~~30~~ COP3502 4/7/26

Missing: Mia, Gabriel O,

Today - Speed Run

1) Created FP turn in Wed 11:59pm
   - can turn in late (I need all of them!!!)

Topic #1: Binary Heaps

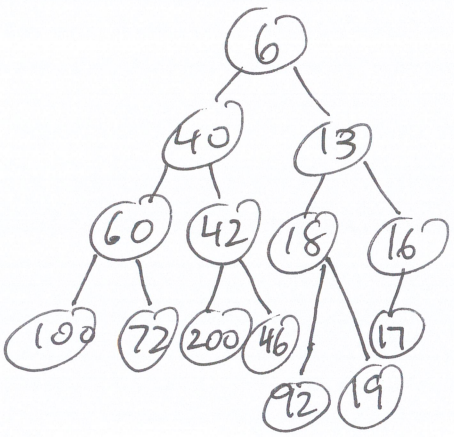
- PriorityQueue in APIs

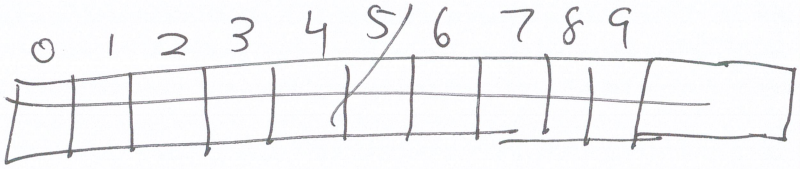
Behavior

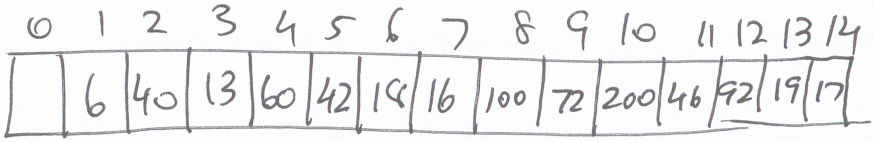
Insert Item	O(lg n)
Delete minimum	O(lg n)

2 properties of a binary heap

1) complete binary tree
2) each node adheres to the binary heap invariant:
   for each node all value in its subtree below it are greater than it

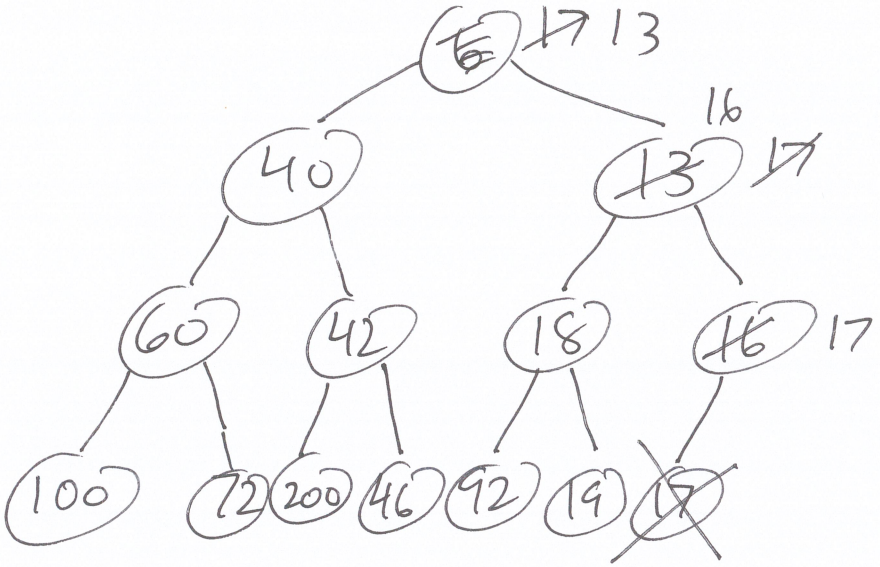






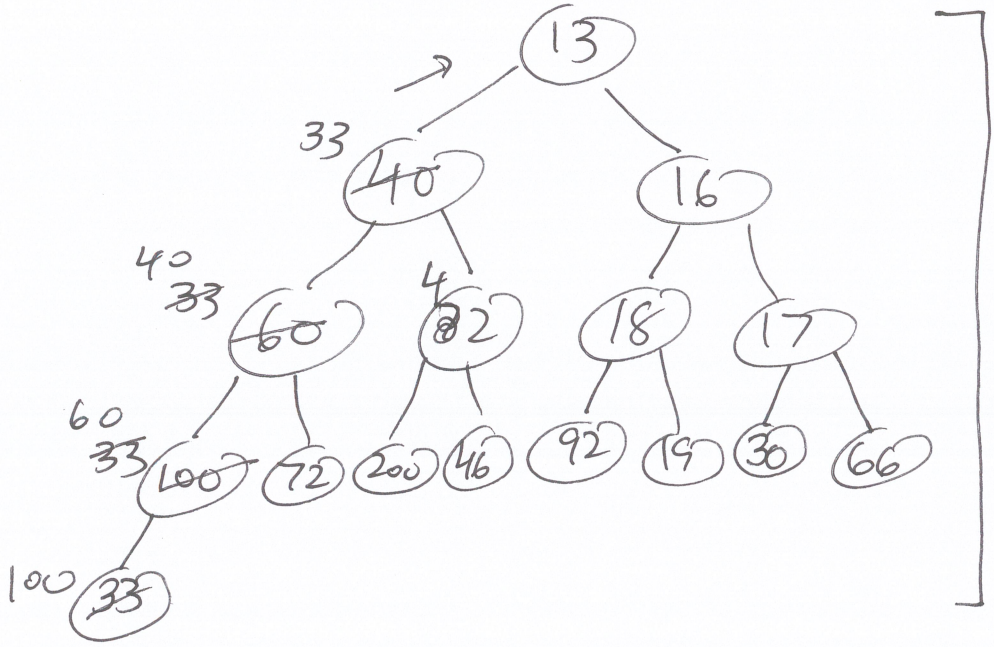
me: i left: 2i right: 2i+1
parent: i/2

6



Delete Min

1. Save root val arr[1]
2. Move last value into root
3. Percolate Down on root.
4. Return deleted value



Insert

1. Put in 1st empty slot
2. Percolate Up

If there is a height of h

Fewest nodes the tree could have is 2^h.

$2^{h+1} > n \geq 2^h$

$\log_2 n \geq h$

$\log_2 2^{h+1} > \log_2 n$

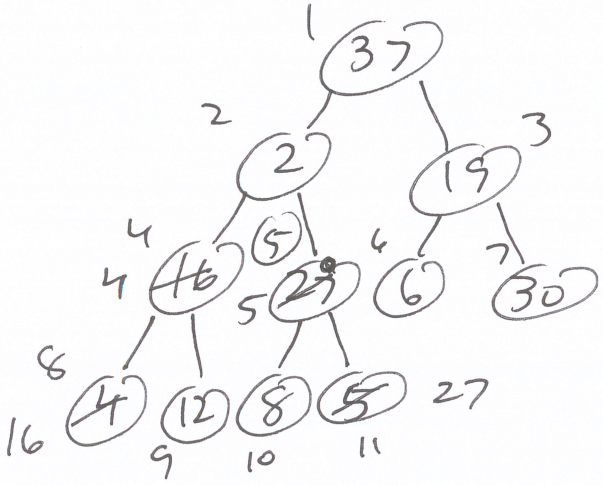
$h+1 > \log_2 n$

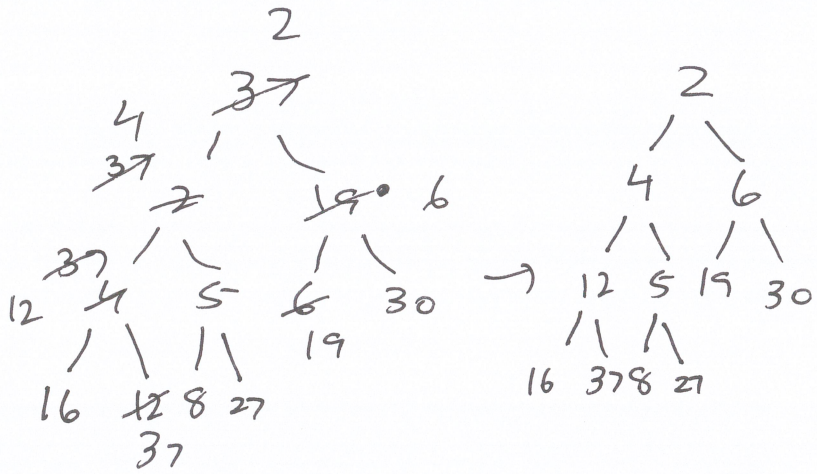
$h > \log_2 n - 1$

$h = O(\lg n)$

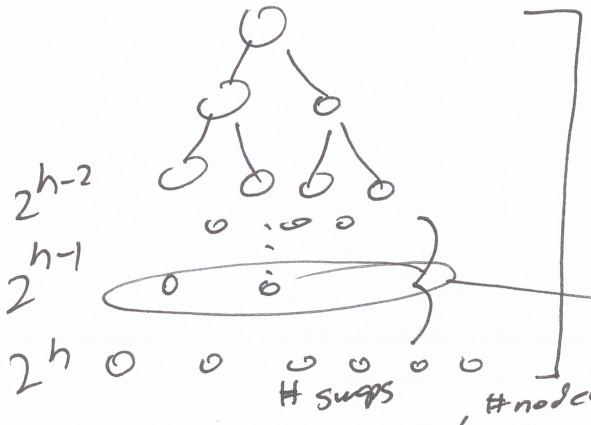
both functions run in $O(h)$ time so $O(\lg n)$.

CLRS Algorithms Books





Heapify Run-Time



height is h

$n = \boxed{2^{h+1} - 1}$

→ run ~~Heapify~~ percolate down on these

$$S = \underline{1 \times 2^{h-1}} + 2 \times 2^{h-2} + 3 \times 2^{h-3} + 4 \times 2^{h-4} + \cdots (h) \times 2^0$$

$$-\frac{S}{2} = \quad 1 \times 2^{h-2} + 2 \times 2^{h-3} + 3 \times 2^{h-4} + \cdots (h-1) \times 2^0 + \frac{h}{2}$$

$$\frac{S}{2} = 2^{h-1} + 2^{h-2} + \cdots + 2^0 - \frac{h}{2}$$

$$S = 2\left(2^0 + 2^1 + 2^2 + \cdots + 2^{h-1}\right) - h$$

$$S = 2(2^h - 1) - h = \boxed{2^{h+1} - 2 - h} = O(n)$$

Heapsort #1

```
int * arr;
// Fill array size n
struct heap* myheap;
init(myheap);
for (int i=0; i<n; i++)
    insert(myheap, arr[i]);
for (int i=0; i<n; i++)
    arr[i] = delmin(myheap);
```

$\lg 1 + \lg 2 + \lg 3 + \lg 4 + \cdots + \lg n = \lg n!$

Stirling's top: $n! \sim \left(\frac{n}{e}\right)^n$

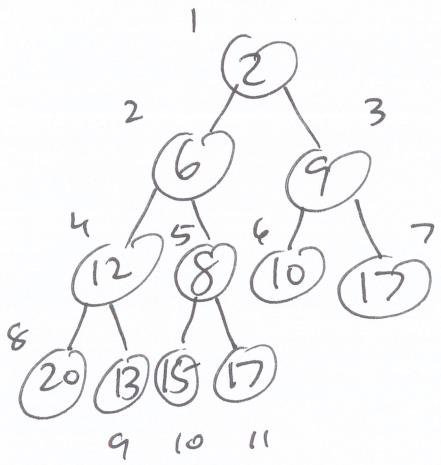
$O(n \lg n)$

Heapsort #2

1) copy unsorted values into array

2) run heapify on array

3) Call delete min n times.



heapify · code

```
for (i = n/2; i >= 1; i--)
    percolateDown(heap, i)
```

Run time = $O(n)$

Hash Tables

1) Add
2) Search
3) not actually sorted
GOAL: achieve sub O(lgn) run-times
4) delete (not easily supported for some strategies)

4 strategies to implement

Internal Storage

array of a fixed size n (0 to n-1)
must have access to a hash function that takes as input any arbitrary item that could be stored in the table and returns an int in between 0 and n-1.

```
int f(char* s);
```

Properties of good hash functions

1) probability of each output is roughly equally likely
2) Small changes in the input produce unpredictable changes in the output.

2 MAIN PRACTICAL USES (usually $n = 2^{512}$ or something)
1) password storage / crypto
2) implementing hash tables

Issue: Collisions!

$f(x) = f(y)$ but we need to store both x, y.

4 ways to handle

1. Do nothing, overwrite old value!

2. Linear Probing
   - if index i is full, goto index $i+1, i+2, \ldots, n-1, 0, 1, 2, \ldots$
   - Search: must look until 1st empty slot
   - issue - clustering

3. Quadratic Probing
   - Next time

4. Linear Chaining Hashing
   each index is a linked list

Trace

f(cat) = 5

f(dog) = 7

f(elephant) = 2

f(giraffe) = 4

f(koala) = 5