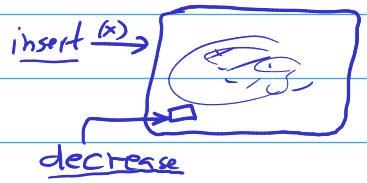
Heap sort & priority queue

"Heap" (min-heap, max-heap) (priority queue)



Maintains: set S of items

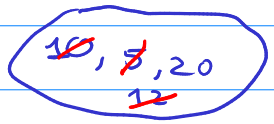
- T_I • insert(x): $S \leftarrow S \cup \{x\}$
- T_E • extractMin: $m \leftarrow \min_{x \in S} x$ *
  $S \leftarrow S - \{m\}$
  return m
- readMin: return $m \leftarrow \min_{x \in S} x$
- decrese(x, elt, x'): $S \leftarrow S - \{x\} \cup \{x'\}$
  $x' \leq x$

min heap

Input: $A[1, 2, \ldots, n]$

```
Build Heap (A)                    O(n · T_I)
for i ← 1, ..., n :               O(n · T_E)
    x ← ExtractMin()              O(T_E)
    output x
```

ex



insert(10)
insert(5)
insert(20)
extractMin → 5
extractMin → 10
insert(12)
extractMin → 12

BuildHeap(A): (direct)

```
for i ← 1, ..., n                 O(n · T_I)
    Insert(A[i])
```

H. Insert(A[i])

Running time: $O(n \cdot (T_I + T_E))$

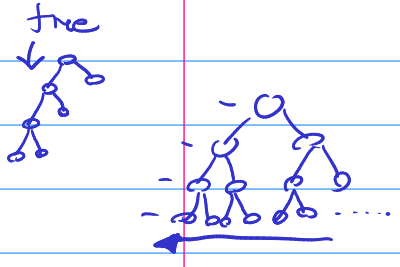
n = ขนาดของ heap

■ Heap Implementation of priority queue: $T_I = O(\log n)$, $T_E = O(\log n)$
readMin $O(1)$, decreasekey $= O(\log n)$

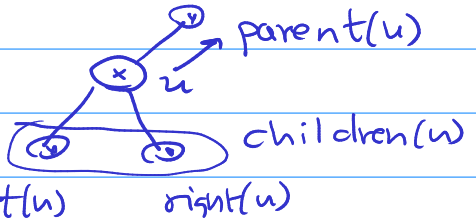
→ HeapSort $O(n \log n)$

→ BuildHeap — $O(n)$ เร็วกว่าการ insert.

full binary tree



Heap = (nearly) complete binary tree + heap criteria (heap property)



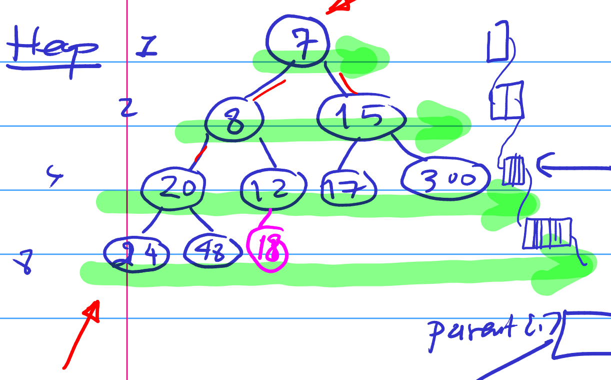
Heap Property สำหรับทุก ๆ node u ที่ไม่ใช่ root

$key[u] \geq key[parent(u)]$

น้อยสุด (ถ้า satisfy Heap property)

key[root] น้อยสุด — readMin

Heap

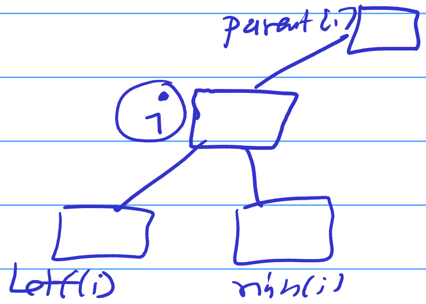


Data structure เก็บใน array ได้

1 2 3 4 5 6

7, 8, 15, 20, 12, 17, 300, 24, 48, 18

0 1 2 3 4 5

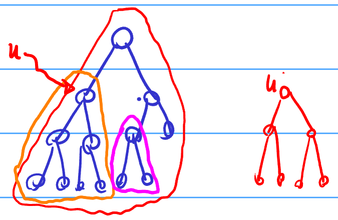


0-based array: parent $\lfloor (i-1)/2 \rfloor$

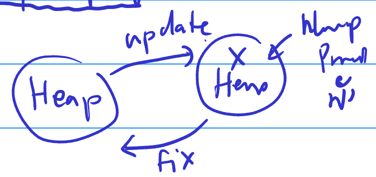
Left: $2i+1$ right: $2i+2$

1-based array: parent $\lfloor i/2 \rfloor$

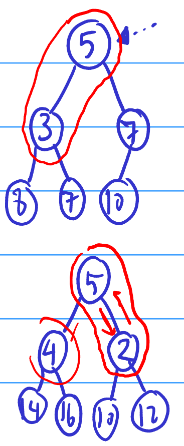
Left: $2i$ right: $2i+1$



Maintain Heap Property



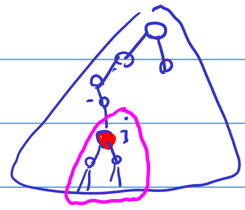
Percolate down

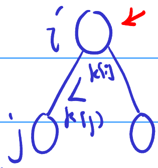


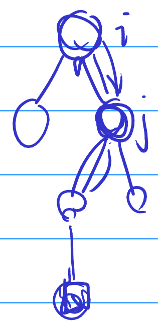
Heapify

Heapify(i)

Precondition: Heap property เป็นจริงทั่ว Heap แต่(อาจ)ไม่จริง ที่ i เท่านั้น กับ child j ของ i ($j = Left(i)$ หรือ $j = Right(i)$) ที่ $Key[j] < Key[i]$







Correctness

```
Heapify(i):
  while True:
    if IsLeaf(i): return                      ← Term ✓
    if hasRightLeaf(i)
      if Key[Left(i)] < Key[Right(i)]
        j ← Left(i)
      else j ← Right(i)
    else
      j ← Left(i)
    if Key[j] < Key[i]
      Swap(Key[i], Key[j])                    ← Heap Property จริงที่ i
      i ← j                                   ← อาจไม่จริงที่ j
    else
      return .                                ← Term ✓
```

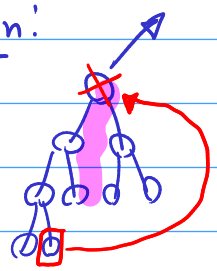
i จะเพิ่มขึ้น

แต่ละรอบ while — O(1)

Running Time: ทำไม่เกิน h รอบ เมื่อ h คือความสูงของ Heap

→ $h = O(\log n)$

✓ **ExtractMin:** ใช้ s แทน size ของ heap. (1-based)



Heap Sort

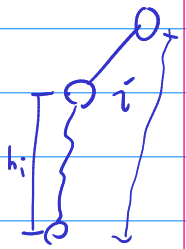
ExtractMin:

```
m ← ReadMin
Key[1] ← Key[s]
S ← S - 1
Heapify(1)
return m
```

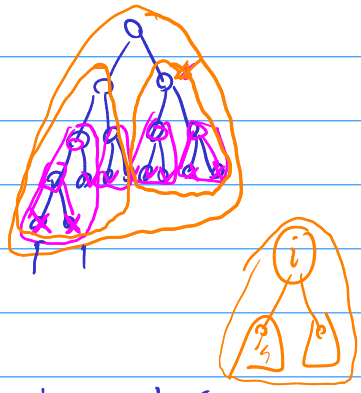
$O(\log n)$

✓ **BuildHeap:** ใช้ ข้อมูล ของ array A สร้าง Heap

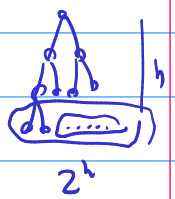
Init: เริ่ม จาก "Heap" ที่มี ข้อมูล ครบทุกตัว แต่ Heap Property อาจจะไม่




```
for i ← 1,2,...,n        // copy ข้อมูล
    Key[i] ← A[i]
for i ← n, n-1, n-2, ..., 1
    Heapify(i)
```

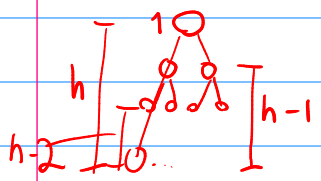


Correctness: Claim: หลัง Heapify(i) SubHeap ที่ root ที่ i จะสอดคล้องกับ Heap Property (by induction)



Running Time: Assume ว่า Heap เป็น complete binary tree ที่มี h level มี node เท่ากับ $1 + 2^1 + 2^2 + 2^3 + \cdots + 2^{h-1} = n = (2^h - 1)$

$h \cdot 1 + (h-1) \cdot 2 + (h-2) \cdot 4 \cdots$



ผลรวมของทั้งหมด

$$\sum_{j=1}^{h} (2^{j-1})(h+1-j)$$

Geometric Series $a^0 + a^1 + a^2 + \cdots + a^k = \frac{a^{k+1} - 1}{a - 1}$

$$S = a^0 + a^1 + \cdots + a^k \quad -(1)$$

$$aS = a^1 + a^2 + \cdots + a^{k+1} \quad -(2)$$

$$(a-1)S = a^{k+1} - a^0 \quad (2)-(1)$$

$$S = \frac{a^{k+1} - 1}{a - 1}$$

If $a < 1$, $a^0 + a^1 + a^2 + a^3 + \cdots = \frac{a^\infty - 1}{a - 1} = \left(\frac{1}{1-a}\right)$

$1 + \frac{1}{2} + \frac{1}{4} + \cdots = \frac{1}{1-\frac{1}{2}} = 2$

$$\sum_{j=1}^{h} (2^{j-1})(h+1-j) = 2^h \left[\sum_{j=1}^{h} (2^{j-1-h})(h+1-j)\right]$$

$$= 2^h \left(\sum_{j=1}^{h} (h+1-j)\frac{1}{2^{h+1-j}}\right)$$

$$= 2^h \left[\frac{h}{2^h} + \frac{(h-1)}{2^{h-1}} + \cdots + \frac{1}{2^1}\right]$$

$$= 2^h \left[\sum_{i=1}^{h} i \cdot 2^{-i}\right] \quad O(1) \quad = O(n)$$

$\frac{1}{2} + \frac{2}{4} + \frac{3}{8} + \frac{4}{16} + \cdots$

$S = \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \cdots$

$S/2 = \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \cdots$

$S/4 = \frac{1}{8} + \frac{1}{16} + \cdots$

$S/8 = \frac{1}{16} + \cdots$

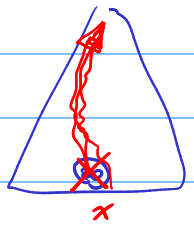
$\leq S(1 + \frac{1}{2} + \cdots +) \leq 2S$

Build Heap: running time $O(n)$

Insert / decrease Key $\quad S$ = size of Heap.

Insert(x):

```
S ← S + 1
Key[s] ← ∞        // Heap Property ok's
decrease Key (s, ∞, x)
```



percolate up

Decrease Key (i, x, x') $\quad (\text{Key}[i] = x)$

$x' < x$

Running Time: $O(\log n)$

Implement Build Heap using Insert:

Running Time: $\Theta(n \log n)$