CS213 : DATA STRUCTURES

Assignment III : Priority Queue and Binary Heaps

Heap เป็นต้นไม้ไบนารีที่เกือบสมบูรณ์ซึ่งเหมาะกับการจัดเก็บข้อมูลในอาร์เรย์ ข้อมูลในทุกโหนดจะต้องเป็นไปตามคุณสมบัติของ heap (คือค่าคีย์ของข้อมูลที่โหนดใดๆ จะต้องมีค่าน้อยกว่าค่าคีย์ของข้อมูลที่โหนด parent เสมอ) ข้อมูลของ heap จะถูกจัดเก็บในอาร์เรย์โดยเรียงลำดับจากซ้ายไปขวาจากระดับบนไประดับล่างตามลำดับ ให้โหนด root เก็บไว้ที่ช่องที่หนึ่งของอาร์เรย์ (ช่องที่ 0 ให้เว้นไว้ไม่ต้องใส่ข้อมูลใด ๆ) แล้วตามด้วยโหนดลูกด้านซ้าย และตามด้วยโหนดลูกด้านขวาไปเรื่อย ๆ เราอ้างถึงโหนดใน heap ด้วยการใช้ index ในอาร์เรย์ ดังนั้นโหนดที่ช่อง 1 คือ root ลูกทางซ้ายของ root อยู่ช่องที่ 2(1) = 2 และลูกทางขวาของ root อยู่ที่ช่อง 2(1)+1 = 3 ดังนั้นโหนดที่ช่องเบอร์ 4 คือลูกทางซ้ายของต้นไม้ย่อยด้านซ้ายของ root นั่นเอง

หลังจากสร้างและทดสอบโครงสร้าง heap แล้ว เราจะประยุกต์ใช้งานฮีปเพื่อทำเป็น PriorityQueue (รายละเอียดด้านล่าง) โดย implement คิวเป็นแบบ max-priority queue ที่ข้อมูลที่มีค่ามากที่สุดออกจากคิวก่อน

ให้นักศึกษาเขียนโปรแกรมภาษา C++ ดังต่อไปนี้

<u>ตอนที่ 1</u> สร้างคลาส Heap ที่มีข้อมูลและเมธอด ดังนี้

```
const int MAX_SIZE 100;

class Heap{
        …
        Heap();
        Heap(int initData[], int len);

        //return true if heap property is satisfied, false if violated
        bool IsHeap();

   private:
        //i is the numbering of node (starting from 1)
        //i is not 1 (as root node has no parent)
        //the numbering of node i's parent is returned

        int parent(int i);     //return i/2

        //return the numbering of the left child of node i
        //if node i has no left child, return -1
        int leftchild (int i);       //return 2*i

        //return the numbering of the right child of node i
        // if node i has no right child, return -1
        int rightchild (int i);      // return 2*i+1

        int data[MAX_SIZE];   //You could store elements from index 1

        int heapLength;   // the actual number of elements in the heap
};
```

ให้เขียนเมธอดเหล่านี้เพิ่มเติมลงใน class (สามารถเขียนเมธอดเพิ่มเติมมากกว่านี้ได้ตามที่คิดว่าจำเป็น):

1. **`heapify (i)`**: สมมติว่าต้นไม้ย่อยทางด้านซ้ายและด้านขวาของโหนด **`i`** เป็นไปตามคุณสมบัติของ Heap แต่โหนด **`i`** เองที่ไม่เป็นไปตามคุณสมบัติของ เมธอดนี้จะแก้ไขต้นไม้ย่อยนี้ที่มี root เป็นโหนด **`i`** ด้านล่างนี้เป็นกรณีที่จำเป็นต้องเรียก **`heapify ()`**
   - เมื่อค่าของโหนด **`i`** ถูกเปลี่ยนเป็นค่าที่น้อยกว่าเดิม ดังนั้น **`heapify (i)`** จะแก้ไข Heap ทั้งต้น

- หากค่าของโหนด `i` ถูกเปลี่ยนเป็นค่าที่มากกว่าเดิม ต้นไม้ย่อยที่มี root เป็น `i` ยังคงเป็น heap แต่ต้นไม้ทั้งต้นอาจไม่เป็นแล้ว (หากค่าใหม่ของโหนด `i` มากกว่าค่าของโหนด parent ของมัน (เราต้องเรียก **`heapify ()`** โดยส่งโหนด parent ของ `i` ไปเพื่อแก้ไข)
- เมธอดนี้จะถูกเรียกใช้ในเมธอด **`build_heap()`** ด้วยโดยที่เราเรียก **`heapify ()`** กับโหนดที่ไม่ใช่โหนดใบไม้ทั้งหมด โดยเริ่มจากโหนดที่ระดับล่างสุด และทำงานไล่ขึ้นมาจนถึง root

2. **`build_heap()`** สมมติว่าข้อมูลอยู่ในอาร์เรย์เรียบร้อยแล้ว แต่อาร์เรย์นี้ยังไม่เป็น heap เมธอดนี้จะทำการเรียงค่าในอาร์เรย์ใหม่เพื่อให้อาร์เรย์ตรงตามคุณสมบัติ heap โดยเมธอดนี้ควรจะถูกเรียกใน constructor ตัวที่สอง **`Heap(int initData[] int len)`**
3. **`Insert(int item):`** ใส่ข้อมูล item ใหม่ใน heap เพื่อให้ต้นไม้ยังคงความเป็น heap ต้องทำตามขั้นตอนดังนี้
   a. เพิ่มความยาวของ heap ขึ้นหนึ่งช่อง
   b. เก็บข้อมูลใหม่ที่ด้านท้ายของ heap สมมติว่าเป็นตำแหน่ง `j`
   c. เริ่มตรวจสอบตั้งแต่โหนด `j` ไปที่ root โดยเปรียบเทียบค่าที่เก็บในโหนด `j` กับโหนด parent ของมัน และอาจจะต้องสลับค่าถ้าจำเป็น
4. **`int GetRoot()`** คืนค่า item ที่เก็บในโหนด root
5. **`int RemoveRoot()`** ลบโหนด root และคืนค่า item ที่โหนด root

ตอนที่ 2 : (Optional) สร้างคลาส Priority Queue (PQ)

สร้าง Priority Queue (PQ) ด้วย Heap class ที่สร้างขึ้น คุณสมบัติของ Priority Queue จะแตกต่างจาก Queue ปกติ(ซึ่งเป็นแบบ FIFO) แต่ Priority Queue ที่จะสร้างนี้ ข้อมูลที่มีค่ามากที่สุดจะถูกลบ ก่อน

```
class PriorityQueue{
   public:
        //insert an element into the priority queue
        void Insert(int x);

        //return element in the queue with the largest key
        int Max();

        //return element in the queue with the largest key, and remove
        //that element from the key
        int ExtractMax():

        //change element x to newx and fix the PQ if new insertion
        //violates its property
        void Modify(int x,int newx):

   private:
        Heap elements;

}
```

ให้เขียน main() เพื่อทดสอบคลาสทั้งตอนที่ 1 และ 2 ดังนี้

- การทดสอบ Heap class
  1. สร้าง heap object และ initialize ด้วยอาร์เรย์ของข้อมูล
  2. ทดสอบว่าเป็น Heap หรือไม่ด้วยเมธอด **`IsHeap()`**
     ถ้าไม่เป็น Heap เรียกเมธอด **`build_heap()`**
  3. Insert ข้อมูลใหม่ไปใน heap และทดสอบว่าหลังการ insert ยังเป็น heap หรือไม่ ถ้าไม่เป็นให้แก้ไข

- การทดสอบ Priority Queue class (ถ้าเลือกทำตอนที่ 2)
    1. สร้าง object ของ PQ โดยเริ่มจากคิวว่าง
    2. ให้ผู้ใช้ใส่ข้อมูลหลายๆ ตัว เข้าไปใน PQ
    3. พิมพ์ค่าสูงสุดด้วย เมธอด **`Max()`** และลบค่าสูงสุดด้วย ExtracMax()
    4. ทดสอบเมธอด Modify()

<u>วิธีการส่ง</u>

- ให้นักศึกษารวมงานทั้งหมดเป็นไฟล์เดียว โดยตั้งชื่อ *เลขทะเบียน.zip* เช่น *6709650123.zip*
- ในไฟล์ .zip ประกอบด้วย *heap.h, heap.cpp* และ *main.cpp* (*pq.h* และ *pq.cpp* ถ้าทำตอนที่ 2)
- ส่งงานทาง courses.cs.tu.ac.th ในกล่องส่งการบ้านของวิชา (ไม่รับงานช่องทางอื่นและไม่รับงานล่าช้า)
- กำหนดส่ง วันศุกร์ที่ 14 พฤศจิกายน 2568 ภายใน 23:59 น.

<u>เกณฑ์การให้คะแนน</u>

ตอนที่ 10 คะแนน (Heap Class – 30%, Methods เพิ่มเติม – 50%, main 20%)
(ตอนที่ 2 - BONUS 3 คะแนน)

คำแนะนำเพิ่มเติม

heapify

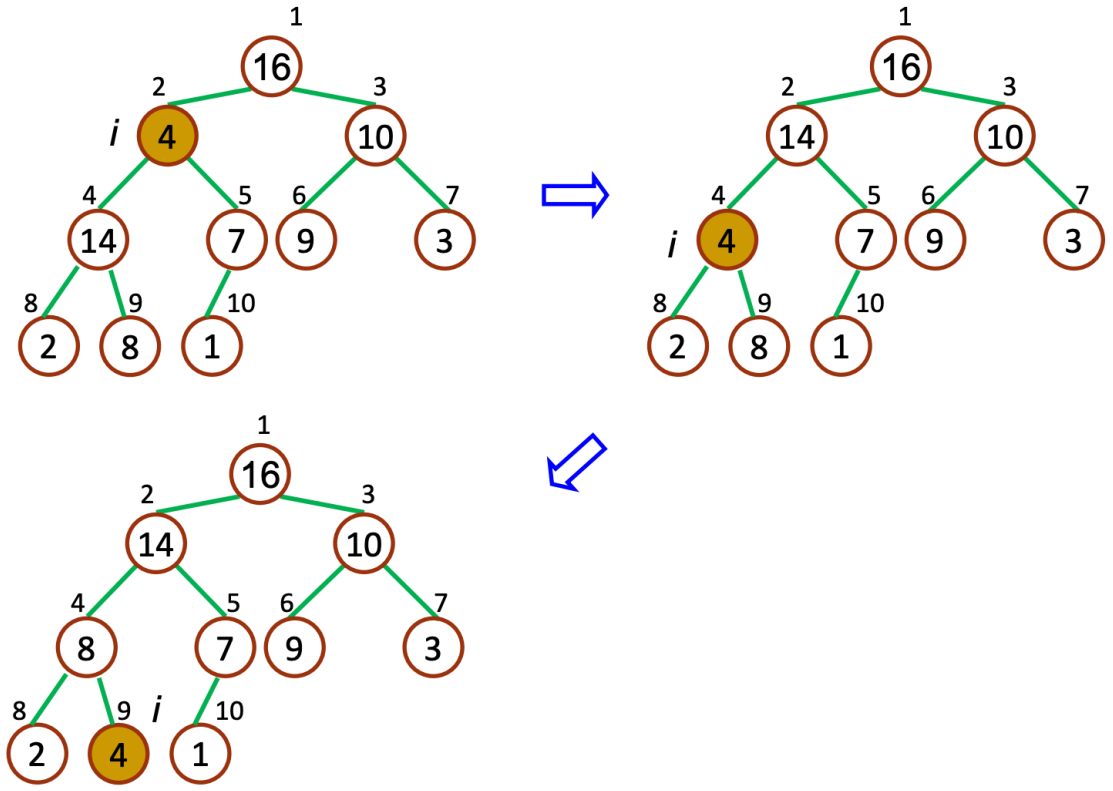
Input: an array A (อาจประกาศเป็นตัวแปรแบบ global) และค่า index i

Output: ต้นไม้ย่อยที่มี root ที่ index i และกลายเป็น max heap

Assume: ต้นไม้ไบนารีที่มี root ที่ตำแหน่ง LEFT(i) และ RIGHT(i) เป็น max-heaps แต่ A[i] อาจจะมีค่าน้อยกว่าโหนดลูกของมัน

Method: ให้ค่าของ A[i] สลับลงไปตำแหน่งที่เหมาะสมด้านล่างใน max-heap

```
heapify(A, i)
        l ← LEFT(i)
        r ← RIGHT(i)
        if l ≤ heap-size[A] and A[l] > A[i]
                then largest ← l
                else largest ← i
        if r ≤ heap-size[A] and a[r] > A[largest]
                then largest ← r
        if largest ≠ i
                then exchange A[i] A[largest]
                        heapify (A, largest)
```



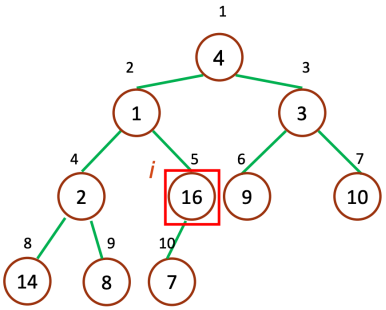
build_heap

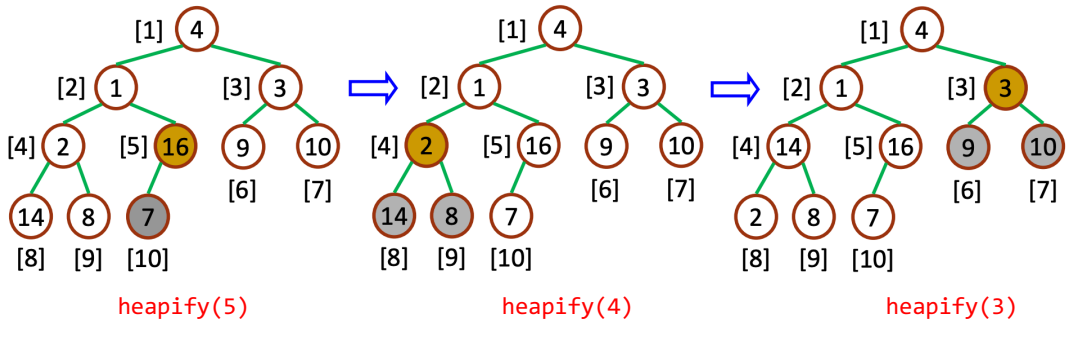
เราสามารถเรียกใช้ **heapify** เพื่อแปลงอาร์เรย์ A=[1..n] เป็น max-heap โดยทำแบบ bottom-up ข้อมูลในอาร์เรย์ตั้งแต่ช่องที่ A[(⌊n/2⌋+1)...n] จะเป็นโหนดใบไม้ทั้งหมด กระบวนการของ **build_heap** จะทำกับโหนดที่เหลือที่ไม่ใช่โหนดใบไม้โดยเรียก heapify ทีละโหนด เริ่มจากโหนดที่อยู่ก่อนโหนดใบไม้ ดังภาพ

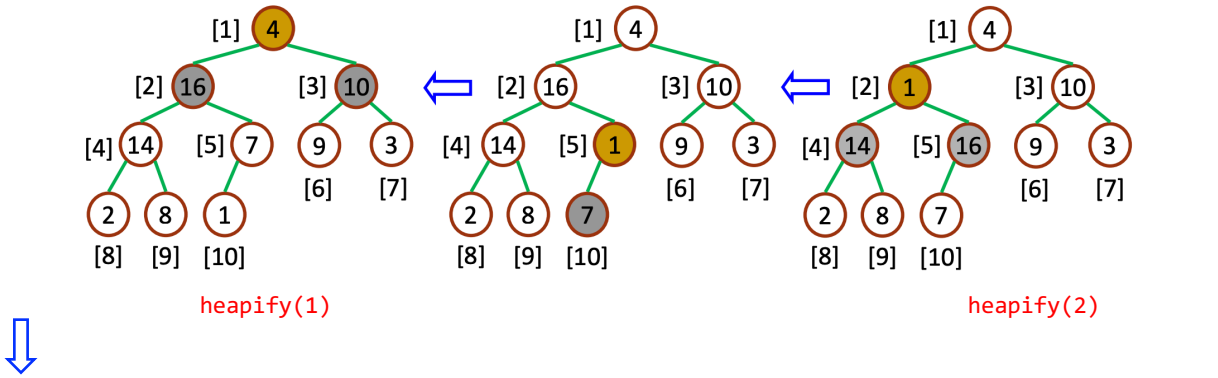
```
build_heap ()
        heap-size[A] ← length[A]
        for i ← ⌊length[A]/2⌋ downto 1
                do heapify(i)
```

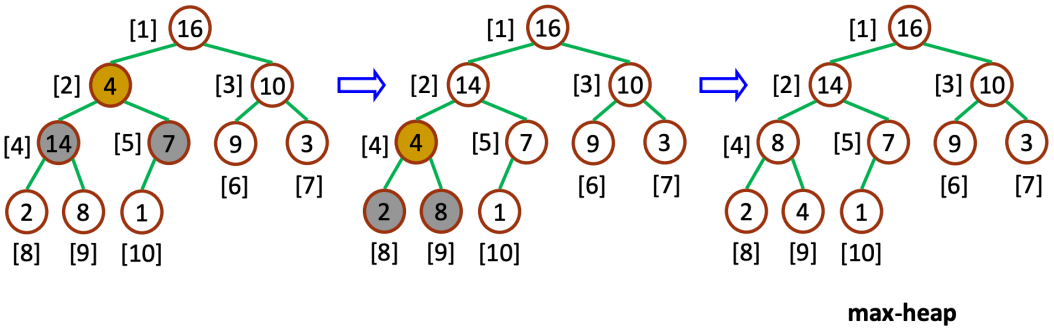
Example:

	1	2	3	4	5	6	7	8	9	10
A	4	1	3	2	16	9	10	14	8	7









max-heap