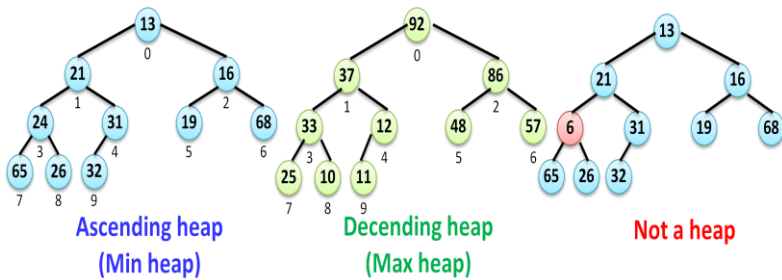


Lab 8 : Heap & Lab 9 : Sort

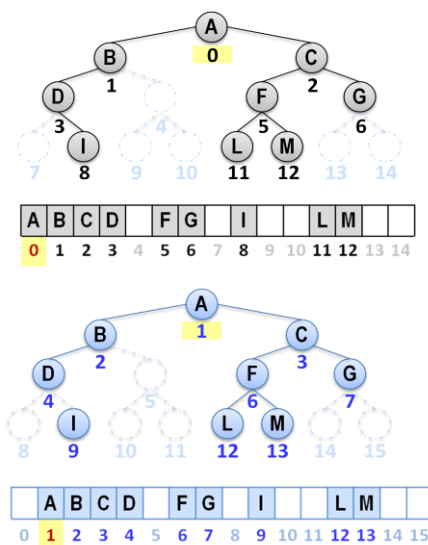
วัตถุประสงค์ ฝึกหัดเขียน Sort แบบต่างๆ



Binary Heap : complete Binary Tree ซึ่ง key ของ node ใดๆ

1. **\leq key** ของ **decendents** ของ มัน : **Min heap**
เช่น 13 น้อยกว่าลูกหลานทั้งหมดของมัน
2. **\geq key** ของ **decendents** ของ มัน : **Max heap**
เช่น 37 มากกว่าลูกหลานทั้งหมดของมัน

เนื่องจากเป็น complete binary tree จึงควร implement ด้วย data structure แบบ _____



ซึ่งโครงสร้างแบบ implicit array (sequential array) นี้ เรามักนับตำแหน่งของ node นิยมทำกับ 2 แบบ คือ ให้ root เริ่มที่ 0 (รูปซ้าย) และ ให้ root เริ่มที่ 1 (รูปขวา) นับตำแหน่งไปเรื่อยๆ เรียงตามลำดับที่ละ level จากซ้ายไปขวา นับไม่เว้น node ที่ไม่มี (ในรูปเป็นเส้นประ) แต่ละ node จึงมีตำแหน่งกำกับดังรูป และใช้ implicit array (sequential array) เก็บข้อมูล data ของแต่ละ node ใน index ตามตำแหน่งของมัน

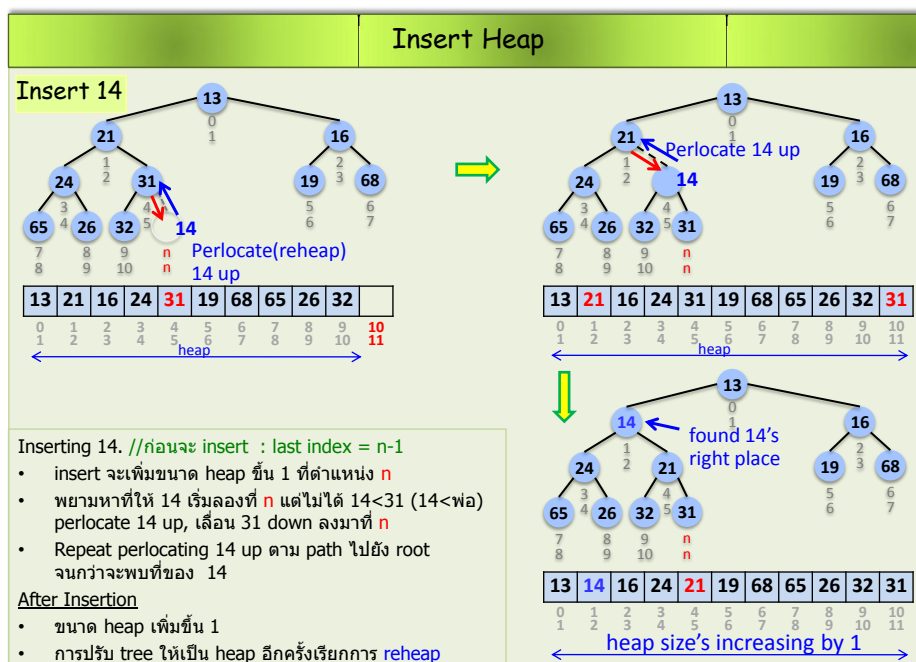
จะเห็นว่า data structure แบบ implicit array นี้ เก็บเฉพาะ data ไม่จำเป็นต้องเก็บ link เนื่องจากสามารถคำนวณได้

เริ่ม root ที่ index 0 เริ่ม root ที่ index 1

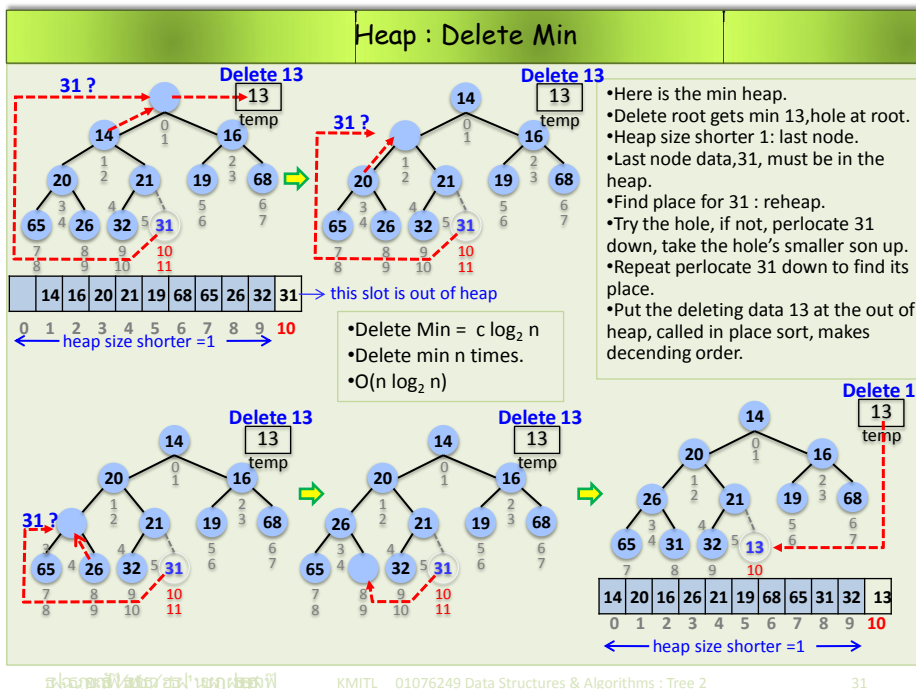
ลูกข้างซ้ายของ node ที่ index i อยู่ที่ index

„ ໂພນ „

”



Algorithm ในการสร้าง heap ใช้วิธี insert data เข้าไปใน heap ที่ละตัว ซึ่งทำให้ size ของ heap เพิ่มขึ้นจาก index สุดท้ายไป 1 ตัว ในรูปกำลัง insert 14 index ที่เพิ่มขึ้น n คือ 10 (root เริ่มที่ index 0) / 11 (root เริ่มที่ index 1) โดยดูว่า สามารถ insert data เข้าที่ตำแหน่ง n ได้หรือไม่ (ผิดกฎของ heap หรือไม่) หากไม่ได้ จะทำการ perlocate data up ขึ้นไปตามทางพ่อของมันไปยัง root ทำให้การ insert data 1 ตัวทำงานอย่างมากเท่ากับความสูงของ tree คือ $\log_2 n$ ดังนั้น insert n ตัว มีลำดับเป็น



min heap ข้อมูลตัวที่น้อยที่สุดอยู่ที่ _____ การ delete min คือการดึง root ออกมา และทำการ reheap (จัดข้อมูลที่เหลือให้เป็น heap ใหม่) ข้อมูลที่น้อยเป็นลำดับถัดมาจะขึ้นไปอยู่ที่ root ดังนั้น หากเรา delete min ออกมาเรื่อยๆ จนหมด heap เราจะได้ข้อมูลที่ sorted จากน้อยไปมาก ascending order การทำ in place sort คือ delete min แล้วนำมาใส่ไว้ใน array ตัวเดิม โดยใส่ไว้ที่ index ตัวสุดท้ายของ heap ซึ่งจะเป็นตำแหน่งที่หายจาก heap เมื่อ delete data ออก หาก delete min ออกจนหมด การทำ in place sort บน min heap จะให้ array ที่เรียงแบบ

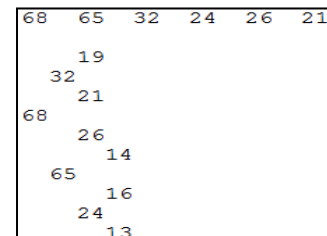
_____ order จากมากไปน้อย

การทดลอง 1 Heap Sort : ต้องมี array สำหรับเป็น data structure ของ heap ในภาษา Python ใช้ Python list ข้อมูลที่เก็บใน heap เป็น class ใดๆ เช่น class student class book ... ซึ่งมี key เพื่อใช้เป็นแทนของแต่ละ record ใน heap สำหรับในการทดลอง เพื่อให้ง่ายขึ้น ให้ทำ in place heap sort โดยใช้ python list เก็บ key ที่เป็น integer (แต่ให้เข้าใจว่าความจริงแล้ว heap เก็บทั้ง record ของข้อมูลได้ หรือ อาจเก็บเฉพาะ key แล้วมี data structure อื่นเสริมเพื่อ link key ไปยัง record อีกทีก็ได้ ดังนั้นข้างล่างจึงใช้คำว่า type T ใดๆ หมายถึง class ต่างๆ ที่เป็นข้อมูล) ให้เขียนฟังก์ชันต่อไปน้โดยคิด parameter เอง พร้อมทั้งทดสอบความถูกต้องด้วย (คำว่า array a ข้างล่างหมายถึง python list a ใช้เป็น input data)

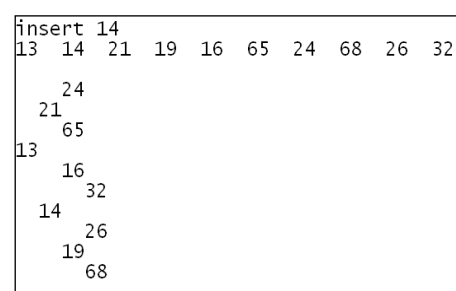
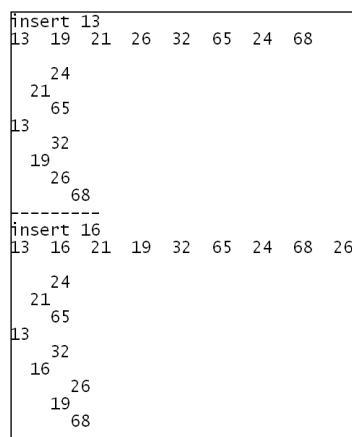
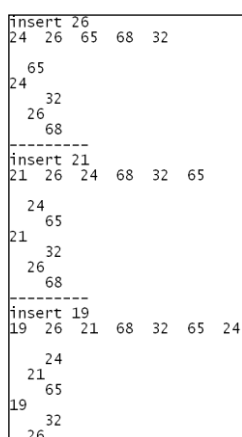
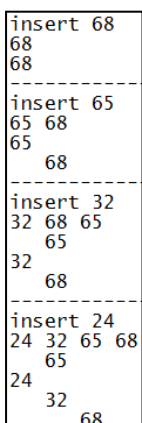
1) เพื่อทดสอบความถูกต้องของ heap เขียนฟังก์ชันเพื่อพิมพ์ tree

- print () พิมพ์ array a ของ type T ใดๆ ตั้งแต่ index 0 ถึง i (python ไม่ต้องเขียน เพราะมี print(python_list) ให้เรียกใช้สำเร็จรูป ซึ่งในรูปแรกจะพิมพ์ [68, 65, ..., 14] แต่ในรูปครุเขียนด้วย C จึงเป็นเช่นนั้น (code เฉลยเป็น C python code ดูได้ใน lecture note)
- print90 () พิมพ์รูป tree ของ array a ของ type T ใดๆ ตั้งแต่ index 0 ถึง i ในรูปหมุนซ้าย 90 องศา หากนศ.ใช้โครงสร้างที่ root อยู่ที่ index 1 ปรับฟังก์ชันให้พิมพ์ตั้งแต่ index 1

กำหนด input array a พร้อมรันฟังก์ชันในข้อ 1 เพื่อทดสอบ a = [68,65,32,24,26,21,19,13,16,14]



2) กำหนด array heap (python list heap) เขียนฟังก์ชัน insert () เพื่อ insert data เข้าไปใน heap ให้วน loop insert data จาก array a เข้าไปใน heap ทีละตัวจนหมด แต่ละครั้งที่ insert ให้พิมพ์ heap เพื่อทดสอบ



- 3) ทำ Heap Sort โดยเขียนฟังก์ชัน deleteMin() เพื่อทำการดึงค่า root ของ heap ในตัวอย่างทำ inplace sort heap ปรับ reheap พร้อมทั้งลดค่า last index ของ heap ลง ทดสอบความถูกต้องโดยให้พิมพ์ array heap ทั้งหมด วน loop deleteMin จนหมดทุกตัวมาใส่ที่ array a พิมพ์ a จะได้ ascending list พิมพ์ h จะได้ decending list ข้างล่างเป็นตัวอย่าง output

```
***deleteMin***
input heap:
13 14 21 19 16 65 24 68 26 32
  24
    21
      65
13      16
      32
    14      26
      19      68
===== heap sort =====
deleteMin =13 FindPlaceFor 32
14 16 21 19 32 65 24 68 26 13
  24
    21
      65
14      32
      13
    16      26
```

```
deleteMin =24 FindPlaceFor 32
26 32 65 68 24 21 19 16 14 13
  19
    65
      21
26      24
      13
    32      14
      68      16
deleteMin =26 FindPlaceFor 68
32 68 65 26 24 21 19 16 14 13
  19
    65
      21
32      24
      13
    68      14
      26      16
```

```
deleteMin =14 FindPlaceFor 26
16 19 21 26 32 65 24 68 14 13
  24
    21
      65
16      32
      13
    19      14
      26      68
deleteMin =16 FindPlaceFor 68
19 26 21 68 32 65 24 16 14 13
  24
    21
      65
19      32
      13
    26      14
      68      16
```

```
deleteMin =32 FindPlaceFor 65
65 68 32 26 24 21 19 16 14 13
  19
    32
      21
65      24
      13
    68      14
      26      16
deleteMin =65 FindPlaceFor 68
68 65 32 26 24 21 19 16 14 13
  19
    32
      21
68      24
      13
    65      14
      26      16
===== Sorting a =====
13 14 16 19 21 24 26 32 65 68
```

```
deleteMin =19 FindPlaceFor 24
21 26 24 68 32 65 19 16 14 13
  19
    24
      65
21      32
      13
    26      14
      68      16
deleteMin =21 FindPlaceFor 65
24 26 65 68 32 21 19 16 14 13
  19
    65
      21
24      32
      13
    26      14
      68      16
```

การทดลอง 2 เขียน Sorting แบบต่างๆ ตาม algorithm ที่เรียน ทดลองโดยกำหนด data เองให้ครอบคลุมทุกกรณี หากทำไม่ได้ ดู Pseudocode คิดตาม algorithm ถึง complexity ของแต่ละ sort ตามที่เรียน ลองใส่ code print จำนวนการ compare data ในแต่ละ pass ในกรณีต่างๆ best case และ worst case

Bubble Sort

Ascending Order จากน้อย -> มาก



Key idea :
Bubble the largest,

- Scan from the left, swap unordered successive elements.

Each scan (pass) :

- Floats the largest (or smallest) up in the right place.
- Remains 1-element-shorter file to process.
- Repeat bubbling.


Then the next largest, etc.

8	3	9	4	5	Original File
3	8	9	4	5	
3	8	9	4	5	
3	8	4	9	5	
3	8	4	5	9	After 1 st pass: the biggest, 9, floats up
3	8	4	5	9	From 1 st pass
3	8	4	5	9	
3	4	8	5	9	
3	4	5	8	9	After 2 nd pass: 2nd biggest, 8, floats up
3	4	5	8	9	From 2 nd pass
					After 3 rd pass: 3 rd biggest 5 floats up

home

Straight Selection Sort (Pushed Down Sort)

Ascending Order เรียงจากน้อย -> มาก



Key idea :

- Scan to select the biggest /smallest, swap with the 1st /last element.


Each scan (pass) :

- Floats the largest (or smallest) up in the right place.
- Remains 1-element-shorter file to process.
- Repeat selection.

6	9	8	5	4	Original File : size n
6	4	8	5	9	After pass 1
6	4	5	8	9	After pass 2
5	4	6	8	9	...
4	5	6	8	9	Sorted File: After Pass ____

home

Insertion Sort



Key idea :

- Scan to insert the card in sorted file in hand.

Each scan (pass) :

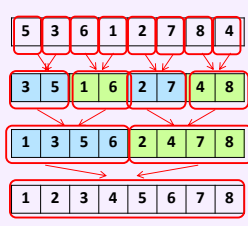
- Compare to insert the the right place to insert.
- If that is not the place, move that data next to its old place to give the room for the inserted card.
- Remains 1-element-shorter file to process.
- Repeat insertion.

8	6	7	5	9
6	8	7	5	9
6	7	8	5	9
5	6	7	8	9
5	6	7	8	9

Ascending Order
จากน้อย -> มาก

home

Merge Sort



Key idea :

- Merge sorted successive pair to be sorted bigger one.
- Merge size 1 successive pair -> sorted size 2
- Merge size 2 successive pair -> sorted size 4
- Merge size 4 successive pair -> sorted size 8

Complexity:
#comparisons = ____
#assignments = ____
Merge = O(____)

home

Quick Sort

Key idea :

- Choose the pivot**
 - 1st element
 - median of 3
 - average
- Partition :**
Pivot partitions files into 2 halves
 - Left half < pivot
 - Right half > pivot
 - Pivot goes to its right place
- Repeat partitioning both left & right half**

0	1	2	3	4	5	6	7	8	9
5	1	4	9	6	3	8	2	7	0

pivot

3	1	4	0	2	5	8	6	7	9
---	---	---	---	---	---	---	---	---	---

Left half < pivot pivot Right half > pivot

pivot's right place

0	1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---	---

Ordered List

home

```

#include<iostream>
#include<fstream>
using std::ofstream;
//using std::cout;
ofstream cout("d:out.txt");
#include<stdio.h>
#include<stdlib.h>
#define size 10
typedef int T;

void print(T a[], int last){
    for(int i = 0; i<=last; i++){
        printf("%d ",a[i]); cout<<a[i]<<" ";
        printf("\n");cout<<"\n";
    }
}

void print90(T *a, int i, int i2, int indent){
    //print root at index i with indent indent
    //root start at index 0, last index = i2
    //left son of index i is at 2*i+1 and right son is at 2*i+2
    if (i<=i2){
        print90(a,2*i+2,i2,indent+1); //recursive print its right son
        //printing root
        for(int ind = 0; ind<indent; ind++){
            printf(" "); cout<<" ";
        }
        printf("%d\n",a[i]); cout<<a[i]<<"\n";
        print90(a,2*i+1,i2,indent+1); //recursive print its left son
    }
}

void insert(T *h, int d, int last){
    //insert heap h with data d
    //heap h start with root at index 0
    //after inserting, index last = last position of the heap
    printf("insert %d\n",d);cout<<"insert " <<d<<"\n";
    int hole = last; // h = hole index
    int fa = (hole-1)/2; //father index
    while (hole && h[fa]>d){
        h[hole] = h[fa];
        hole = fa;
        fa = (hole-1)/2;
    }
    h[hole] = d;
}

T deleteMin(T *h,int last){ //in place delete root
    //last = last index of heap before delete
    T del = h[0]; //delete data
    T input = h[last]; //find place for input
    h[last--] = del; //in place sort, decrease last

    printf("deleteMin = %d FindPlaceFor %d\n",del,input);
    cout<<"deleteMin =" <<del<<" FindPlaceFor " <<input<<"\n";

    int hole = 0; //hole index, 1st at root
    int li = 2*hole+1, ri = 2*hole+2; //left & right son index of hole
    T ls = h[li], rs = h[ri]; //left & right son of hole

    while ((li <= last && input > ls) || (ri <= last && input > rs)){
        if (ri <= last) //both (ri & li) < last
            if (ls < rs){ //fill hole with smaller son & has new hole
                h[hole] = ls;
                hole = li;
            }else {
                h[hole] = rs;
                hole = ri;
            }
    }
}

```

```

        else { //only li <= last
            h[hole] = ls;
            hole = li;
        }
        li = 2*hole+1; //left son index of hole
        ri = 2*hole+2; //right son index of hole
        ls = h[li]; //left son of hole
        rs = h[ri]; //right son of hole
    }
    h[hole] = input;
    return del;
}

int main() {
    T a[size] = {68,65,32,24,26,21,19,13,16,14}; //input data
    T heap[size]; //This heap, root is at index 0
    print(a, 5);
    print90(a,0,size-1,0); //root at index 0

    //loop to insert data in to the heap
    for(int i=0; i<size; i++){
        printf("-----\n"); cout<<"-----\n";
        insert(heap,a[i],i); //insert a[i] to heap h
        //after insert last element of the heap is at index i
        print(heap, i);
        print90(heap,0,i,0);
    }

    printf("****deleteMin***\ninput heap:\n");
    cout<<"****deleteMin***\ninput heap:\n";
    print(heap, size-1);
    print90(heap,0,size-1,0);
    printf("==== heap sort ==== \n");
    cout<<"==== heap sort ==== \n";
    //loop to in place delete data out
    int k = 0;
    for(int last=size-1; last>=1; last--,k++){
        T del = deleteMin(heap,last); //last = last index of heap
        a[k] = del;
        print(heap, size-1);
        print90(heap,0,size-1,0);
    }
    a[k] = heap[0];
    printf("==== Sorting a ==== \n");
    cout<<"==== Sorting a ==== \n";
    print(a,k);
    return 0;
}

```