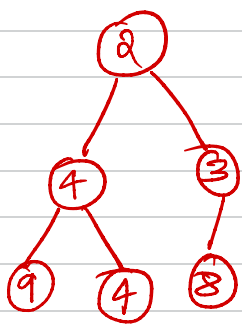
Heap

- Structure property
- Order property

- Heap is implemented as an array
- Heap is a complete tree

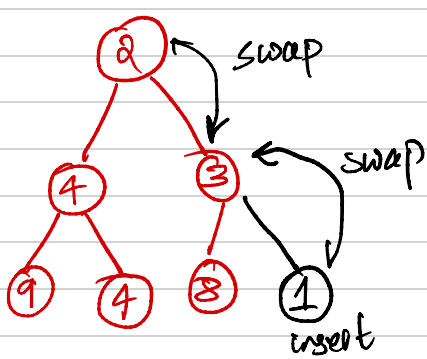
Parent - $k/2$ index
left child - 2^k index
right child - $2^k + 1$ index

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



Insert

- Insert to a gap
- restore ordering property



Runtime:

of swaps in worst case

$O(\log n)$

Insert array rep'

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

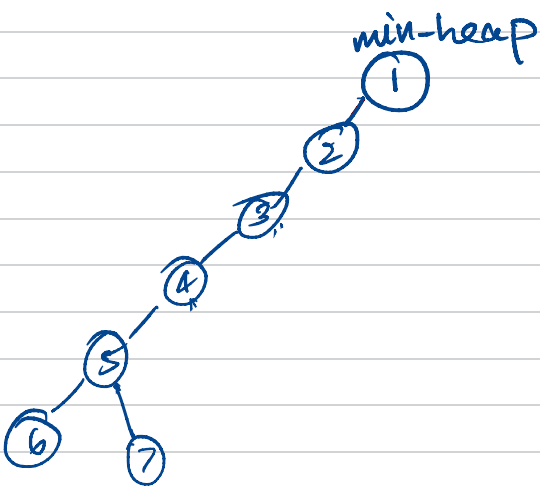
Implementation: It is a single for loop

This process is called percolation.

DP1

Discussion Problem 1

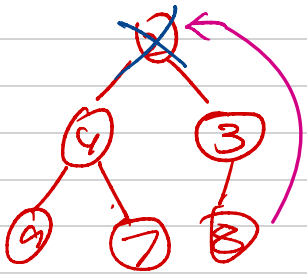
The values 1, 2, 3, . . . , 63 are all inserted (in any order) into an initially empty min-heap. What is the smallest number that could be a leaf node?



Proof by example

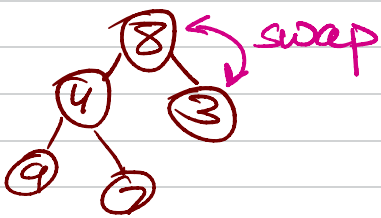
$63 = 64 - 1 = 2^6 - 1$

deleteMin (remove the root)

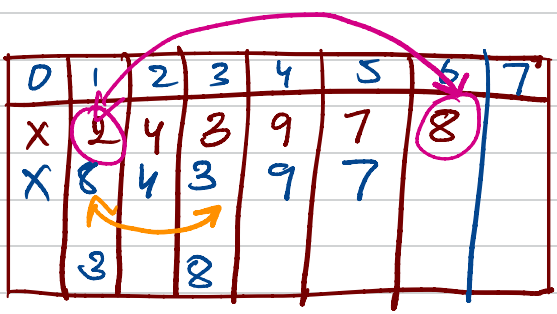


① preserve structural property : move the last item to the root

② percolate down



Runtime : O(log n)



Implementation: a single for loop

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, use only API operations: insert and deleteMin.

Compare $3 < 6$

A.delete

$5 < 6$

A.delete

$7 > 6$

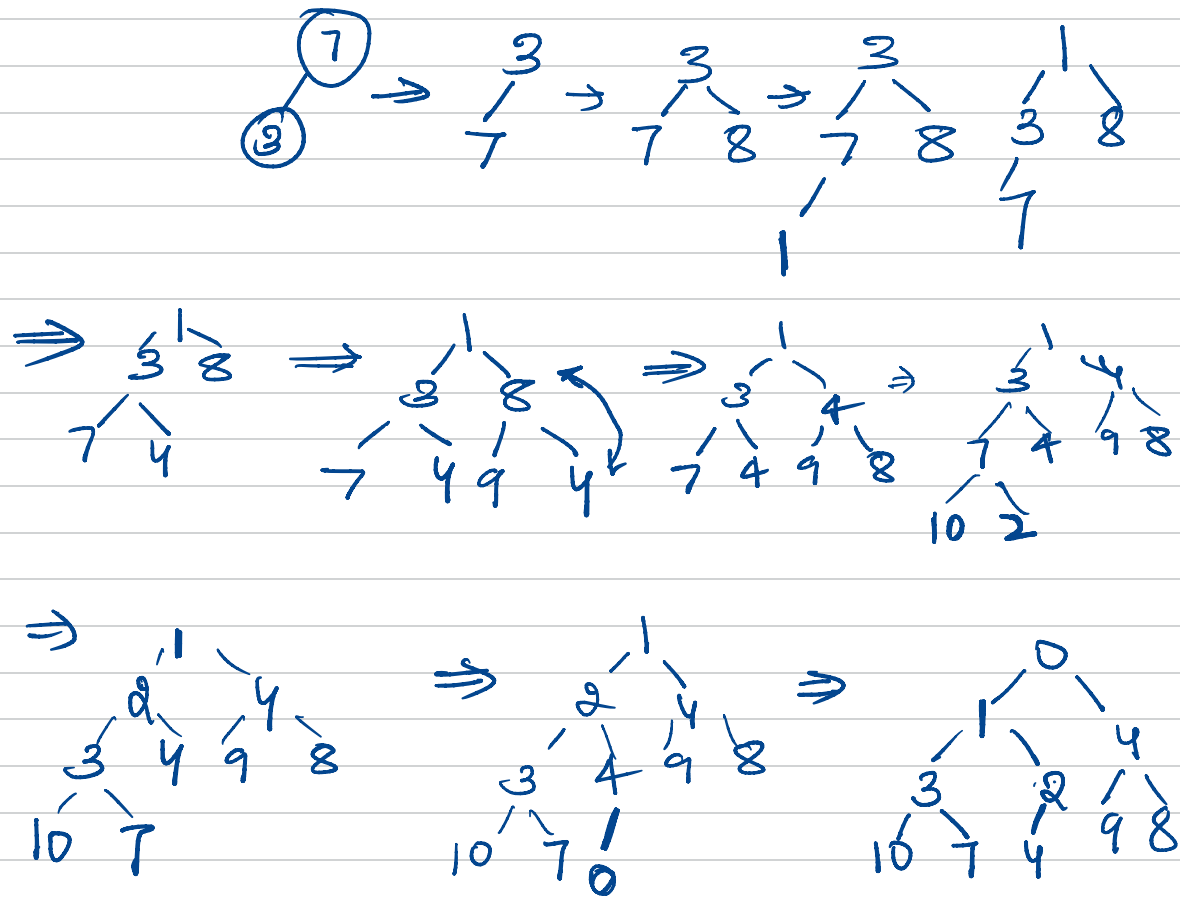
B.delete

$7 = 7$

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

Build a Heap by insertion

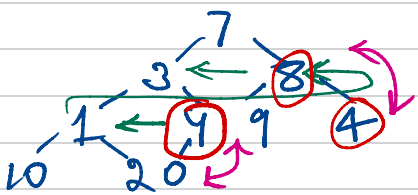
7, 3, 8, 1, 4, 9, 4, 10, 2, 0



Runtime = $O(n \log n)$

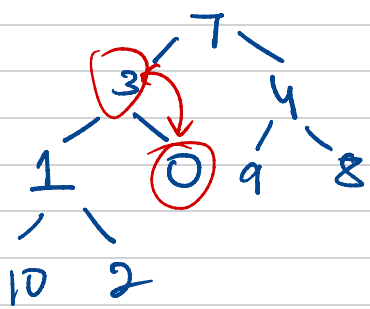
Build a heap in O(n)

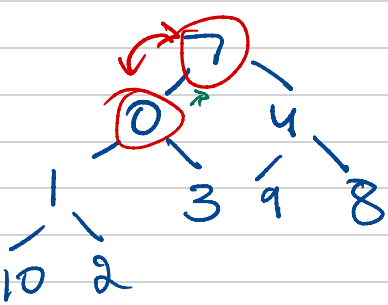
Heapify : 7, 3, 8, 1, 4, 9, 4, 10, 2, 0

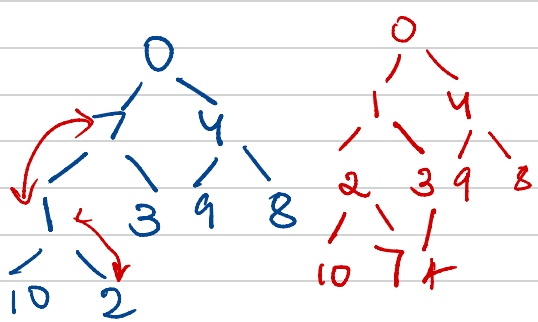


Algorithm

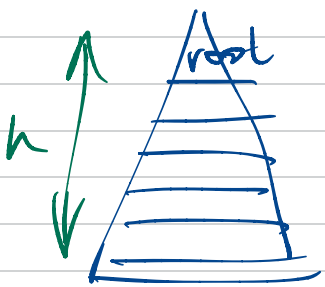
1. Start at index $n/2$
2. Compare w/ its children, & swap with the smallest child if it exists
3. Go to the left sibling
4. Repeat #2







Complexity of Heapify



height	# of nodes	# swaps
0 root	1	h
1	2	h-1
2	4	h-2
3	8	
⋮	⋮	⋮
h-1	2^{h-1}	1

Runtime = Total cost =

$$1 \times h + 2 \times h-1 + 4 \times h-2 + 2^{h-1} \times 1$$

$$= \sum_{k=0}^{h-1} 2^k \times (h-k) = \text{Wolfram Alpha}$$

$$= O(n)$$

P.T.O

Discussion Problem 3

How would you sort using a binary heap?

What is it runtime complexity?

Sorting using BST : $O(n^2)$

1. Make a BST : $O(n^2)$
2. Traversal, inorder : $O(n)$

Heapsort algorithm :

1. Make a heap : $O(n)$ - heapify
2. delete : $O(n \log n)$

From GeeksforGeeks

- For ascending order use a max heap
- call deleteMax()
- Restructure
- Continue for the whole heap

Discussion Problem 4

How would you merge two binary min-heaps?

What is it runtime complexity? O(n)

Since the heap is basically an array & 2 heaps are just 2 arrays, merge the 2 heaps & heapify.

Assuming I know the input.

(1) Offline algorithm

All data is available, we can preprocess the data & use heapify. O(n)

(2) Online algorithm (Streaming data)

Here we will be using insertion, an dhence O(n log n) by insertion

Discussion Problem 5

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

① Offline algorithm (all data available)

Algorithm:

a) build a max-heap
b) run delete k times

Runtime: $O(n + k \log n)$

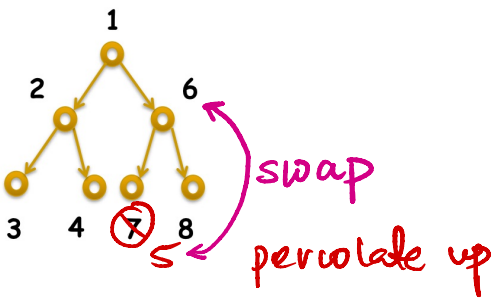
② Online algorithm (streaming data)

Algorithm:

a) wait for k items, then build a heap, min-heap
b) wait for the $(k+1)$st element
c) new item $\leq$ root, ignore it
else: run deleteMin
then insert

decreaseKey

decreaseKey



Runtime : $O(\log n)$
worst case

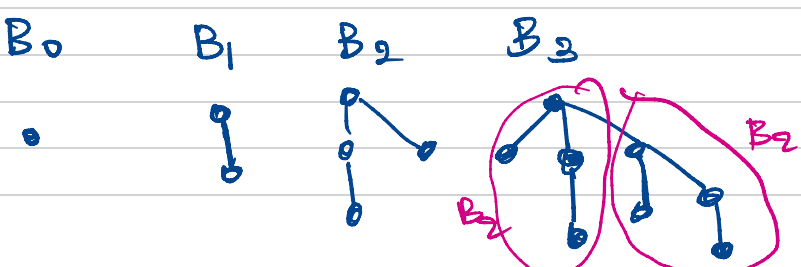
We find the data in $O(1)$ by using hash tables.

Hash tables contain the pair of index & value & when you search for an element we can hash its index & change it in the heap array.

Binomial Trees B_K

B_K is defined as

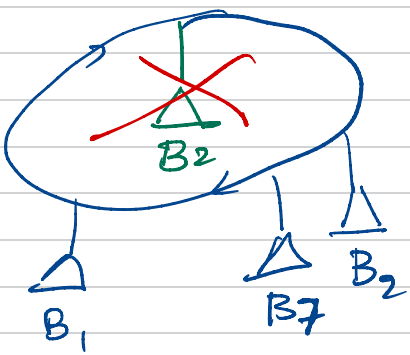
1. B_0 is a single node
2. B_K is formed by joing 2 B_{K-1} trees



Binomial Heaps

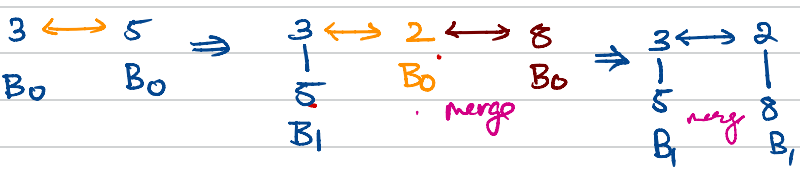
A binomial heap is a collection (a linked list or a queue) of at most Celling(log n) binomial trees (of unique rank) in increasing order of size where each tree has a heap ordering property.

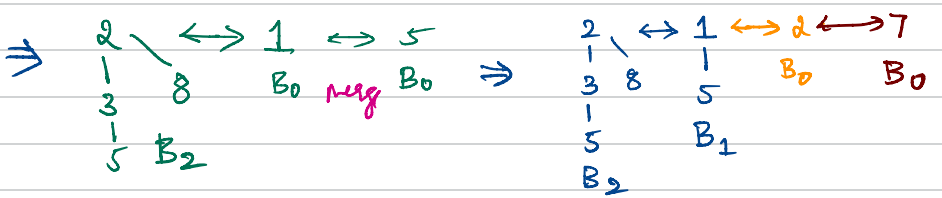
min heap

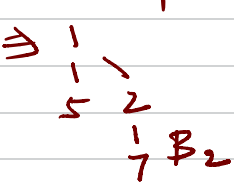


Discussion Problem 6

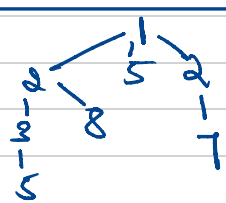
Given a sequence of numbers: 3, 5, 2, 8, 1, 5, 2, 7.
Draw a binomial heap by inserting the above numbers reading them from left to right







FINAL HEAP



Discussion Problem 7

How many binomial trees does a binomial heap with 25 elements contain?
What are the ranks of those trees?

$25_2 = (16+8+1)_2 =$ 11001

B_4 B_3 B_0

Insertion

Insertion

What is its worst-case runtime complexity?

$15_2 = (8+4+2+1)_2 =$ 1111

add 1 — 1

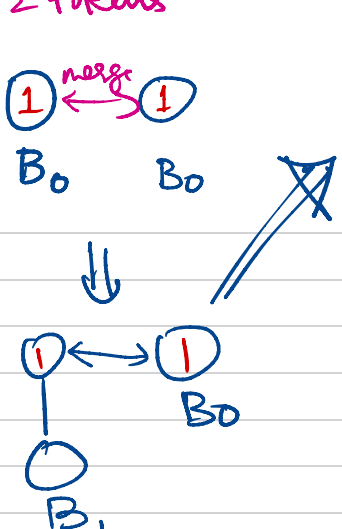
10000

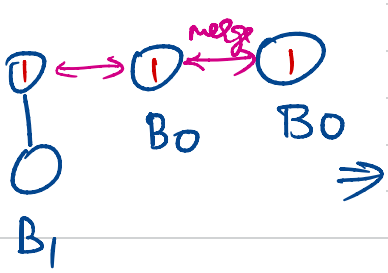
$O(\log n)$

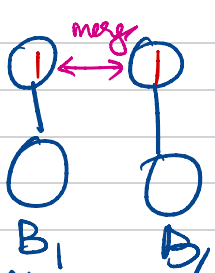
What is its amortized runtime complexity?

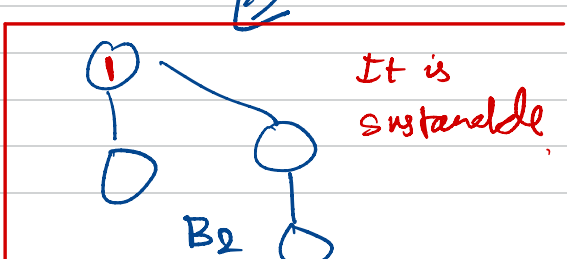
Use an accounting method. how many tokens for each insert

2 tokens







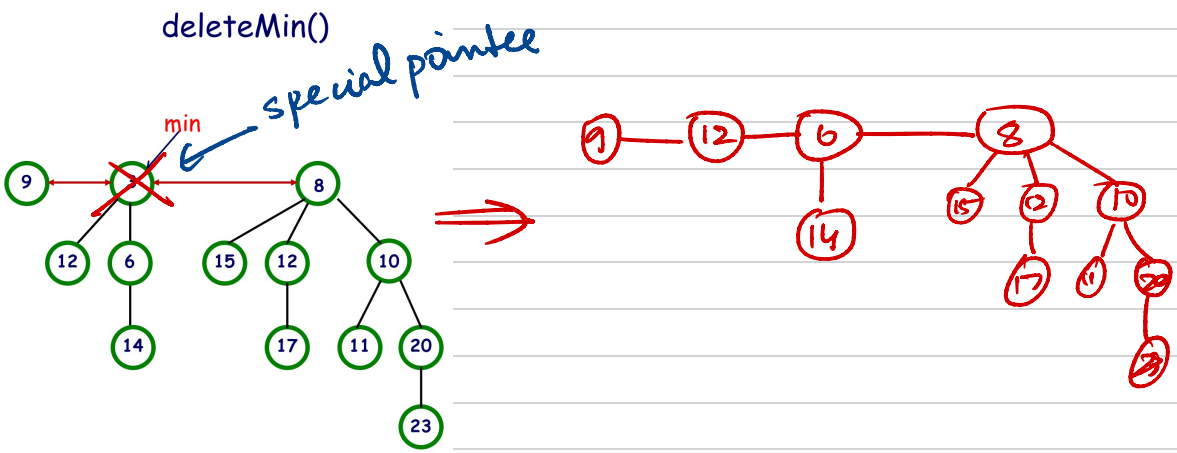


Building : Binomial vs Binary Heaps

The cost of inserting n elements into a binary heap, one after the other, is $\Theta(n \log n)$ in the worst-case. This is an online algprithm.

If n is known in advance (an offile algorithm), we run heapify, so a binary heap can be constructed in time $\Theta(n)$.

The cost of inserting n elements into a binomial heap, one after the other, is $\Theta(n)$ (amortized cost), even if n is not known in advance.



P.T.O

deleteMin()

Algorithm:

1. We delete the min in O(1) by make use of a special pointer to track the minimum element.
2. Move the subtrees to the top (LL) level
3. Traverse the LL & merge trees of the same rank
4. Update the min pointer

Runtime: O(log n)

Discussion Problem 8

Devise an algorithm for merging two binomial heaps and discuss its complexity. Merge $B_0B_1B_2B_4$ with B_1B_4.

Merging 2 binary heaps :- O(n)
binomial heaps — O(log n)

Algorithm for binomial heaps

① Merge 2 LLs — O(1)

② Traverse & merge binomial trees of the same rank, O(log n)

```
 10111
+10010
------
101001
```

$\Longrightarrow B_0\ B_3\ B_5$

FIBONACCI HEAPS

Idea: *relaxed (lazy)* binomial heaps

Goal: decreaseKey in O(1) ac.

Runtime Complexities

	Binary	Binomial	Fibonacci
findMin	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
deleteMin	$\Theta(\log n)$	$\Theta(\log n)$	$O(\log n)$ (ac)
insert	$\Theta(\log n)$	$\Theta(1)$ (ac)	$\Theta(1)$
decreaseKey	$\Theta(\log n)$	$\Theta(\log n)$	$\Theta(1)$ (ac)
merge	$\Theta(n)$	$\Theta(\log n)$	$\Theta(1)$ (ac)