HEAPS

Problem Solving with Computers-II

```
#include <iostream>
using namespace std;

int main(){
    cout<<"Hola Facebook\n";
    return 0;
}
```

How is PA2 going ?

A. Finished !

B. Making progress, on track to finish

C. Some progress

D. Little progress

E. Haven't started.

Heaps (priority queue)

- Clarification
  - *heap*, the data structure is not related to *heap,* the region of memory
- What are the operations supported?
- What are the running times?

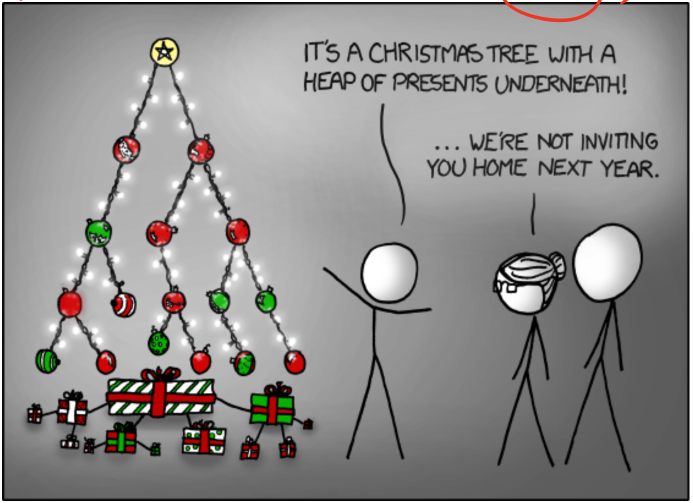
top()

insert (push), min, deletemin
OR ″ max, deletemax

push: O(log n)

min: O(1)

delete min: O(logn).



IT'S A CHRISTMAS TREE WITH A HEAP OF PRESENTS UNDERNEATH!
... WE'RE NOT INVITING YOU HOME NEXT YEAR.

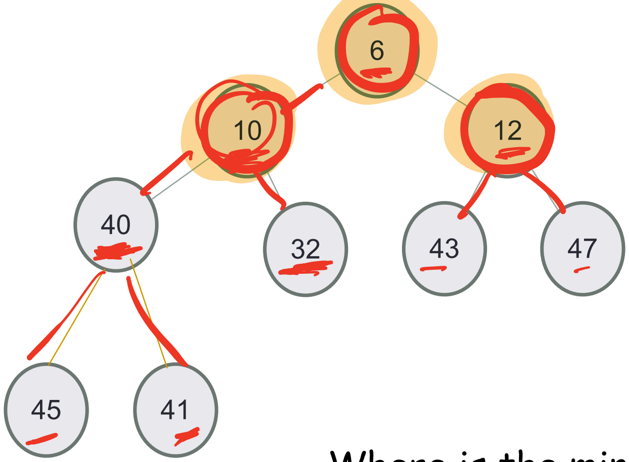
Heaps as binary trees

- **Rooted binary tree that is as complete as possible**
- **In a min-Heap, each node satisfies the following heap property:**

key(x)<= key(children of x)

duplicates are allowed

Min Heap with 9 nodes

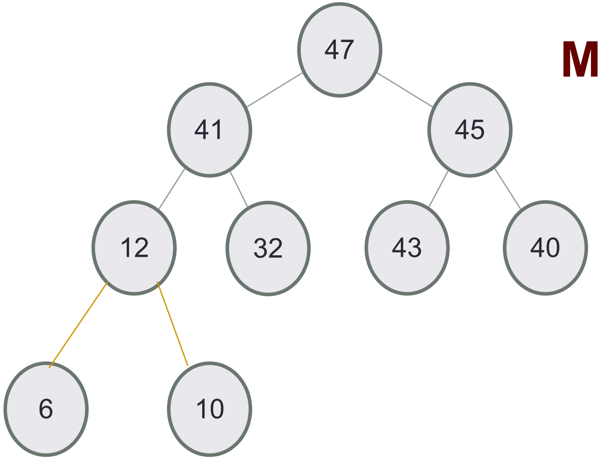


Where is the minimum element?

Heaps as binary trees

- **Rooted binary tree that is as complete as possible**
- **In a max-Heap, each node satisfies the following heap property:**

  key(x)>= key(children of x)



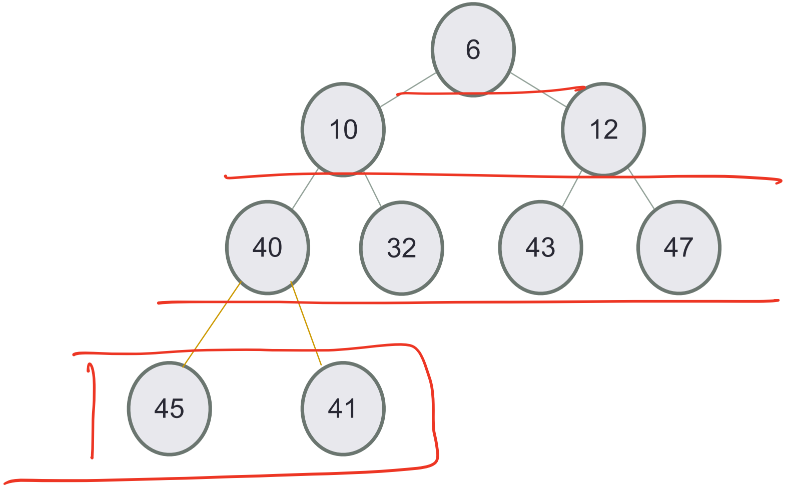
Max Heap with 9 nodes

Where is the maximum element?

Structure: Complete binary tree

A heap is a complete binary tree: Each level is as full as possible.

Nodes on the bottom level are placed as far left as possible



$H = O(\log n)$

Identifying heaps

Starting with the following min-Heap which of the following operations will result in something that is NOT a min Heap

A. Swap the nodes 40 and 32

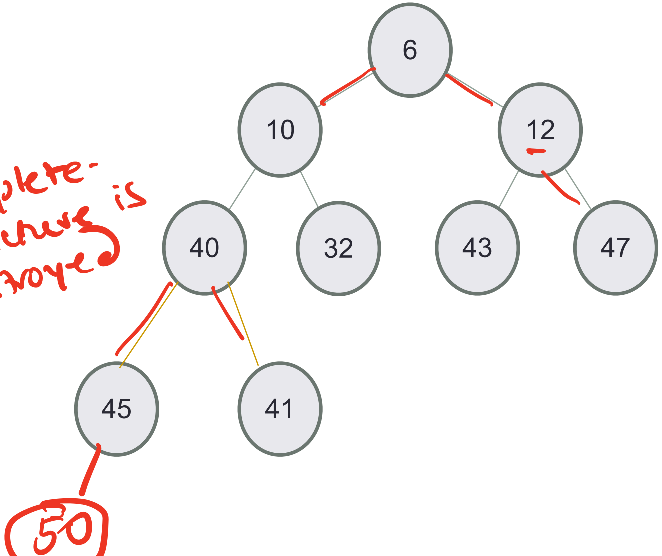
B. Swap the nodes 32 and 43

[✓ C. Swap the nodes 43 and 40]

[✓ D. Insert 50 as the left child of 45]

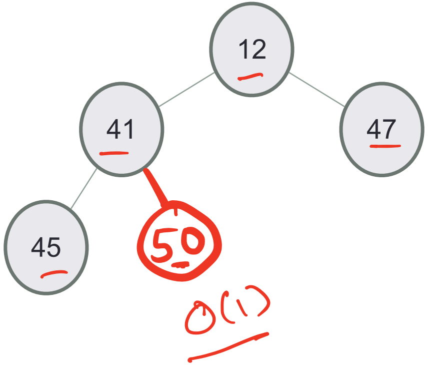
[✓ E. C&D]

complete-structure is destroyed



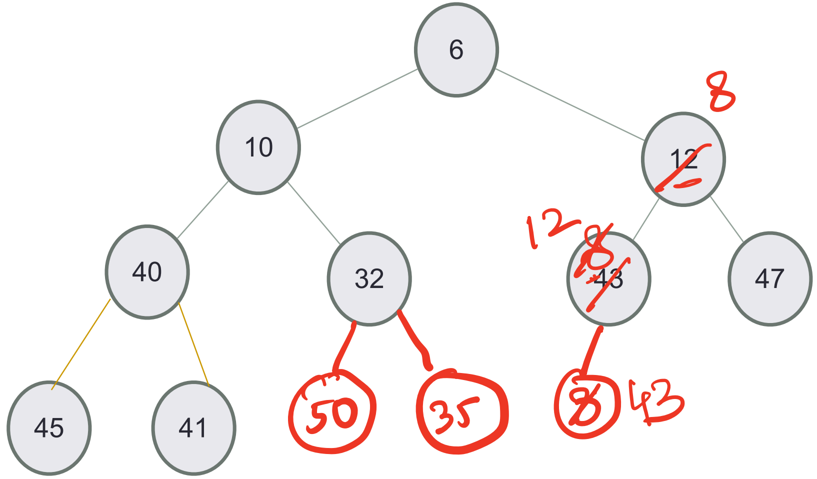
Insert 50 into a min-heap

- Insert key(x) in the first open slot at the last level of tree (going from left to right)
- If the heap property is not violated - Done
- Else: while(key(parent(x))>key(x)) swap the key(x) with key(parent(x))



Insert 50, then 35, then 8

8 Bubbling up until the right it reaches spot



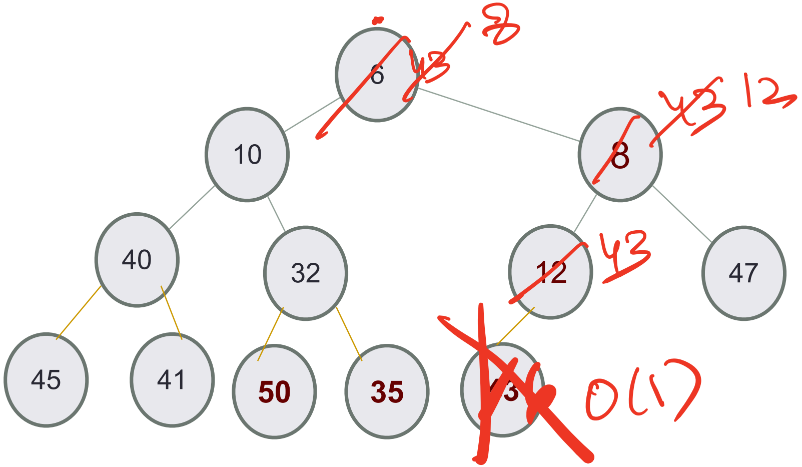
push()

Delete min

pop()

- Replace the root with the rightmost node at the last level
- "Bubble down"- swap node with child with the smallest key value until the heap property is restored

O(log(n))

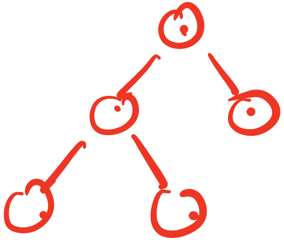


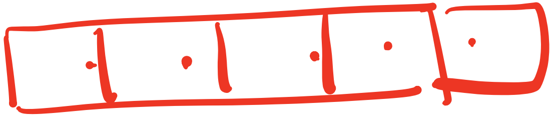
Under the hood of heaps

priority_queue <int, vector<int>, greater<int>>

- An efficient way of implementing heaps is using vectors
- Although we think of heaps as trees, the entire tree can be efficiently represented as a vector!!

heap (in our head)

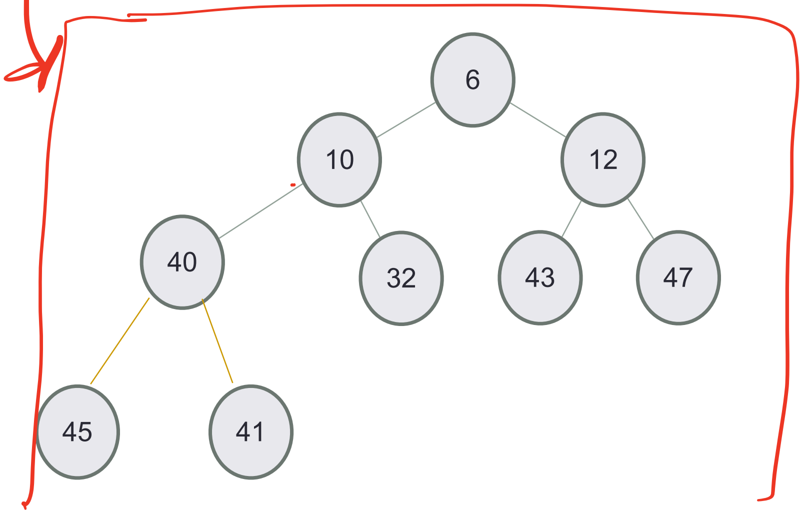




heap (practical)

Implementing heaps using an array or vector

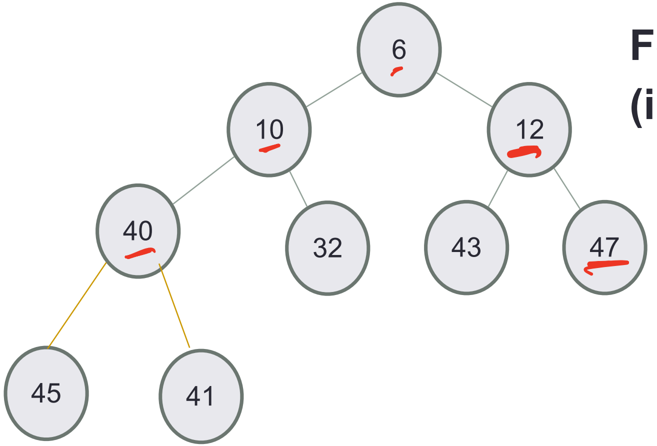
Value	6	10	12	40	32	43	47	45	41	
Index	0	1	2	3	4	5	6	7	8	9



Using vector as the internal data structure of the heap has some advantages:

- More space efficient than trees
- Easier to insert nodes into the heap

Finding the “parent” of a “node” in the vector representation



For a key at index i, index of the parent is (i-1)/2

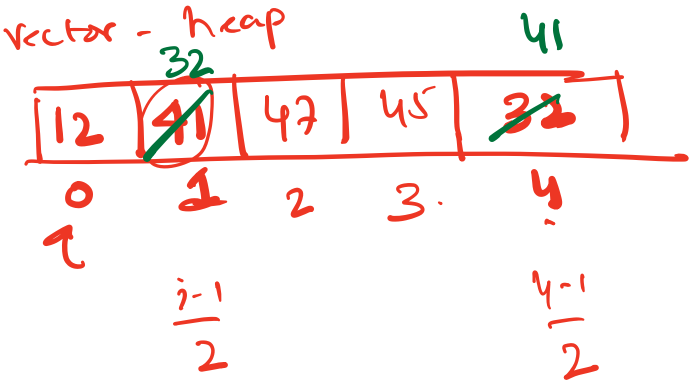
parent Index $(i) = \left\lfloor \frac{i-1}{2} \right\rfloor$

	Value	**6**	**10**	**12**	**40**	**32**	**43**	**47**	**45**	**41**
i	Index	0	1	2	3	4	5	6	7	8
Index of parent		-	0	0	1	1	2	2	3	3

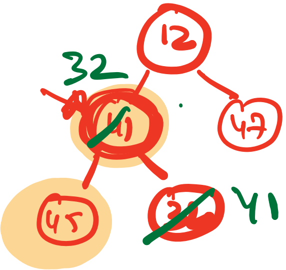
Insert into a heap push

- Insert key(x) in the first open slot at the last level of tree (going from left to right)
- If the heap property is not violated - Done
- Else….

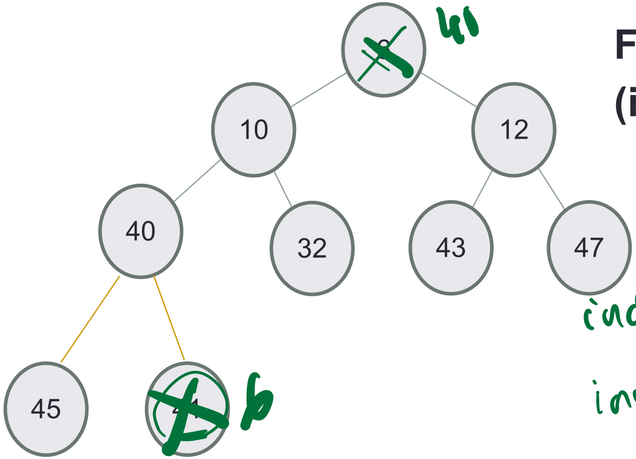
Insert the elements {12, 41, 47, 45, 32} in a min-Heap using the vector representation of the heap



0	1	2	3	4
12	~~41~~ 32	47	45	~~32~~ 41



Insert 50, then 35



For a node at index i, index of the parent is (i-1)/2

V. pop-back()

index_leftchild (i) =

index_rightchild (i) =

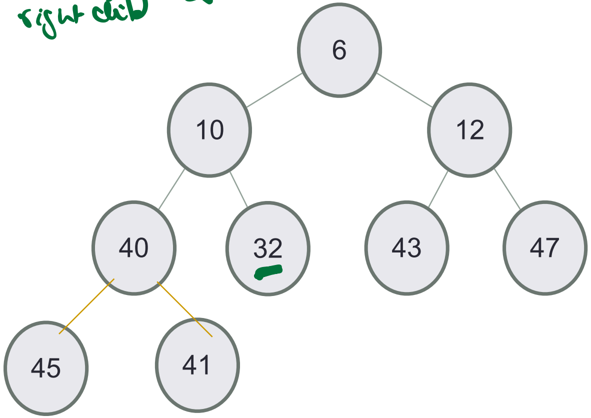
V

Value	~~6~~ 41	10	12	40	32	43	47	45	~~41~~ ~~6~~	
Index	0	1	2	3	4	5	6	7	8	

Traversing down the tree

i

Value	6	10	12	40	32	43	47	45	41	
i Index	0	1	2	3	4	5	6	7	8	
Index of left child	1	3	5	7	~~9~~	~~11~~	~~13~~			
Index of right child	2	4	6	8	~~10~~	~~12~~	~~14~~			



For a node at index i, what is the index of the left and right children?

A. (2*i, 2*i+1)

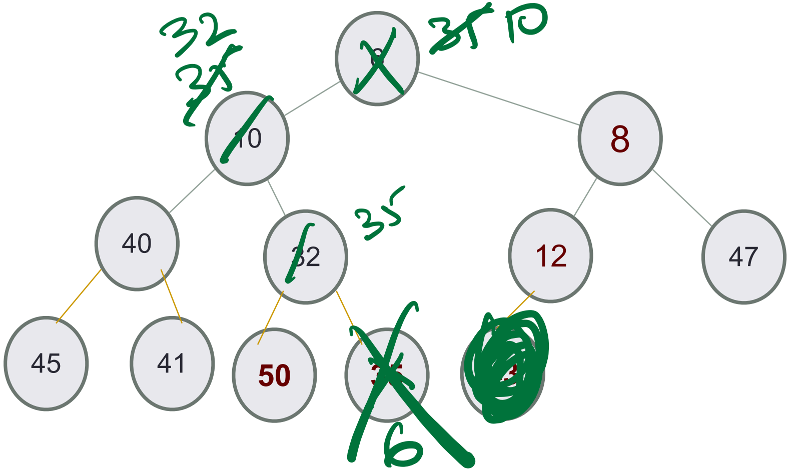
[✓ B.] (2*i+1, 2*i+2)

C. (log(i), log(i)+1)

D. None of the above

Delete min

- **Replace the root with the rightmost node at the last level**
- **"Bubble down"- swap node with one of the children until the heap property is restored**



Delete min (pop)

Value	~~6~~ ~~35~~ 10	~~10~~ ~~35~~ 32	12	40	~~32~~ 35	43	47	45	41	50	~~35~~ ~~6~~
Index	0	1	2	3	4	5	6	7	8	9	10

What is the resulting vector after doing a pop()?