใด ผู้จัดทำขอน้อมรับไว้และขออภัยมา ณ ที่นี้

ผู้จัดทำ

นายพงษ์พันธุ์ เลาวพงศ์

วันที่ 22/09/2567

สารบัญ

โค้ดของโปรแกรม HEAP SORT พร้อมคำอธิบาย

```
#include <stdio.h> //ใช้ printf

#include <conio.h> //ใช้ getch

#include <stdlib.h> //ใช้ random

#include <time.h> //ใช้ time

#define MaxData 100 // กำหนด Max Data (ข้อมูลสูงสุด)

int Data1[MaxData], Data2[MaxData];

int N;

void PrepareRawData(int N)

{

    int i;

    srand(time(NULL)); //สำหรับสร้างตัวเลขสุ่มที่แตกต่างกันใน rand()

    for (i=1;i<=N;i++)

        Data1[i]=1+rand() % 99; //สุ่มตัวเลขที่แตกต่างกัน 1..99

}

void DispData(int Data[],int out) // out คือจุดที่แสดงตัวเลขที่ Output ย้อนกลับ

{

    int i;

    for(i=1;i<=N;i++)

    {

        if(i<out)

        printf("%2d   ",Data[i]); //แสดงตัวเลขความกว้าง 2
```

โค้ดของโปรแกรม HEAP SORT พร้อมคำอธิบาย (ต่อ)

```
        else
        printf("[%2d] ",Data[i]); //แสดง [ ] ถ้าเป็น 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) //หาค่ามากที่สุดจากข้อมูล 2 ตัว
{
    if(a>b)
        return(a);
    else
        return(b);
}
void AdjustTree(int LastNode)
{
```

โค้ดของโปรแกรม HEAP SORT พร้อมคำอธิบาย (ต่อ)

```
int i,Max,lson,rson,son;
bool result;
i=1;
result=false; // false คือยังไม่เสร็จสิ้นการปรับปรุง
while(!result)
{
    lson=(2*i); //คำนวณลูกด้านซ้าย (LSon)
    rson=(2*i)+1; //คำนวณลูกด้านขวา (RSon)
    son=0; //ตั้งค่าเริ่มต้นของลูกเป็น 0
    if(lson==LastNode)
    {
        son=1;
        if(Data2[i]<Data2[lson]) //ตรวจสอบว่าข้อมูลพ่อ < ข้อมูลลูกด้านซ้ายหรือไม่?
        {
            swap(i,lson);
            DispData(Data2,LastNode+1); //แสดงผลลัพธ์ของแต่ละขั้นตอน
        }
        result=true; //เสร็จสิ้นการปรับปรุง
    }
    if(rson<=LastNode)
    {
```

โค้ดของโปรแกรม HEAP SORT พร้อมคำอธิบาย (ต่อ)

```
son=2;
Max=Maximum(Data2[lson],Data2[rson]); //หาข้อมูลที่มากที่สุด
if(Data2[i]<Max) //ตรวจสอบว่าข้อมูลพ่อ < Max หรือไม่?
{
    if(Max==Data2[lson]) //Max == ข้อมูลลูกด้านซ้ายหรือไม่?
    {
        swap(i,lson);
        DispData(Data2,LastNode+1); //แสดงผลลัพธ์ของแต่ละขั้นตอน
        if(rson==LastNode) //ตรวจสอบว่าถึงโหนดสุดท้ายหรือไม่
            result=true; //เสร็จสิ้นการปรับปรุง
        else
            i=lson; //ให้ i ตามลูกด้านซ้าย
    }
    else //ถ้าข้อมูลลูกด้านขวามีค่ามากที่สุด
    {
        swap(i,rson);
        DispData(Data2,LastNode+1); //แสดงผลลัพธ์ของแต่ละขั้นตอน
        if(rson==LastNode) //ตรวจสอบว่าถึงโหนดสุดท้ายหรือไม่
            result=true; //เสร็จสิ้นการปรับปรุง
        else
            i=rson; //ให้ i ตามลูกด้านขวา
```

โค้ดของโปรแกรม HEAP SORT พร้อมคำอธิบาย (ต่อ)

```
            }
        }
        else
            result=true; //เสร็จสิ้นการปรับปรุง
    }
    if(son==0)
        result=true; //เสร็จสิ้นการปรับปรุง
} //จบ While
printf("-----------------------------------------------Adjust Tree Finished at N=%d \n",LastNode);
} //จบฟังก์ชัน
void CreateHeapTree() // สร้างจาก Data1 เป็น Data2
{
    int i,j,k,father;
    bool result;
    //สร้าง Heap Tree
    Data2[1]=Data1[1]; //โหนดแรกของ Heap Tree
    DispData(Data2,N+1); //แสดงผลลัพธ์ของแต่ละขั้นตอน
    for(i=2;i<=N;i++)
    {
        Data2[i]=Data1[i];
        DispData(Data2,N+1); //แสดงผลลัพธ์ของแต่ละขั้นตอน
```

โค้ดของโปรแกรม HEAP SORT พร้อมคำอธิบาย (ต่อ)

```
    result=true;
    j=i; //ตั้งตัวนับย้อนกลับเริ่มที่นี่
    while(result)
    {
        father=j/2; //คำนวณพ่อ
        if((Data2[j]>Data2[father]) && (j>1)) //ปรับ Heap tree
        {
            swap(j,father);
            DispData(Data2,N+1); //แสดงผลลัพธ์ของแต่ละขั้นตอน
            j=father; //ให้ j ตามพ่อใหม่
            result=true;
        }
        else
            result=false;
    } //จบ While
} //จบ for
printf("-------------------------------------------------Create Heap Tree Finished \n");
for(k=1;k<=N;k++) //แสดงดัชนีอาเรย์
    printf("(%d)  ",k);
printf("\n");
for(i=N;i>1;i--)
```

โค้ดของโปรแกรม HEAP SORT พร้อมคำอธิบาย (ต่อ)

```
    {
        swap(1,i); //นำโหนดรากออก
        DispData(Data2,i); //แสดงผลลัพธ์ของแต่ละขั้นตอน
        AdjustTree(i-1); //เรียกฟังก์ชันปรับปรุง Heap Tree
    } //จบ for
} //จบฟังก์ชัน
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); //ข้อมูลที่เรียงลำดับแล้ว
    printf("-----------------------------------------------Sort Finished \n");
    getch();
```

โค้ดของโปรแกรม HEAP SORT พร้อมคำอธิบาย (ต่อ)

```
    return(0);

} //จบ Main
```

หลักการทำงานของโปรแกรม HEAP SORT

โปรแกรม HEAP SORT เป็นโปรแกรมที่ใช้เทคนิค Heap Sort ในการจัดเรียงข้อมูล โดยมีการจัดเรียงข้อมูลจากน้อยไปหามาก (ascending order) ผ่านกระบวนการสร้างต้นไม้ฮีพ (Heap Tree) จากข้อมูลที่สุ่มขึ้นมา โปรแกรมนี้ใช้โครงสร้างของข้อมูลแบบอาร์เรย์ (Array) โดยมีการแยกขั้นตอนออกเป็นหลายฟังก์ชัน แต่ละฟังก์ชันมีหน้าที่เฉพาะเจาะจงในกระบวนการเรียงลำดับ ฟังก์ชันเหล่านี้รวมถึงการเตรียมข้อมูลดิบ การแสดงข้อมูล การสลับข้อมูล การปรับโครงสร้างฮีพ และการสร้างต้นไม้ฮีพ

1. การนำเข้าไลบรารี

```
#include <stdio.h> //ใช้ printf

#include <conio.h> //ใช้ getch

#include <stdlib.h> //ใช้ random

#include <time.h> //ใช้ time
```

ในส่วนของการนำเข้าไลบรารี (#include) จะมีรายละเอียดดังนี้

- <stdio.h> : ไลบรารีนี้ใช้สำหรับฟังก์ชันการรับและแสดงผลข้อมูล เช่น printf() ที่ใช้ในการพิมพ์ข้อความออกทางหน้าจอ และ scanf() ที่ใช้สำหรับการรับข้อมูลจากผู้ใช้
- <conio.h> : ไลบรารีนี้ใช้ในการทำงานกับการอินพุตจากคีย์บอร์ดในรูปแบบที่ง่ายขึ้น เช่น getch() ซึ่งใช้เพื่อรอให้ผู้ใช้กดปุ่มก่อนที่จะดำเนินการต่อ
- <stdlib.h> : ไลบรารีนี้มีฟังก์ชันที่เกี่ยวข้องกับการจัดการหน่วยความจำ การแปลงค่า และการสุ่ม เช่น rand() ที่ใช้สำหรับสร้างค่าตัวเลขสุ่ม
- <time.h> : ไลบรารีนี้มีฟังก์ชันที่เกี่ยวข้องกับเวลาและวันที่ เช่น time() ที่ใช้เพื่อรับค่าชั่วโมง นาที และวินาทีในรูปแบบ timestamp

2. การกำหนดค่าคงที่

```
#define MaxData 100 // กำหนด Max Data (ข้อมูลสูงสุด)
```

ในส่วนของการกำหนดค่าคงที่ จะมีรายละเอียดดังนี้

- #define MaxData 100 : การใช้คำสั่ง #define นี้ใช้เพื่อกำหนดค่าคงที่ (constant) ในโปรแกรม โดย MaxData กำหนดค่าที่ 100 ซึ่งเป็นการกำหนดขนาดสูงสุดของอาร์เรย์ Data[] ในโปรแกรม ค่าคงที่นี้สามารถถูกใช้ในหลายส่วนของโปรแกรม เช่น การวนลูปหรือจัดการข้อมูล เพื่อให้แน่ใจว่าอาร์เรย์ Data[] จะไม่เกินขนาดที่กำหนด (100 ข้อมูล)

3. การประกาศตัวแปร

```
int Data1[MaxData], Data2[MaxData];

int N;
```

ในส่วนของการประกาศตัวแปร จะมีรายละเอียดดังนี้

- เป็นการประกาศตัวแปรอาร์เรย์สองตัว คือ Data1 และ Data2 ซึ่งทั้งสองตัวจะมีขนาด MaxData (100 ตำแหน่ง) โดย Data1[MaxData] เก็บข้อมูลดิบที่สุ่มขึ้นมา (ยังไม่ได้เรียงลำดับ) และ Data2[MaxData] เก็บข้อมูลที่ผ่านการจัดโครงสร้างฮีพ (Heap) และจัดเรียงแล้ว
- int N : ตัวแปร N เป็นตัวแปรชนิด int (จำนวนเต็ม) ที่ใช้เก็บจำนวนข้อมูลที่โปรแกรมจะทำการสุ่มและเรียงลำดับ โดยค่าของ N จะถูกกำหนดในฟังก์ชัน main() ในโค้ดตัวอย่างนี้กำหนดให้ N = 8 ซึ่งหมายถึงจะสุ่มข้อมูลจำนวน 8 ตัว

4. ฟังก์ชัน PrepareRawData

```
void PrepareRawData(int N)

{

    int i;

    srand(time(NULL)); //สำหรับสร้างตัวเลขสุ่มที่แตกต่างกันใน rand()

    for (i=1;i<=N;i++)

        Data1[i]=1+rand() % 99; //สุ่มตัวเลขที่แตกต่างกัน 1..99

}
```

ฟังก์ชัน PrepareRawData มีหน้าที่สุ่มข้อมูลดิบแบบตัวเลข เพื่อใส่ลงไปในอาร์เรย์ Data1 จำนวน N ตัว ซึ่งจะถูกใช้ในกระบวนการเรียงลำดับภายหลัง โดยหลักการทำงานจะใช้ srand(time(NULL)) เพื่อทำให้การสุ่ม

ตัวเลขมีความแตกต่างกันทุกครั้งที่รันโปรแกรม ฟังก์ชันนี้จะสร้างเมล็ดสุ่ม (seed) โดยอ้างอิงจากเวลาปัจจุบัน จากนั้นจะใช้ลูป for เพื่อวนรอบตั้งแต่ i = 1 ถึง i = N โดยแต่ละรอบจะทำการสุ่มตัวเลขระหว่าง 1 ถึง 99 ด้วยฟังก์ชัน rand() จากนั้นนำไปเก็บในอาร์เรย์ Data1 ที่ตำแหน่ง i ตัวอย่างเช่น ถ้า N = 5 โปรแกรมจะสุ่มตัวเลข 5 ตัว เช่น 45, 12, 78, 23, และ 9 จากนั้นเก็บค่าทั้งหมดใน Data1[1..5]

5. ฟังก์ชัน DispData

```
void DispData(int Data[],int out) // out คือจุดที่แสดงตัวเลขที่ Output ย้อนกลับ
{
    int i;
    for(i=1;i<=N;i++)
    {
        if(i<out)
        printf("%2d   ",Data[i]); //แสดงตัวเลขความกว้าง 2
        else
        printf("[%2d] ",Data[i]); //แสดง [ ] ถ้าเป็น Output
    }
    printf("\n");
}
```

ฟังก์ชัน DispData ใช้สำหรับการแสดงข้อมูลในอาร์เรย์ โดยข้อมูลจะถูกแสดงออกมาทีละตัวในลำดับที่อยู่ในอาร์เรย์ Data[] และมีการแสดงเครื่องหมายพิเศษ ([]) รอบตัวเลขที่ถูกจัดว่าเป็น output (ตัวเลขที่ถูกนำออกในขั้นตอนการเรียงลำดับ) โดยหลักการทำงานเริ่มต้นจากจะใช้งานลูป for จะวนรอบตั้งแต่ i = 1 ถึง i = N เพื่อพิมพ์ข้อมูลทีละตัว ถ้าค่าของ i น้อยกว่า out จะพิมพ์ค่าตัวเลขธรรมดาด้วยการจัดให้แสดงกว้าง 2 ตำแหน่ง (%2d), ถ้าค่าของ i เท่ากับหรือตำแหน่ง out จะพิมพ์ข้อมูลในรูปแบบ [ ] เพื่อบอกว่าตัวเลขนั้นถูกนำออกแล้ว ตัวอย่างเช่น ถ้า Data[] = {25, 40, 17, 68} และ out = 4 ผลลัพธ์ที่จะแสดงคือ 25 40 17 [68] ซึ่งหมายความว่าตัวเลข 68 ถูกนำออกมาในขั้นตอนการจัดเรียง

6. ฟังก์ชัน swap

```
void swap(int a,int b)
{
    int temp;
    temp=Data2[a];
    Data2[a]=Data2[b];
    Data2[b]=temp;
}
```

ฟังก์ชัน swap ทำหน้าที่สลับตำแหน่งของข้อมูลระหว่างตำแหน่ง a และ b ในอาร์เรย์ Data2 โดยหลักการทำงานจะใช้ตัวแปร temp เพื่อเก็บค่าชั่วคราวของตำแหน่ง a ก่อนจะทำการสลับค่าของตำแหน่ง a และ b ซึ่งเป็นฟังก์ชันที่ถูกเรียกใช้บ่อยในการสร้างและปรับโครงสร้างฮีพ เมื่อข้อมูลในฮีพมีการสลับตำแหน่งตามเงื่อนไขต่างๆ ตัวอย่างเช่น ถ้า a = 1 และ b = 3 และ Data2 = {10, 20, 30} หลังจากการเรียกใช้ swap(1, 3) อาร์เรย์จะกลายเป็น Data2 = {30, 20, 10}

7. ฟังก์ชัน Maximum

```
int Maximum(int a, int b) //หาค่ามากที่สุดจากข้อมูล 2 ตัว
{
    if(a>b)
        return(a);
    else
        return(b);
}
```

ฟังก์ชัน Maximum ทำหน้าที่เปรียบเทียบระหว่างค่าของ a และ b แล้วคืนค่าที่มากกว่าออกมา ใช้ในการเปรียบเทียบค่าของลูกซ้าย (LSon) และลูกขวา (RSon) เพื่อหาค่าที่มากที่สุดระหว่างทั้งสอง

ตัวอย่างเช่น ถ้า a = 15 และ b = 25 ฟังก์ชันจะคืนค่า 25 ซึ่งเป็นค่าที่มากกว่า

8. ฟังก์ชัน AdjustTree

```
void AdjustTree(int LastNode)
{
    int i,Max,lson,rson,son;
    bool result;
    i=1;
    result=false; // false คือยังไม่เสร็จสิ้นการปรับปรุง
    while(!result)
    {
        lson=(2*i); //คำนวณลูกด้านซ้าย (LSon)
        rson=(2*i)+1; //คำนวณลูกด้านขวา (RSon)
        son=0; //ตั้งค่าเริ่มต้นของลูกเป็น 0
        if(lson==LastNode)
        {
            son=1;
            if(Data2[i]<Data2[lson]) //ตรวจสอบว่าข้อมูลพ่อ < ข้อมูลลูกด้านซ้ายหรือไม่?
            {
                swap(i,lson);
                DispData(Data2,LastNode+1); //แสดงผลลัพธ์ของแต่ละขั้นตอน
            }
            result=true; //เสร็จสิ้นการปรับปรุง
```

8. ฟังก์ชัน AdjustTree (ต่อ)

```
    }
    if(rson<=LastNode)
    {
        son=2;
        Max=Maximum(Data2[lson],Data2[rson]); //หาข้อมูลที่มากที่สุด
        if(Data2[i]<Max) //ตรวจสอบว่าข้อมูลพ่อ < Max หรือไม่?
        {
            if(Max==Data2[lson]) //Max == ข้อมูลลูกด้านซ้ายหรือไม่?
            {
                swap(i,lson);
                DispData(Data2,LastNode+1); //แสดงผลลัพธ์ของแต่ละขั้นตอน
                if(rson==LastNode) //ตรวจสอบว่าถึงโหนดสุดท้ายหรือไม่
                    result=true; //เสร็จสิ้นการปรับปรุง
                else
                    i=lson; //ให้ i ตามลูกด้านซ้าย
            }
            else //ถ้าข้อมูลลูกด้านขวามีค่ามากที่สุด
            {
                swap(i,rson);
                DispData(Data2,LastNode+1); //แสดงผลลัพธ์ของแต่ละขั้นตอน
                if(rson==LastNode) //ตรวจสอบว่าถึงโหนดสุดท้ายหรือไม่
```

8. ฟังก์ชัน AdjustTree (ต่อ)

```
                    result=true; //เสร็จสิ้นการปรับปรุง
                else
                    i=rson; //ให้ i ตามลูกด้านขวา
            }
        }
        else
            result=true; //เสร็จสิ้นการปรับปรุง
    }
    if(son==0)
        result=true; //เสร็จสิ้นการปรับปรุง
  } //จบ While
  printf("-----------------------------------------------Adjust Tree Finished at N=%d \n",LastNode);
} //จบฟังก์ชัน
```

ฟังก์ชัน AdjustTree เป็นฟังก์ชันหลักในการปรับโครงสร้างของฮีพ (Heap Tree) เพื่อให้ข้อมูลในฮีพถูกต้องตามเงื่อนไขที่ว่า "พ่อ" (Father) ต้องมีค่ามากกว่าลูกซ้าย (LSon) และลูกขวา (RSon) หลังจากที่โหนดราก (Root Node) ถูกนำออกมา โดยหลักการทำงานเริ่มต้นจากตำแหน่งโหนดพ่อ (i = 1) จากนั้นจะคำนวณหาตำแหน่งลูกซ้าย (lson = 2 * i) และลูกขวา (rson = 2 * i + 1) เพื่อเปรียบเทียบค่าของพ่อกับลูกซ้ายและลูกขวา ถ้าลูกคนใดมีค่ามากกว่าพ่อ จะทำการสลับค่ากับพ่อ โดยกระบวนการจะทำซ้ำจนกว่าฮีพจะถูกปรับโครงสร้างให้ถูกต้อง หรือจนกว่าโหนดที่เหลือจะอยู่ในลำดับที่ถูกต้อง ตัวอย่างเช่น ถ้าโหนดรากมีค่าน้อยกว่าลูกข้างใดข้างหนึ่ง ฟังก์ชันนี้จะทำการสลับตำแหน่งจนกว่าฮีพจะกลับมาอยู่ในรูปแบบที่ถูกต้องตามกฎ

9. ฟังก์ชัน CreateHeapTree

```
void CreateHeapTree() // สร้างจาก Data1 เป็น Data2
{
    int i,j,k,father;
    bool result;
    //สร้าง Heap Tree
    Data2[1]=Data1[1]; //โหนดแรกของ Heap Tree
    DispData(Data2,N+1); //แสดงผลลัพธ์ของแต่ละขั้นตอน
    for(i=2;i<=N;i++)
    {
        Data2[i]=Data1[i];
        DispData(Data2,N+1); //แสดงผลลัพธ์ของแต่ละขั้นตอน
        result=true;
        j=i; //ตั้งตัวนับย้อนกลับเริ่มที่นี่
        while(result)
        {
            father=j/2; //คำนวณพ่อ
            if((Data2[j]>Data2[father]) && (j>1)) //ปรับ Heap tree
            {
                swap(j,father);
                DispData(Data2,N+1); //แสดงผลลัพธ์ของแต่ละขั้นตอน
                j=father; //ให้ j ตามพ่อใหม่
```

9. ฟังก์ชัน CreateHeapTree (ต่อ)

```
            result=true;
        }
        else
            result=false;
    } //จบ While
} //จบ for
printf("-------------------------------------------------Create Heap Tree Finished \n");
for(k=1;k<=N;k++) //แสดงดัชนีอาเรย์
    printf("(%d)  ",k);
printf("\n");
for(i=N;i>1;i--)
{
    swap(1,i); //นำโหนดรากออก
    DispData(Data2,i); //แสดงผลลัพธ์ของแต่ละขั้นตอน
    AdjustTree(i-1); //เรียกฟังก์ชันปรับปรุง Heap Tree
} //จบ for
} //จบฟังก์ชัน
```

ฟังก์ชัน CreateHeapTree ทำหน้าที่สร้างต้นไม้ฮีพจากข้อมูลดิบในอาร์เรย์ Data1 และเก็บผลลัพธ์ไว้ในอาร์เรย์ Data2 โดยแต่ละโหนดจะถูกเพิ่มลงในต้นไม้ทีละโหนด และมีการปรับโครงสร้างฮีพให้ถูกต้องทุกครั้งที่มีการเพิ่มโหนดใหม่ หลักการทำงานจะเริ่มจาก นำข้อมูลจาก Data1[1] มาใส่ใน Data2[1] ซึ่งเป็นโหนดรากสำหรับโหนดถัดไป ฟังก์ชันจะนำข้อมูลจาก Data1 ใส่ลงไปใน Data2 แล้วปรับฮีพให้ถูกต้องผ่านการสลับตำแหน่งและเปรียบเทียบค่าของโหนดพ่อกับลูกซ้ายและลูกขวา เมื่อสร้างฮีพเสร็จสิ้น จะเริ่มต้นกระบวนการ

สลับราก (Root Node) ออกมาเป็นผลลัพธ์ และปรับโครงสร้างฮีพที่เหลือโดยเรียกฟังก์ชัน AdjustTree() ตัวอย่างเช่น หากมีข้อมูลใน Data1 เช่น [30, 20, 50, 10, 40] ฟังก์ชันนี้จะสร้างฮีพขึ้นมาใน Data2 โดยในระหว่างการสร้าง ฮีพจะถูกปรับให้โหนดพ่อมีค่ามากกว่าลูกเสมอ จนข้อมูลในฮีพถูกเรียงตามลำดับ

10. ฟังก์ชัน main

```
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); //ข้อมูลที่เรียงลำดับแล้ว
    printf("--------------------------------------------------Sort Finished \n");
    getch();
    return(0);
} //จบ Main
```

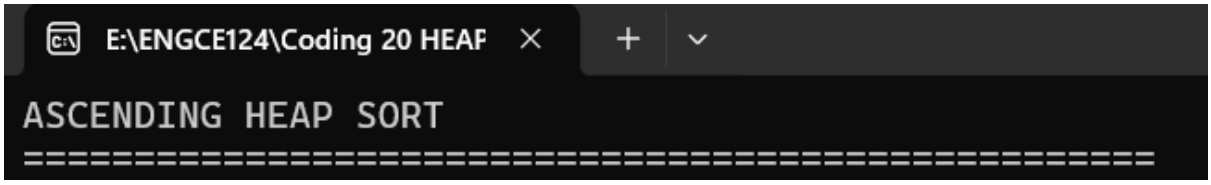
ฟังก์ชัน main เป็นจุดเริ่มต้นของโปรแกรม ซึ่งมีหลักการทำงานเริ่มต้นจาก กำหนดจำนวนข้อมูลที่จะสุ่ม (N=8) จากนั้นจะเรียกฟังก์ชัน PrepareRawData() เพื่อสุ่มข้อมูลและนำข้อมูลมาเก็บใน Data1 และแสดงข้อมูลดิบที่สุ่มขึ้นมาโดยเรียก DispData() ต่อมาจะเรียกฟังก์ชัน CreateHeapTree() เพื่อสร้างต้นไม้ฮีพจากข้อมูลดิบสุดท้ายแสดงข้อมูลที่ถูกจัดเรียงแล้วจากอาร์เรย์ Data2

ผลลัพธ์การใช้งานโปรแกรม HEAP SORT

โปรแกรม HEAP SORT นี้ออกแบบมาเพื่อสุ่มข้อมูลจำนวนหนึ่ง จากนั้นทำการจัดเรียงข้อมูลเหล่านั้นด้วยวิธี Heap Sort ซึ่งเป็นอัลกอริทึมการจัดเรียงแบบลำดับจากน้อยไปมาก (ascending) ในขั้นตอนนี้จะมีการแสดงผลการทำงานในแต่ละขั้นตอน เพื่อให้เห็นภาพของการจัดเรียงข้อมูลในแต่ละลำดับชัดเจนมากขึ้น

1. การเริ่มต้นโปรแกรม

เมื่อเริ่มต้นรันโปรแกรม จะมีการพิมพ์ข้อความ “ASCENDING HEAP SORT” เพื่อแจ้งให้ผู้ใช้ทราบว่าโปรแกรมกำลังดำเนินการเรียงลำดับข้อมูลแบบ Heap Sort โดยจัดเรียงจากน้อยไปมาก (Ascending)

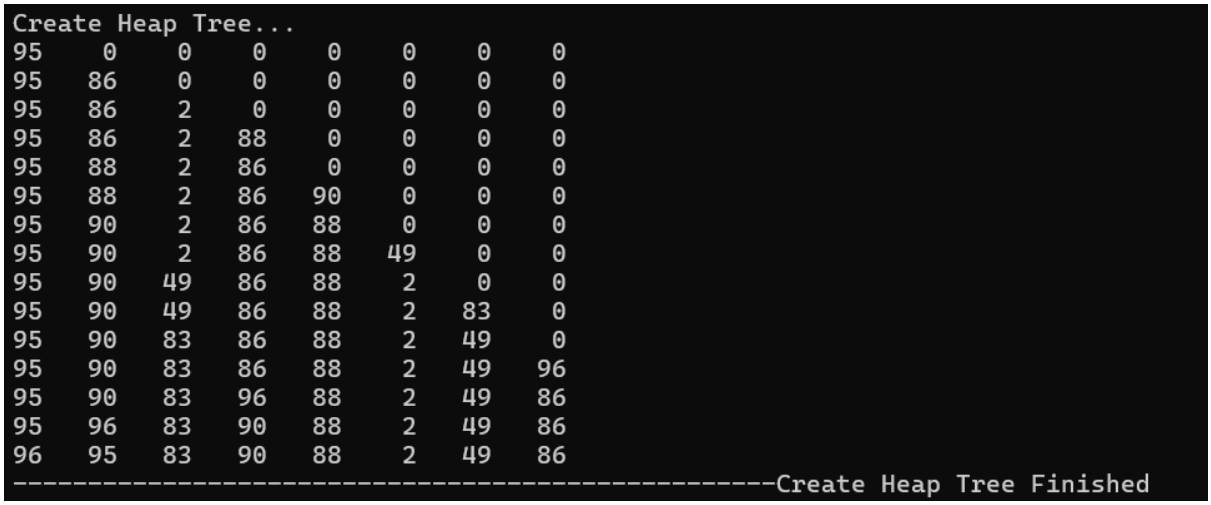


2. กำหนดจำนวนข้อมูลและสุ่มข้อมูลดิบ

โปรแกรมกำหนดให้สุ่มข้อมูลจำนวน N = 8 ซึ่งหมายถึงจะมีข้อมูลดิบจำนวน 8 ตัว โดยข้อมูลแต่ละตัวจะเป็นตัวเลขที่สุ่มจากช่วง 1 ถึง 99 โดยข้อมูลที่สุ่มคือข้อมูลดิบที่ยังไม่ได้ผ่านการจัดเรียงใดๆ

3. สร้างต้นไม้ฮีพ (Heap Tree)

โปรแกรมจะเริ่มทำการสร้างต้นไม้ฮีพ (Heap Tree) จากข้อมูลดิบ โดยในระหว่างการสร้างจะมีการแสดงผลทีละขั้นตอนว่าแต่ละค่าถูกย้ายหรือสลับที่กันอย่างไร ซึ่งกระบวนการนี้จะแสดงให้เห็นว่าข้อมูลแต่ละตำแหน่งในต้นไม้ถูกสลับตำแหน่งเพื่อจัดโครงสร้างฮีพอย่างไร และแสดงเมื่อกระบวนการสร้างฮีพเสร็จสิ้น



4. การจัดเรียงข้อมูล (Sorting)

หลังจากสร้างต้นไม้ฮีพแล้ว โปรแกรมจะเริ่มทำการจัดเรียงข้อมูลโดยการดึงค่าสูงสุดออกจากต้นไม้ และทำการปรับโครงสร้างฮีพใหม่ซ้ำไปเรื่อยๆ จนกว่าจะได้ข้อมูลที่เรียงลำดับเสร็จสิ้น ระหว่างการจัดเรียง จะมีการแสดงผลการสลับที่ของข้อมูลในแต่ละขั้นตอน โดยในแต่ละขั้นตอน ข้อมูลที่ถูกจัดเรียงแล้วจะถูกใส่เครื่องหมาย [ ] เพื่อให้เห็นว่าข้อมูลใดถูกจัดเรียงออกจากฮีพเรียบร้อยแล้ว

```
(1)  (2)  (3)  (4)  (5)  (6)  (7)  (8)
86   95   83   90   88    2   49   [96]
95   86   83   90   88    2   49   [96]
95   90   83   86   88    2   49   [96]
-------------------------------------------------Adjust Tree Finished at N=7
49   90   83   86   88    2   [95] [96]
90   49   83   86   88    2   [95] [96]
90   88   83   86   49    2   [95] [96]
-------------------------------------------------Adjust Tree Finished at N=6
 2   88   83   86   49   [90] [95] [96]
88    2   83   86   49   [90] [95] [96]
88   86   83    2   49   [90] [95] [96]
-------------------------------------------------Adjust Tree Finished at N=5
49   86   83    2   [88] [90] [95] [96]
86   49   83    2   [88] [90] [95] [96]
-------------------------------------------------Adjust Tree Finished at N=4
 2   49   83   [86] [88] [90] [95] [96]
83   49    2   [86] [88] [90] [95] [96]
-------------------------------------------------Adjust Tree Finished at N=3
 2   49   [83] [86] [88] [90] [95] [96]
49    2   [83] [86] [88] [90] [95] [96]
-------------------------------------------------Adjust Tree Finished at N=2
 2   [49] [83] [86] [88] [90] [95] [96]
-------------------------------------------------Adjust Tree Finished at N=1
```

5. การแสดงผลลัพธ์ของข้อมูล

หลังจากการจัดเรียงเสร็จสิ้น โปรแกรมจะแสดงข้อมูลที่ถูกจัดเรียงเรียบร้อยจากน้อยไปมาก โดยข้อมูลในอาร์เรย์ Data2[] จะถูกเรียงจากค่าน้อยไปหาค่ามากเรียบร้อยแล้ว โดยค่าที่น้อยที่สุดจะถูกใส่ในเครื่องหมาย [ ]

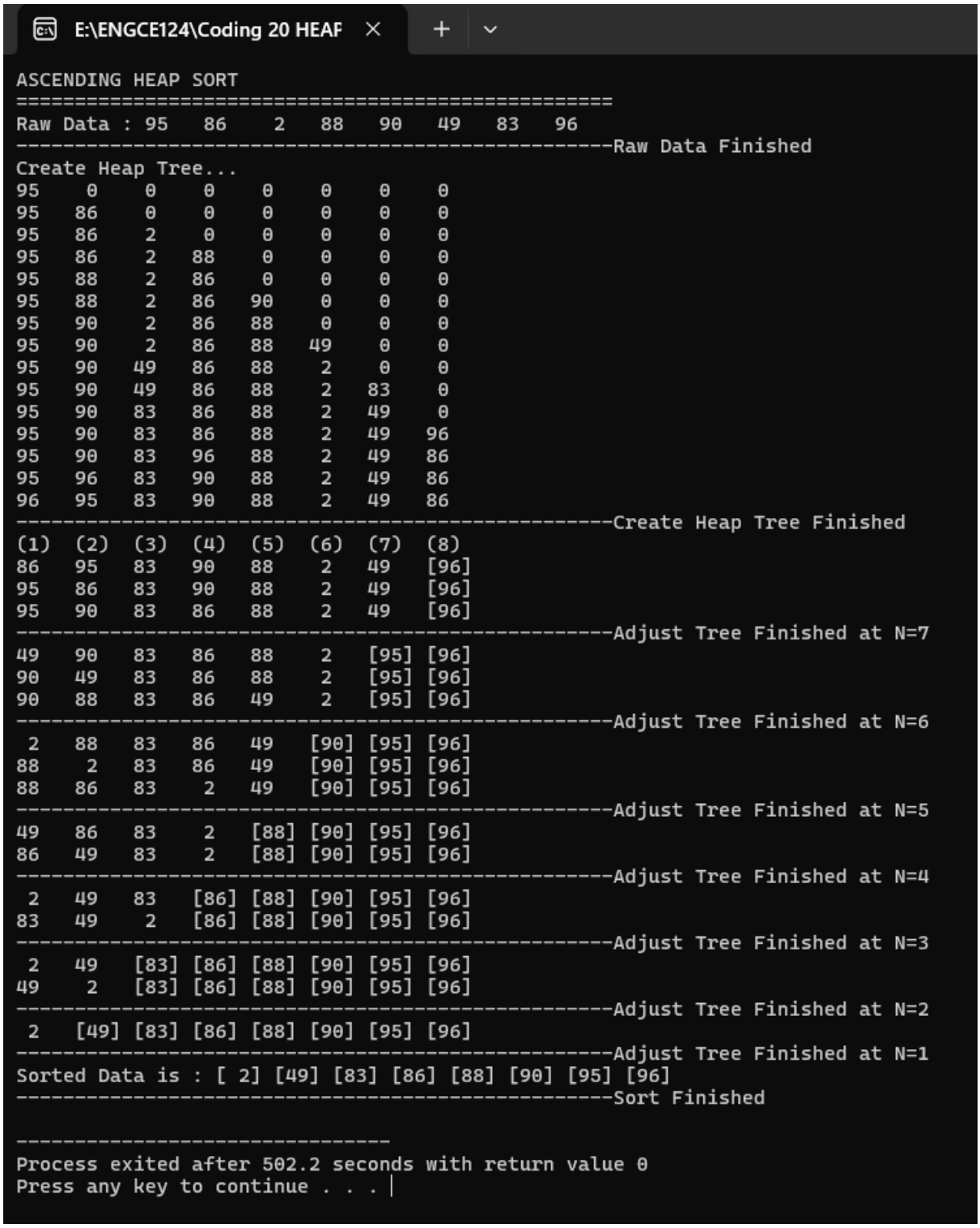
```
Sorted Data is : [ 2] [49] [83] [86] [88] [90] [95] [96]
-------------------------------------------------Sort Finished
```

6. การจบการทำงานของโปรแกรม

โปรแกรมจะหยุดการทำงานชั่วคราวและรอการกดปุ่มใดๆ จากผู้ใช้ก่อนที่จะปิดโปรแกรม (ผ่านฟังก์ชัน getch()) ซึ่งเป็นการจบการทำงานของโปรแกรม

```
--------------------------------
Process exited after 502.2 seconds with return value 0
Press any key to continue . . .
```

ภาพรวมผลลัพธ์ของโปรแกรม HEAP SORT



บรรณานุกรม

ChatGPT. (-). Heap Sort: A Detailed Program Explanation. สืบค้น 22 กันยายน 2567, จาก https://chatgpt.com/