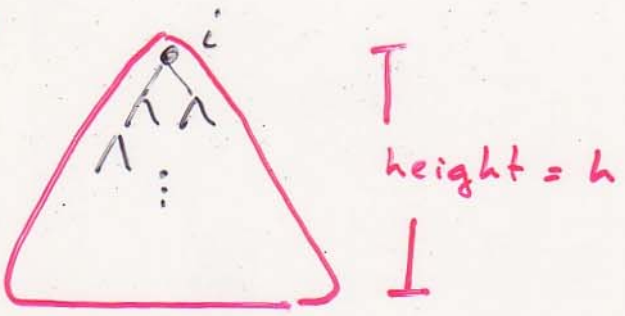
# CS 223 - Lec 3

Running Time for

MAX-HEAPIFY

BUILD-MAX-HEAP




MAX-HEAPIFY

time = $O(h)$

BUILD-MAX-HEAP ( heap size = $n$ )

time = $O(n \lg n)$

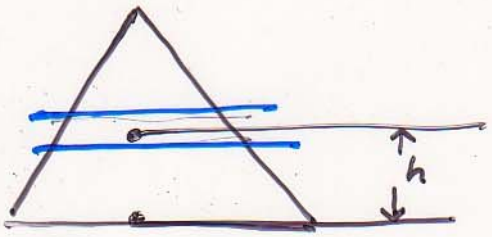
better = $O(n)$

Observe

at height $h$, there are

at most

$$\lceil n/2^{h+1} \rceil \quad \text{nodes}$$




Running time of

BUILD-MAX-HEAP

$$\leq \sum_{h=0}^{\lfloor \lg n \rfloor} \underbrace{\lceil n/2^{h+1} \rceil}_{\substack{\text{this many} \\ \text{nodes of} \\ \text{height } h}} \cdot \underbrace{O(h)}_{\substack{\text{the time} \\ \text{to} \\ \text{call} \\ \text{MAX-HEAPIFY}}}$$

**Fact**

$$\sum_{k=0}^{\infty} x^k = \frac{1}{1-x}, \quad |x| < 1$$

differentiate both sides ...

$$\sum_{k=0}^{\infty} k x^{k-1} = \frac{1}{(1-x)^2}$$

multiply by $x$ ...

$$\sum_{k=0}^{\infty} k x^k = \frac{x}{(1-x)^2}, \quad |x| < 1$$

---

... continuing

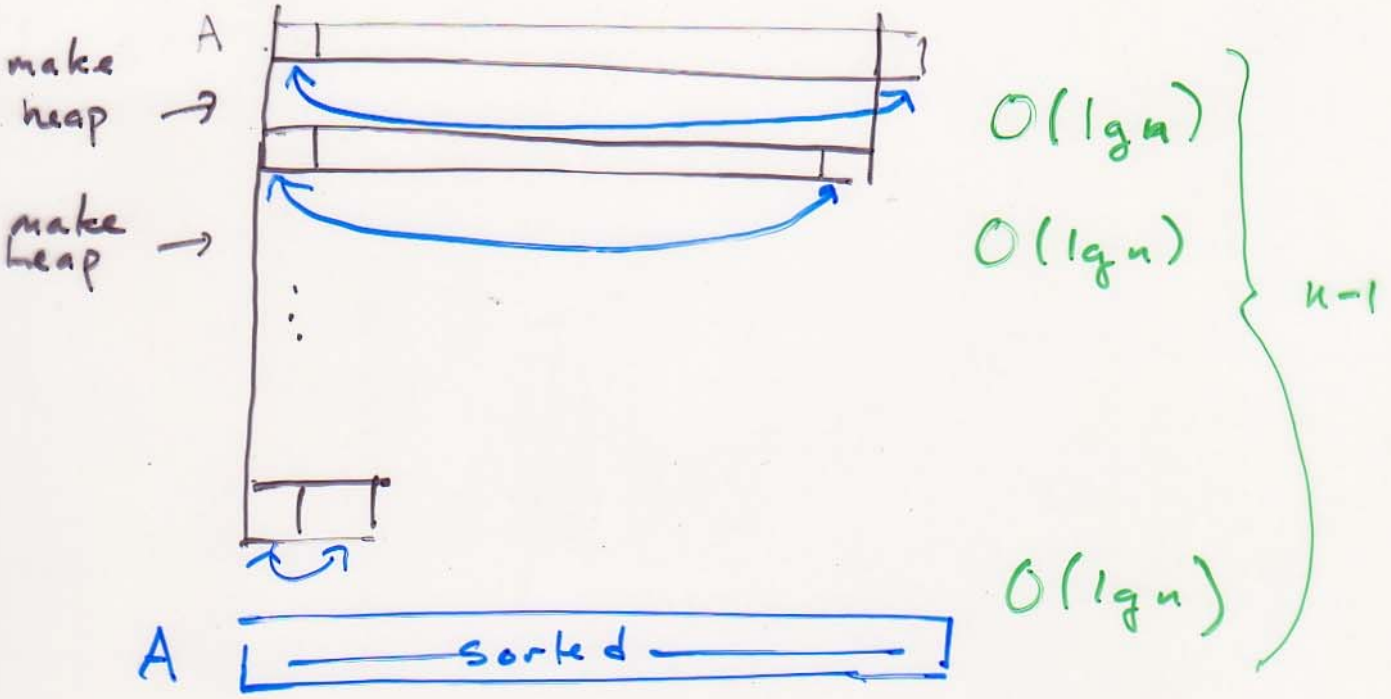
$$= O\left(n \cdot \sum_{h=0}^{\text{Ligd}} \frac{h}{2^h}\right) = O(n)$$

$\leq 2$

plug $x = \frac{1}{2}$

$$\sum_{k=0}^{\infty} \frac{k}{2^k} = \frac{\frac{1}{2}}{(1/2)^2} = 2$$

# Heapsort




running time : $O(n \lg n)$

- This algorithm 'sorts in place' (good thing)

Worst-case time for quicksort $\Omega(n^2)$

# Priority Queue

$S = \{$ set of elements
elements have key value $\}$

## Operations

INSERT(S, x)

inserts element x into S

MAXIMUM(S)

return the element in S with the largest key

EXTRACT-MAX(S)

removes and returns the greatest element in S

INCREASE-KEY(S, x, k)

increases x's key value to k

Idea store S in a heap (max-heap)

```
HEAP-MAXIMUM(A)
    return A[1]
```

time: O(1)

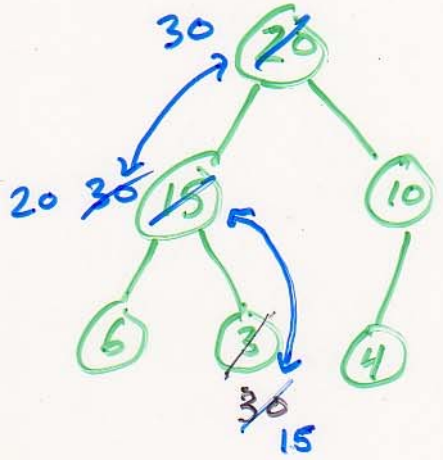
```
HEAP-EXTRACT-MAX(A)
1. if heap-size[A] < 1
2.     then error "heap underflow"
3. max ← A[1]
4. A[1] ← A[heap-size[A]]
5. heap-size[A] --
6. MAX-HEAPIFY(A, 1)
7. return max
```

time: O(lg n)

HEAP-INCREASE-KEY( A, i, key)

1. if key < A[i]
2. then error "too small a key"
3. A[i] = key
4. while i > 1 and A[parent(i)] < A[i]
5. do { exchange A[i] ↔ A[parent(i)]
6. i ← parent(i)
   }

time: $O(\lg n)$



MAX-HEAP-INSERT (A, key)

1. heap-size[A] ++
2. A[heap-size] ← $-\infty$
3. HEAP-INCREASE-KEY
   (A, heap-size[A], key)

time : $O(\lg n)$

Application

- event simulation