Analysis of Algorithms

V. Adamchik

Lecture 3

CSCI 570

University of Southern California

Heaps

Reading: chapter 3

Amortized Analysis

In a sequence of operations the worst case does not necessarily occur in each operation …

Therefore, a traditional worst-case per operation analysis can give overly *pessimistic* bound.

When same operation takes different times, how can we accurately calculate the runtime complexity?

The Aggregate Method

The aggregate method computes the upper bound $T(n)$ on the total cost of n operations.

The amortized cost of an operation is given by $\frac{T(n)}{n}$

In this method each operation will get the same amortized cost, even if there are several types of operations in the sequence.

$T(n) = O(1)$ $T(n) \geq O(n)$

Review Questions ?

2. (T/F) Amortized analysis is used to determine the average runtime complexity of an algorithm.
3. (T/F) Compared to the worst-case analysis, amortized analysis provides a more accurate upper bound on the performance of an algorithm.
4. (T/F) The total amortized cost of a sequence of n operations gives a lower bound on the total actual cost of the sequence.
5. (T/F) Amortized constant time for a dynamic array is still guaranteed if we increase the array size by 5%.
6. (T/F) If an operation takes $O(1)$ expected time, then it takes $O(1)$ amortized time.
7. Suppose you have a data structure such that a sequence of n operations has an amortized cost of $O(n \log n)$. What could be the highest actual time of a single operation?

per insert

$n \cdot 2$ $n \cdot 1.6$ $= 60\%$

$n+2$

seq.: $1, 1, 1, \ldots, 1, n^2$. Total cost $= (n-1) + n^2 = O(n^2)$

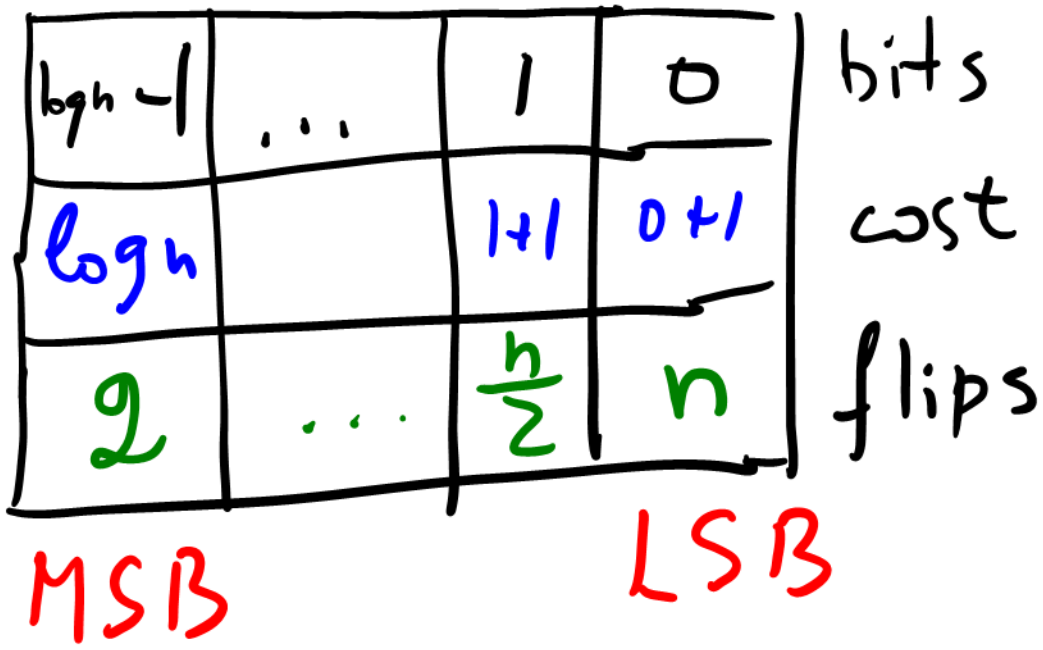
$n-1$

answer: $O(n \log n)$

AC(per increment) = O(n)/n = O(1)

Review: Exercise 2

2. We are incrementing a binary counter, where flipping the i-th bit costs $i + 1$. Flipping the lowest-order bit costs $0 + 1 = 1$, the next bit costs $1 + 1 = 2$, the next bit costs $2 + 1 = 3$, and so on. What is the amortized cost per operation for a sequence of n increments, starting from zero?



$$\text{Cost} = n \cdot 1 + \frac{n}{2} \cdot 2 + \frac{n}{4} \cdot 3 + \ldots + 2 \cdot \log n =$$

$$= n \sum_{K=0}^{\log n - 1} \frac{(K+1)}{2^K} \leq$$

$$\leq n \sum_{K=0}^{\infty} \frac{K+1}{2^K} = O(n)$$

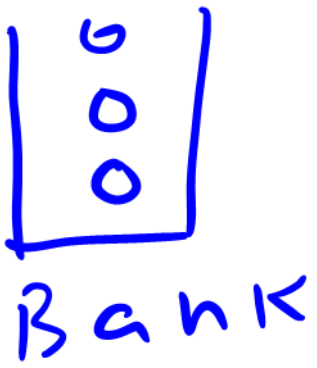
const

The Accounting Method

The accounting method (or the banker's method) computes the individual cost of each operation.

We assign different charges to each operation; some operations may charge more or less than they actually cost.

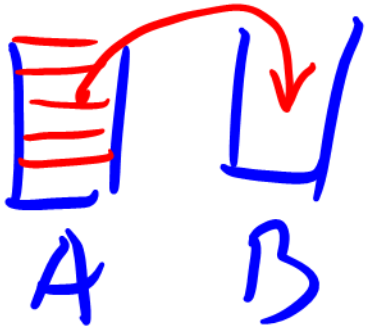
The amount we charge an operation is called its amortized cost. O(1) token



Discussion Problem

You have a stack data type, and you need to implement a FIFO queue. The stack has the usual POP and PUSH operations, and the cost of each operation is 1. The FIFO has two operations: ENQUEUE and DEQUEUE.

We can implement a FIFO queue using two stacks. What is the amortized cost of ENQUEUE and DEQUEUE operations?



O(3) ~~O(1)~~ AC enqueue

O(1) ~~O(3)~~ AC dequeue

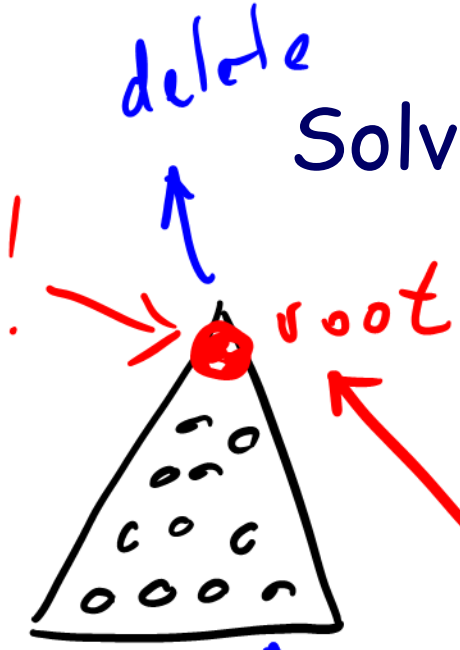
B.pop

Queue

A.push

Bank ≥ 0 !

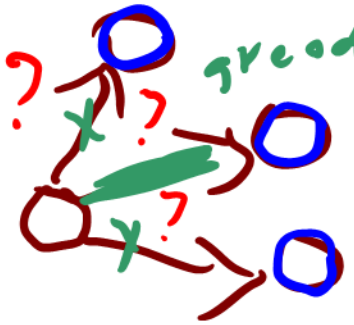
Heap and Priority Queue for Solving Optimization Problems



1. sort O(n log n)
2. findMin O(n)
3. heap O(log n)

Heap — implementation
PQ — interface, API

insert
delete



the best choice

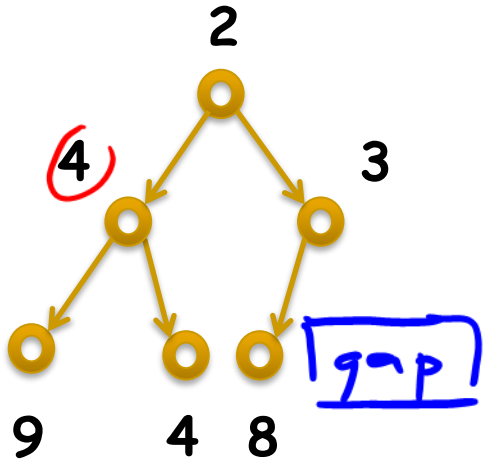
BST

$L < P < R$

max $\Rightarrow$ $P \geq L$ & $P \geq R$

Binary min-Heap

A binary heap is a complete binary tree which satisfies the heap ordering property.



1. Structure Property

2. Ordering Property

$P \leq L$ & $P \leq R$

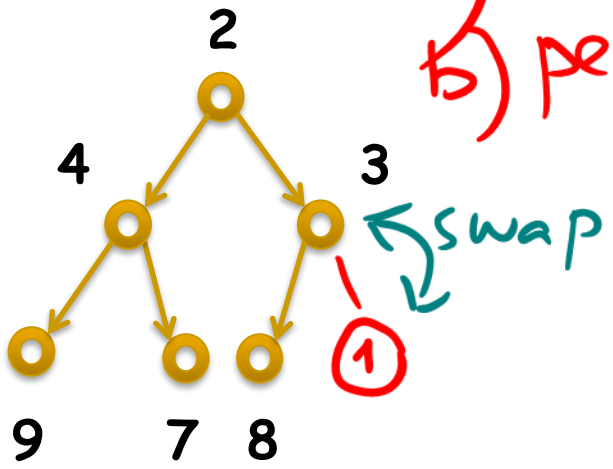
0	1	2	3	4	5	6	7
X	2	4	3	9	4	8	gap

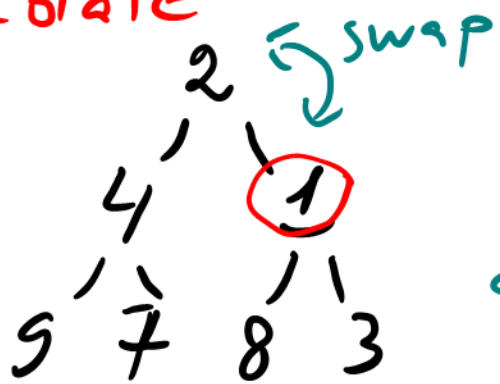
Consider k-th element of the array,

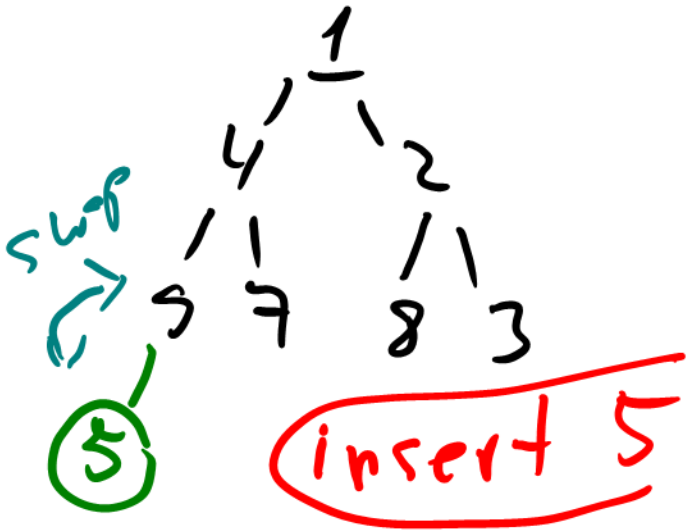
- its left child is located at 2*k index
- its right child is located at 2*k+1 index
- its parent is located at k/2 index $\rightarrow$ $5/2 = 2$

insert (tree reps)

a) into a gap
b) percolate up (bubble up)



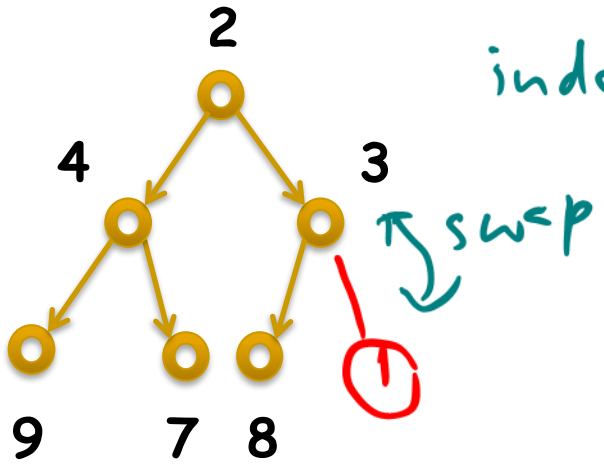




Runtime (worst-case) =
= # of swaps = height of the heap
= $O(\log n)$, here n is the input size

implementation

insert (array reps)



① - append

index	0	1	2	3	4	5	6	7
	X	2	4	3	9	7	8	1
	X	2	4	1	9	7	8	3
	X	1	4	2	9	7	8	3

single for-loop

Discussion Problem 1

Prove it by construction.

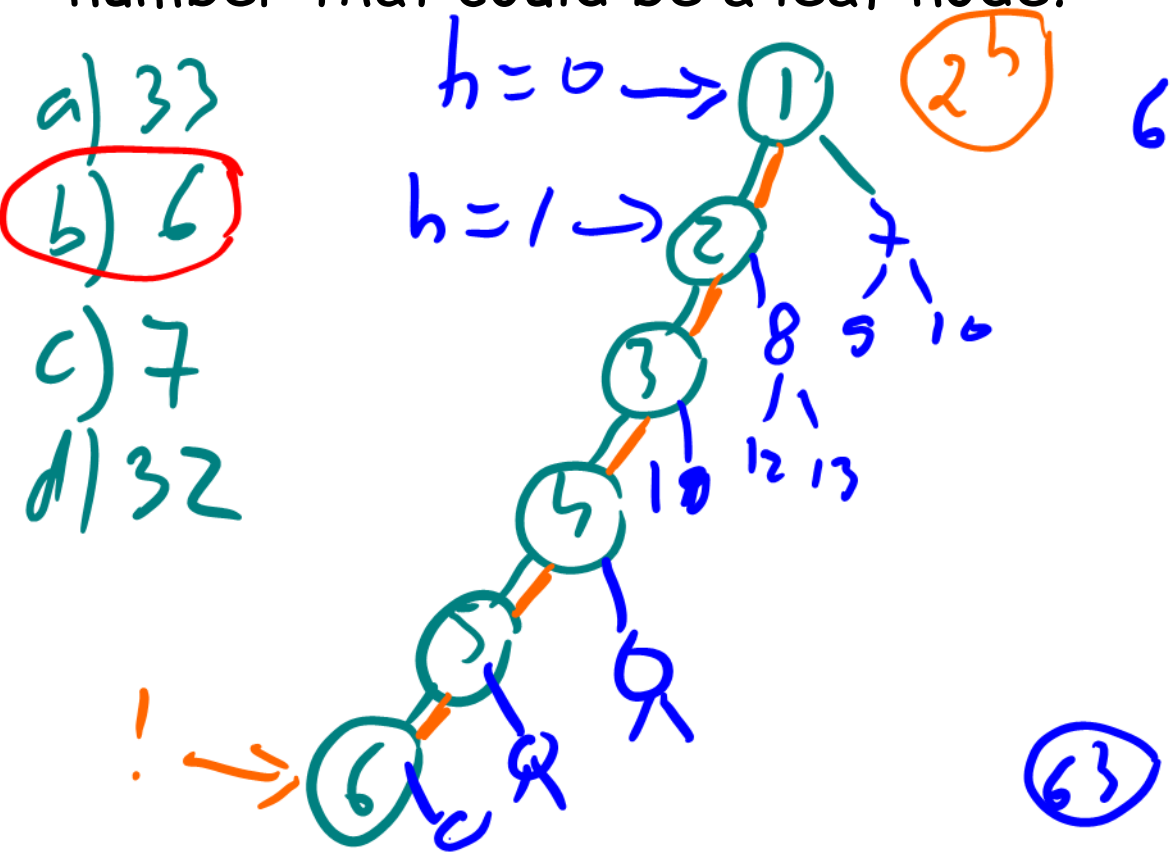
The values 1, 2, 3, . . . , 63 are all inserted (in any order) into an initially empty min-heap. What is the smallest number that could be a leaf node?

a) 33

b) 6

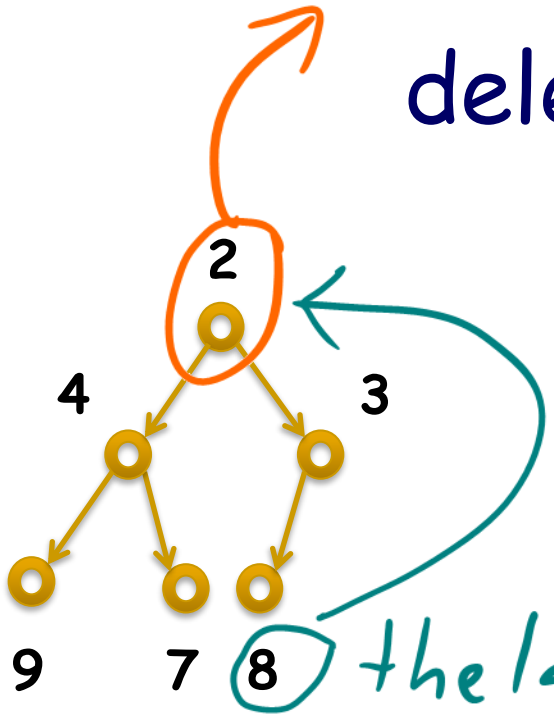
c) 7

d) 32

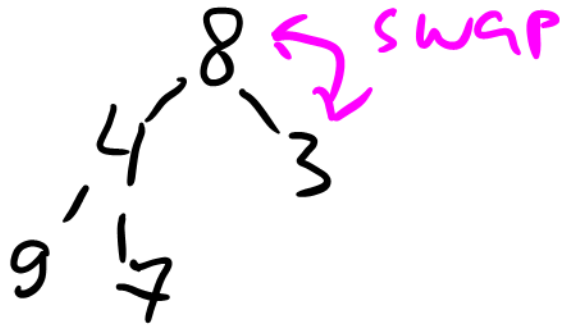


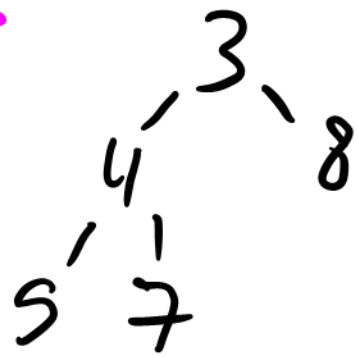
deleteMin (tree reps)

delete the root



the last item



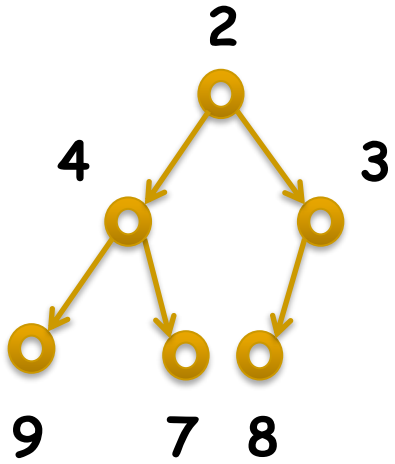


percolate down

Worst-case = # of swaps = $O(\log n)$

implementation

deleteMin (array reps)



0	1	2	3	4	5	6	7
X	2	4	3	9	7	8	
X	8	4	3	9	7		
X	3	4	8	9	7		

swap

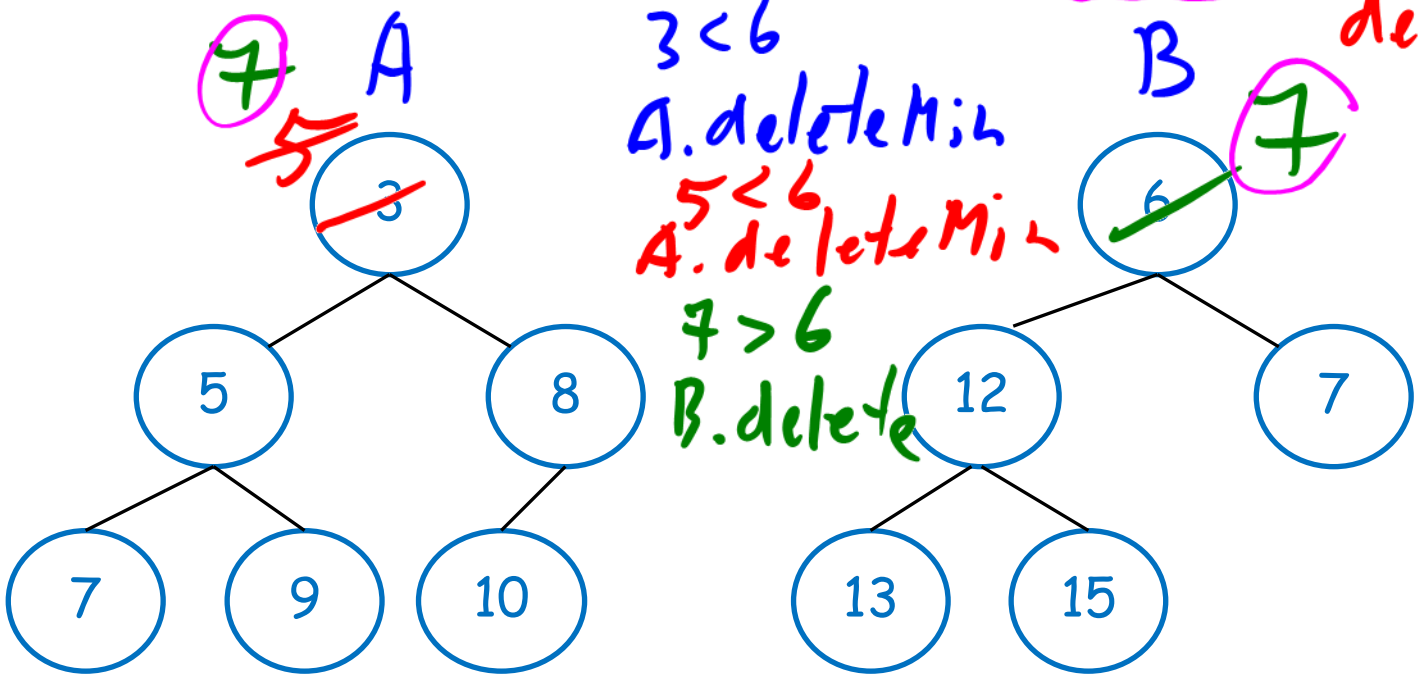
for-loop

G → BFS → spanning tree

Discussion Problem 2

Suppose you have two binary min-heaps, A and B, with a total of n elements between them. You want to discover if A and B have a key in common. Give a solution to this problem that takes time O(n log n). Do not use the fact that heaps are implemented as arrays.

API: insert, deleteMin

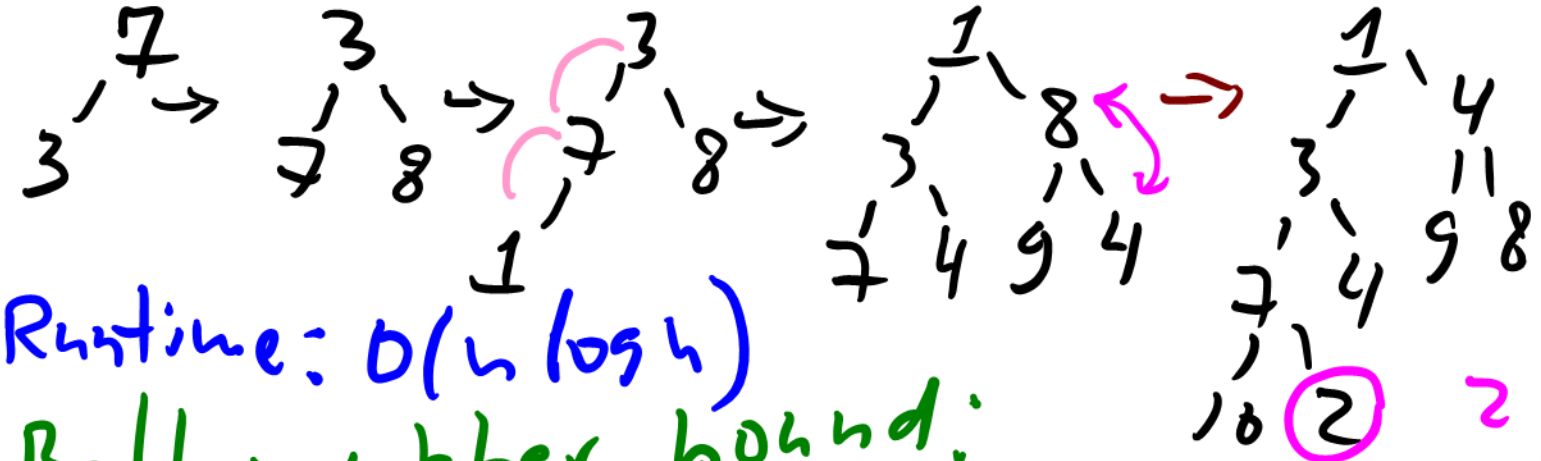


Build a Heap by Insertion

Given an array - turn it into a heap.

input-size n

input: 7, 3, 8, 1, 4, 9, 4, 10, 2, 0



Runtime: $O(n \log n)$

Better upper bound:

$\log 2 + \log 3 + \log 4 + \ldots + \log n \geq \log(n!) =$ (see lect. 1) $= \Omega(n \log n)$

2 swaps

Compute the # of swaps!

Build a Heap in O(n)

Heapify:

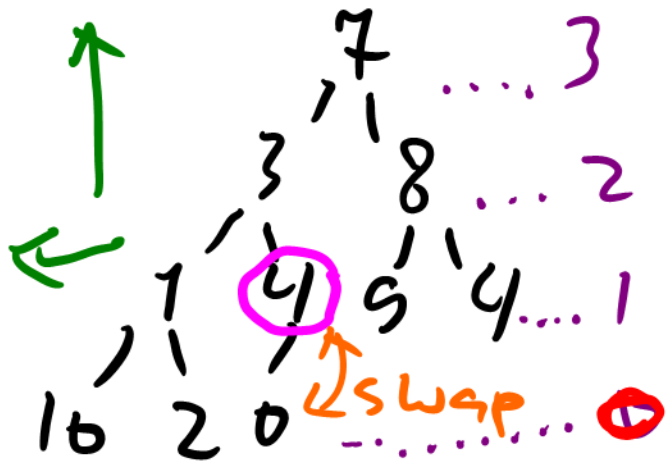
7, 3, 8, 1, 4, ~~9, 4, 10, 2, 0~~

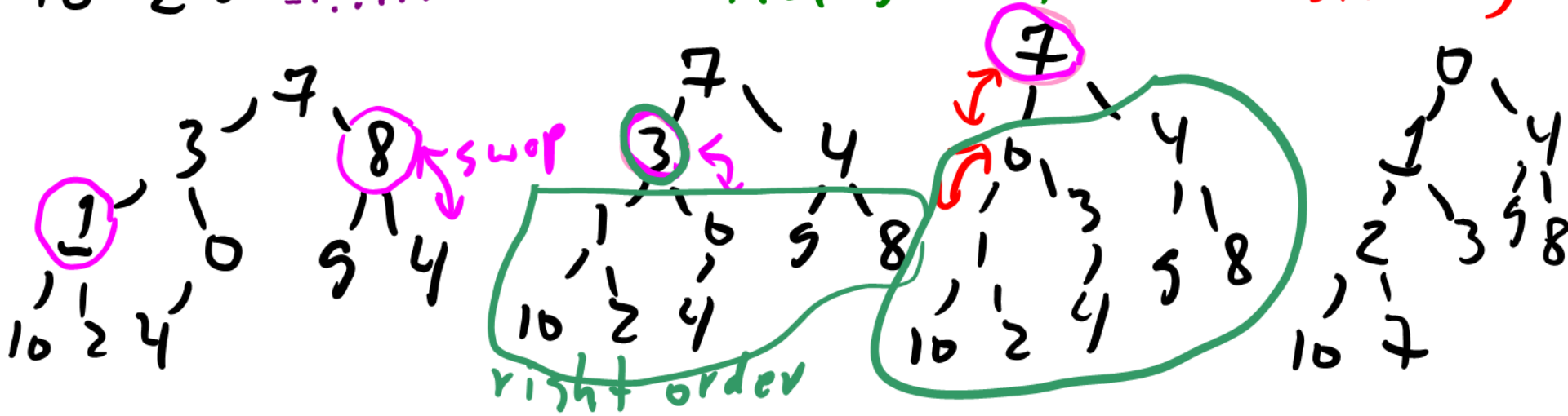
top-bottom

bottom-up

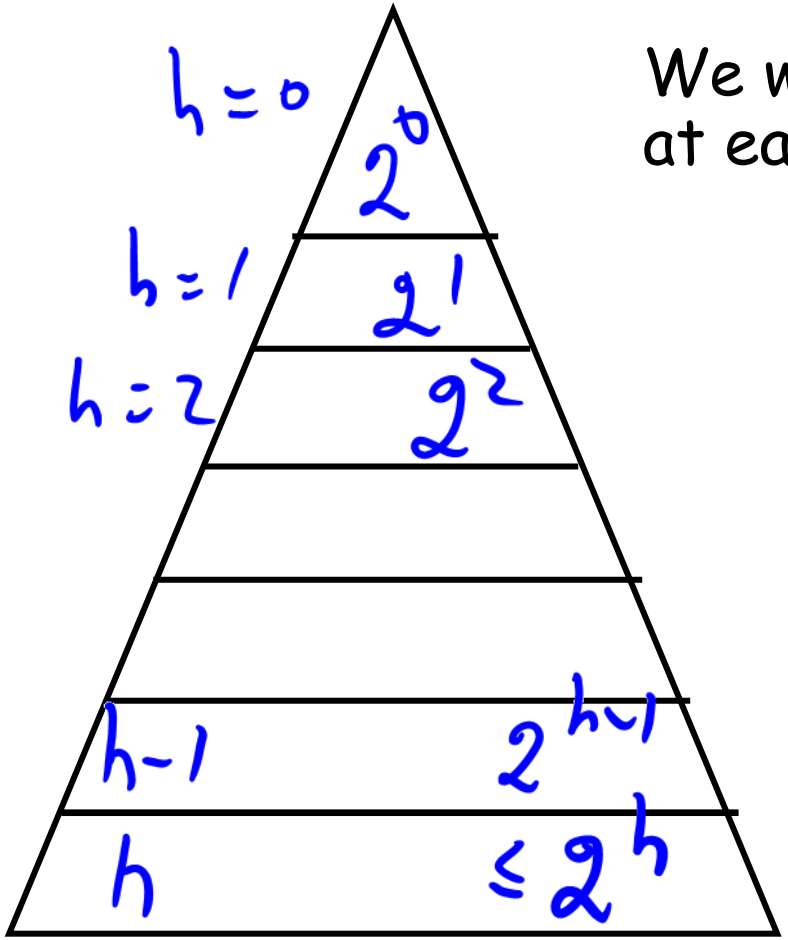
Algo:

1. start at index $\frac{h}{2}$
2. move to the root and swap (percolate down)





heap Complexity of heapify $h = \log n$



We will count the max number of swaps at each level

height	# of nodes	# of swaps
0	1	h
1	2	h-1
2	4	h-2
...	...	...
h-1	2^{h-1}	1

≤ 2

Total # swaps (# of comparisons)

Complexity of heapify

$$1 \cdot h + 2 \cdot (h-1) + 4 \cdot (h-2) + \dots + 2^{h-1} \cdot 1 =$$

$$= \sum_{k=1}^{h} 2^{h-k} \cdot k = 2^h \sum_{k=1}^{h} \frac{k}{2^k} \leq$$

$$\leq 2^h \sum_{k=1}^{\infty} \frac{k}{2^k} = 2^h \cdot O(1) = 2^{\log n} = O(n)$$

$\sum_{k=1}^{\infty} \frac{k}{2^k} \to$ const

$h = \log n$

linear time

Building a BST (by insertion): $n \cdot n = O(n^2)$

Discussion Problem 3

heapsort

How would you sort using a binary heap?

What is it runtime complexity?

Algo: 1. run heapify once, $O(n)$
2. in a loop: deleteMin

Runtime: $O(n + n \cdot \log n) = O(n \log n)$

Sort with a BST (given an array)

Algo: 1. Build a BST, $O(n^2)$
2. in-order traversal, $O(n)$

$O(n^2)$

$2_A, 5, 2_B, 7$

STABLE: $2_A, 2_B, 5, 7$

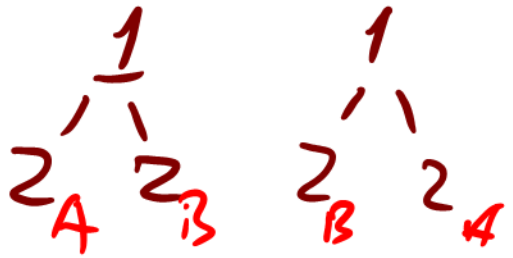
HEAPSORT

2_A precedes 2_B

no extra space

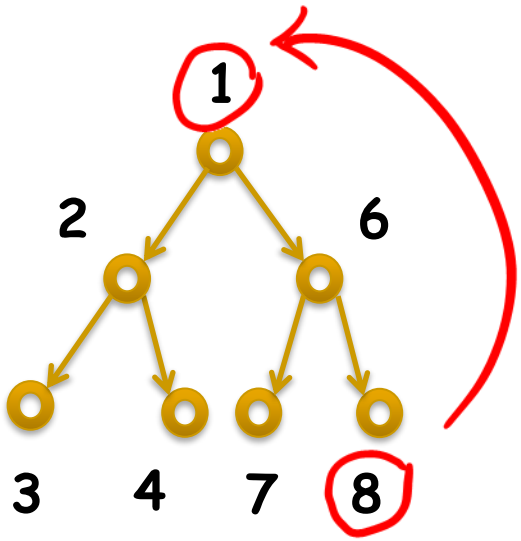
Run delMin n-times

O(n log n)



in-place

nonstable ??



0	1	2	3	4	5	6	7
X	1	2	6	3	4	7	8

	2	3	6	8	4	7	1

	3	4	6	8	7	2	1

	4	8	6	7	3	2	1

sorted part

Discussion Problem 4

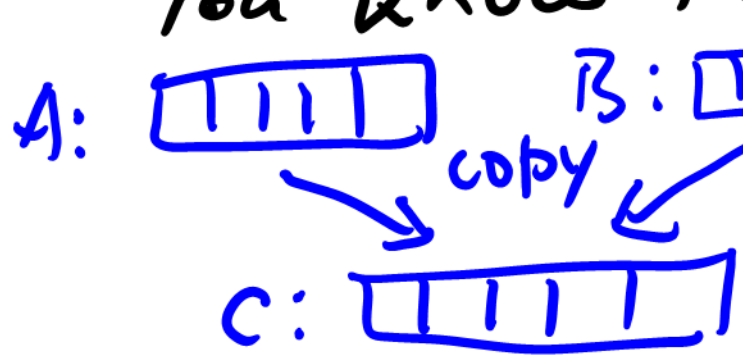
A , B

How would you merge two binary min-heaps?

What is it runtime complexity?

1. API : B.insert(A.deleteMin) , O(n log n)

2. Library implementation.
You know that heap is an array



a) copy A, B to C
b) run heapify on C

O(n)

$k = n/2$

$k = 5$

Discussion Problem 5

Devise a heap-based algorithm that finds the k-th largest element out of n elements. Assume that $n > k$.
What is its runtime complexity?

A. Offline Algorithm (all data available)

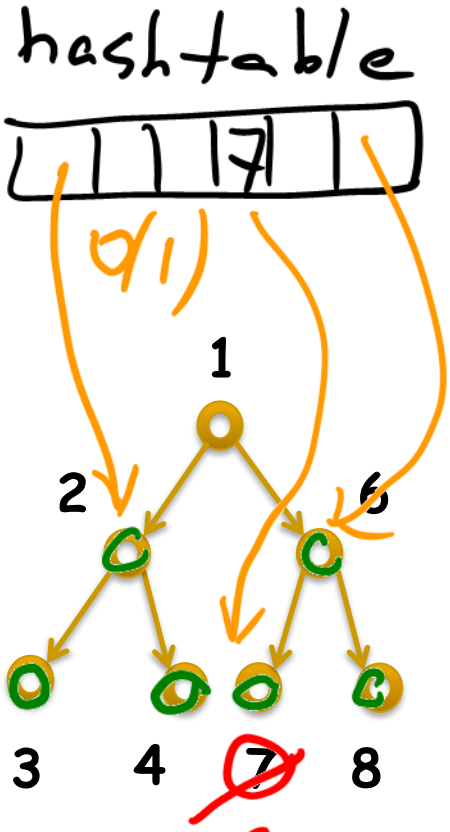
1. build max-heap of size n, $O(n)$
2. deleteMax k times, $O(k \cdot \log n)$

$O(n + k \cdot \log n)$

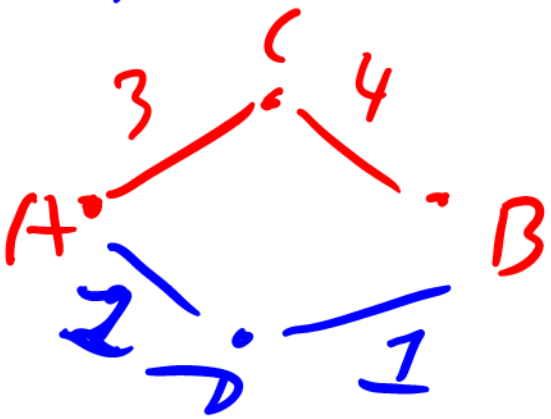
B. Online Algorithm (streaming data)

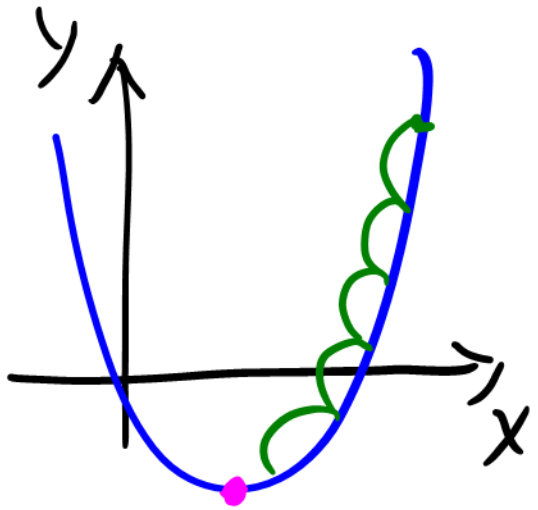
1. build min-heap of size K, $O(K)$
2. wait for $(K+1)$
3. root $< (K+1)$ item, deleteMin, insert
   root $> (K+1)$ item, nothing

$O(K + (n-K)\log K)$



O(log n) decreaseKey for min-heap





percolate up, O(log n)

must be O(1) ???

Algo: 1. find (7) ? ~~O(1)~~

2. replace by 2 and percolate up O(log n)