Week 3 - Priority Queue

blue?

ADT

key, value pair

- insert(k, v) : Insert item with key k, value v
- remove_min() : remove & return element with smallest key
- min() : return element with smallest key
- size() : how many items stored
- is_empty() :

priority can be either minimum / maximum *also possible!!*

Application ex) stock exchange matching program

Buy			Sell		
Shares	Price	Time	Shares	Price	Time
1000	$4.05	20s	500	$4.06	13s
100	$4.05	6s	2000	$4.07	40s
2100	$4.03	20s	3000	$4.10	54s

```
while True:
    bid ← buy_orders.max()
    ask ← sell_orders.min()
    if by.price ≥ sell.price :
        carry out trade (bid, sell)
            buy_orders.remove_max()
                sell_orders.remove_min()
```

Minimum, Maximum

priority determined by
1. price
2. time

Implementation

Unordered Linked List

Insert : O(1) append anywhere

remove : O(n) find item, remove

1 - 5 - 4 - 3 - 2

Ordered Linked List

Insert : O(n)

remove : O(1)

1 - 2 - 3 - 4 - 5

Priority Queue Sorting

1. Iteratively insert keys into an empty priority queue.
2. Iteratively remove_min() to get the keys in sorted order.

```
Input: A
pq ← new priority queue
n ← len(A)
for i in [0:n] do              ] sorted: O(n^2)  unsort: O(n)
    pq.enqueue(A[i])
for i in [0:n] do              ] sorted: O(n)  unsorted: O(n^2)
    A[i] = pq.remove_min()
```

Time Complexity: $O(n^2)$

variant: selection sort

i <-> 미니멈

- sorts in place
- essentially is pq-sort using unsorted sequence

code

```
for i in [0:len(A)] do
    s = i          ← index of minimum element between i:n
    for j in [i:len(A)] do
        if A[j] < A[s] do
            s = j  ← index of element with smaller value
    A[i], A[s] = A[s], A[i]      A[i] <-> A[s]
```

3 1 2 4
1 3 2 4
1 2 3 4

2 3 2 4 1

variant: insertion sort

item 앞으로 옮기도, 앞쪽에서만 왔다갔다 하나, 뒤에 elem 차례대로 visit 하게 됨

3 2 4 1
3 2 4 1
2 3 4 1
2 3 4 1
1 2 3 4

1 2 3 4 1

```
for i in [1:len(A)] do
    x ← A[i]
    j ← i
    while j > 0 and x < A[j-1] do
        A[j] ← A[j-1]      오른쪽으로 밀기
        j -= 1
    A[j] ← x
```

skip 되면 똑같은 걸 assign

5.2

HEAP

HEAP Data structure

↳ binary tree that stores (key, value) items at its nodes.

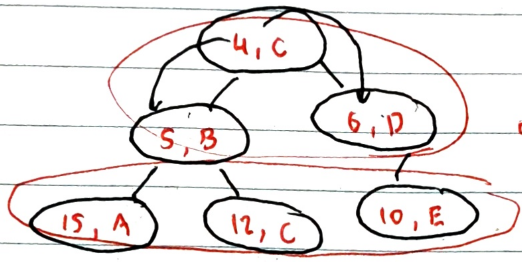
- Heap order: for every node m, ($m \neq$ root)   *Siblings don't matter*

  $m.key \geq m.parent.key$

- Complete binary tree: every level (except the last one) is full
  $i < h$

  Remaining nodes take leftmost positions of level h.

(ex)



other levels : has to be full

last level : nodes are leftmost filled

• child $\geq$ parent

Minimum key of Heap: Always the ROOT

Height of heap: h

$n \geq 2^h \quad \therefore \log_2 n \geq h$ ← 마지막 level에 node 하나면 같음 / 커봤자 1이상 차이 안남!

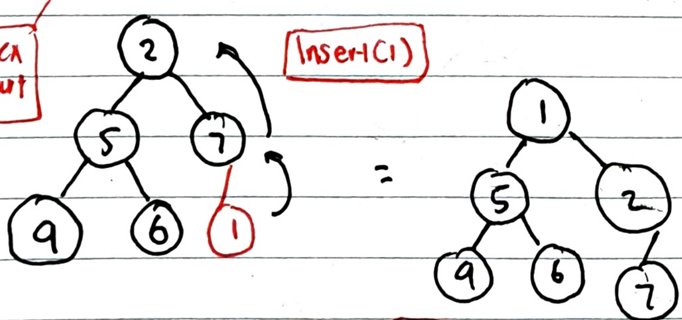
because $n = 1 + 2 + 4 \cdots 2^{h-1} + (1+) \geq 2^h$

Insertion (Upheap algorithm) runs in O(log n)

↳ Find location for new node.

↳ If node has smaller key than its parent, swap location with parent

Complex part



Finding the position ★ [# of children 알고??]

시작점의 left child null 이면 자신이 right

시작점이 right

Keep going up the root until
- left child → leftmost (right subtree) [illegible]의 left
- root (계속 왼쪽 child만 있었으니 NEW LEVEL)

(from last child)

Keep track of !!!

Removing of minimum (root)

Swap last node ↔ root
(w) (r)

delete r that is in w's place before.

restore HEAP ~~Property~~ order (start from root). DOWN-HEAP Algorithm

keep track of new last node