HEAPS

Problem Solving with Computers-II

How is PA02 going?

A. Done
B. On track to finish
C. Having some difficulties
D. Just started
E. Haven’t started

Heaps

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

Operations:

min or max $O(1)$

insert $O(\log N)$

delete $O(\log N)$

Heaps

	Min-Heaps	Max-Heap	(Balanced) BST
• Insert :	O(logN)	O(logN)	O(logN)
• Min:	O(1)	—	"
• Delete Min:	O(logN)	—	
• Max	—	O(1)	O(logn)
• Delete Max	—	O(logN)	O(log)

Applications:

- Efficient sort
- Finding the median of a sequence of numbers
- Compression codes

Choose heap if you are doing repeated insert/delete/(min OR max) operations

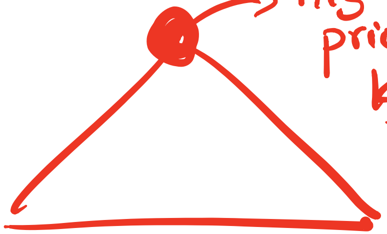
std::priority_queue (STL's version of heap)

A C++ `priority_queue` is a generic container, and can store any data type on which an ordering can be defined: for example `ints`, `structs (Card)`, pointers etc.

type of keys

```
#include <queue>
priority_queue<int> pq;
```

heap object



highest priority key | max key

Methods:

```
* push()   //insert
* pop()    //delete max priority item
* top()    //get max priority item
* empty()  //returns true if the priority queue is empty
```

- You can extract object of highest priority in O(log N)
- To determine priority: objects in a priority queue must be comparable to each other

STL Heap implementation: Priority Queues in C++

What is the output of this code?

```
priority_queue<int> pq;
pq.push(10);
pq.push(2);
pq.push(80);
cout<<pq.top();
pq.pop();
cout<<pq.top();
pq.pop();
cout<<pq.top();
pq.pop();
```

80 10 2

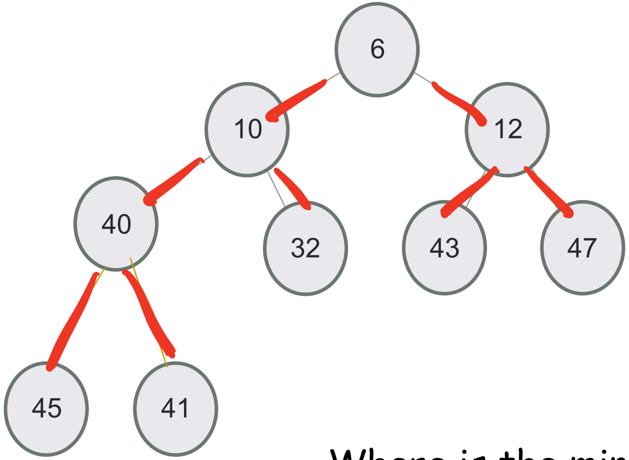
A. 10 2 80
B. 2 10 80
C. 80 10 2
D. 80 2 10
E. None of the above

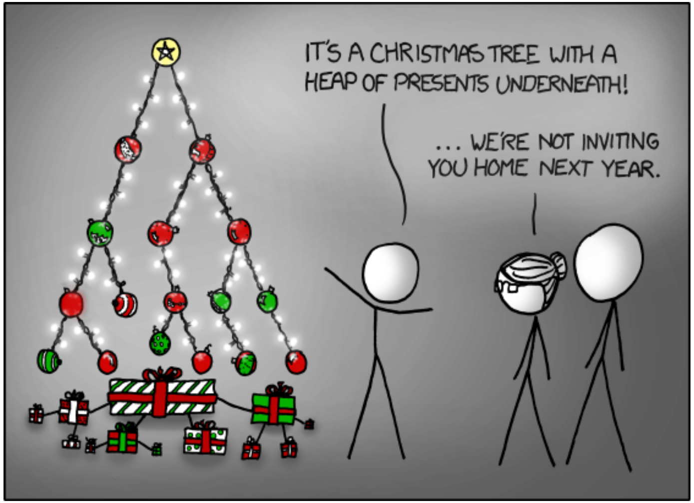
Heaps as binary trees

- **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)

Min Heap with 9 nodes



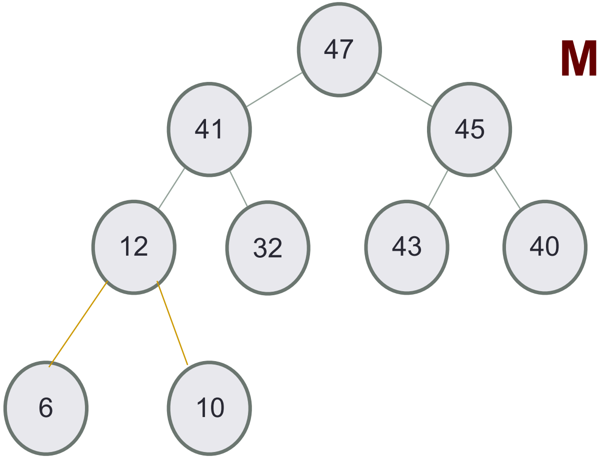


Where is the minimum element?

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)

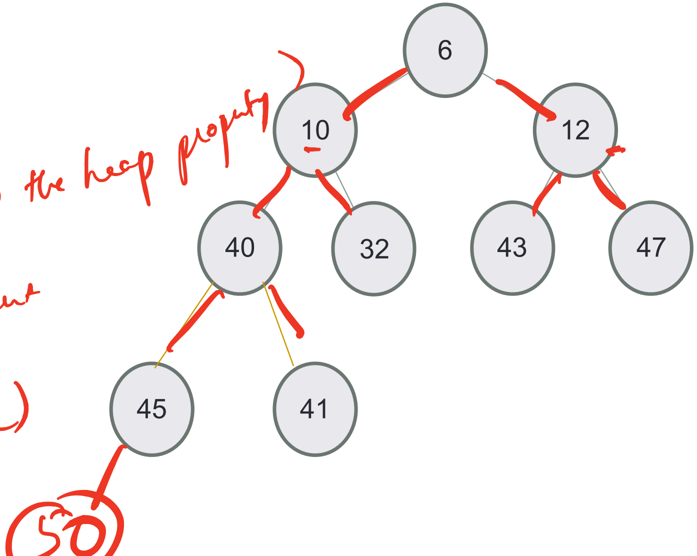


Max Heap with 9 nodes

Where is the maximum element?

Identifying heaps

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

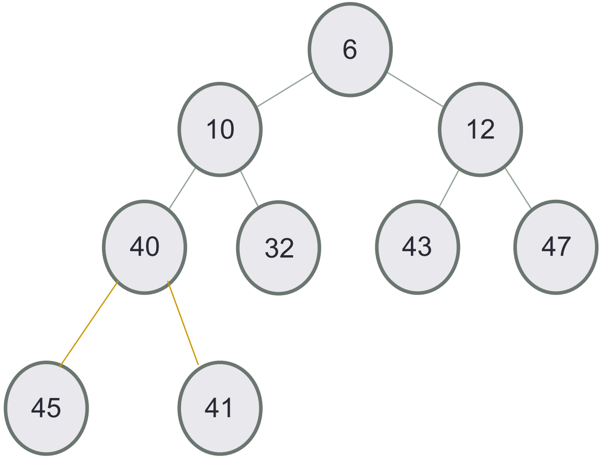


A. Swap the nodes 40 and 32
B. Swap the nodes 32 and 43
C. Swap the nodes 43 and 40 (Violates the heap property)
D. Insert 50 as the left child of 45
E. C&D

(Violates the requirement that the heap should be a complete tree)

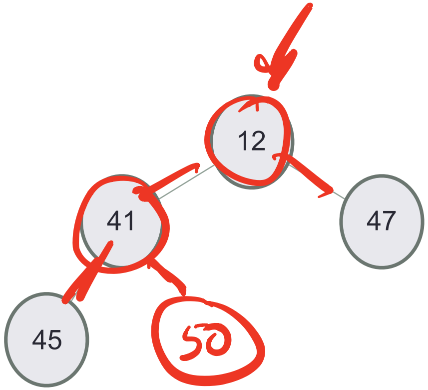
Structure: Complete binary tree

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

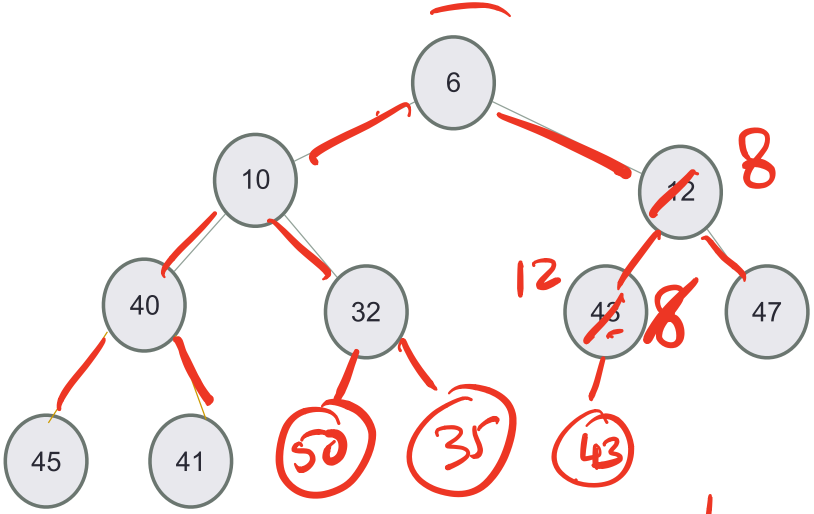


Insert 50 into a heap

- 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))



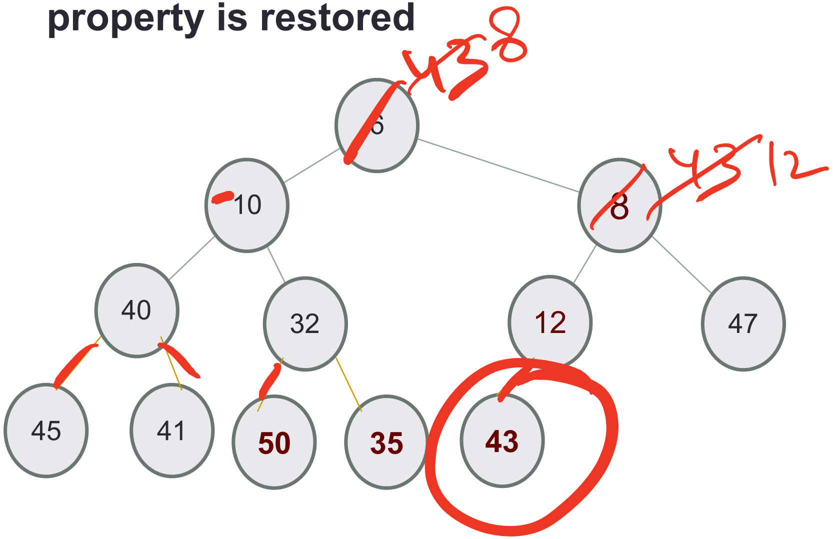
Insert 50, then 35, then 8



bubbling up!

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

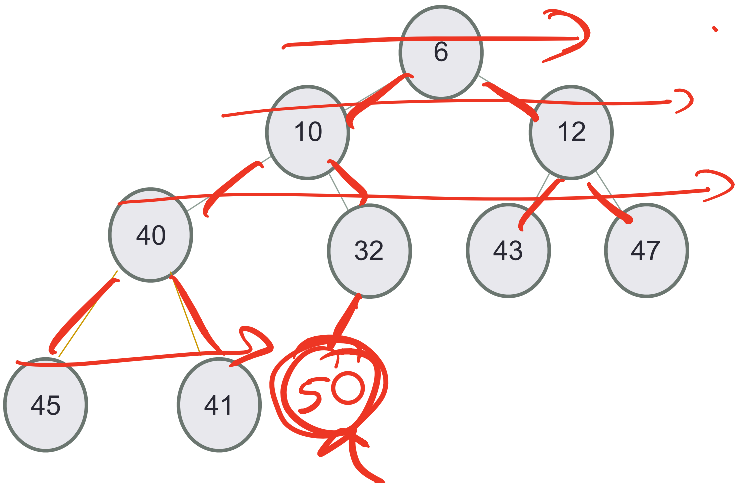


Under the hood of heaps

- 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!!

Implementing heaps using an array or vector

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



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

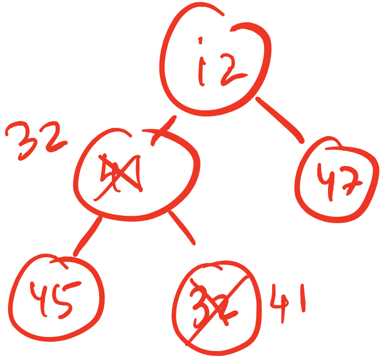
- **More space efficient than trees**
- **Easier to insert nodes into the heap**

Insert into a heap

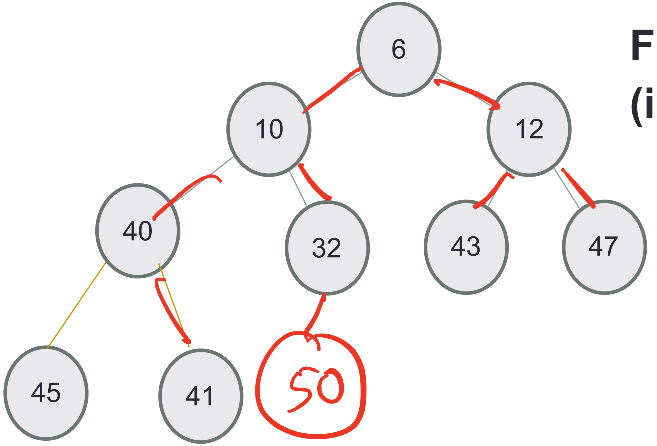
- Insert key(x) in the first open slot at the last level of tree (going from left to right)
- If the heap property is not violated - Done
- Else….

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

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



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



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

For node at index i,

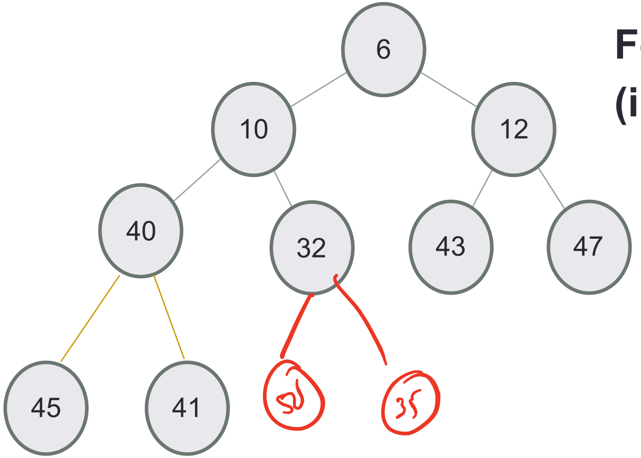
$$\text{Parent}(i) = \frac{(i-1)}{2}$$

$$\text{leftchild}(i) = 2i+1$$

$$\text{Rightchild}(i) = 2i+2$$

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

Insert 50, then 35



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

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

Insert 8 into a heap

Value	6	10	~~12~~ 8	40	32	~~43~~ ~~8~~ 12	47	45	41	50	35	$\boxed{\cancel{8}}$ 43
Index	0	1	2	3	4	5	6	7	8	9	10	11

After inserting 8, which node is the parent of 8 ?

A. Node 6

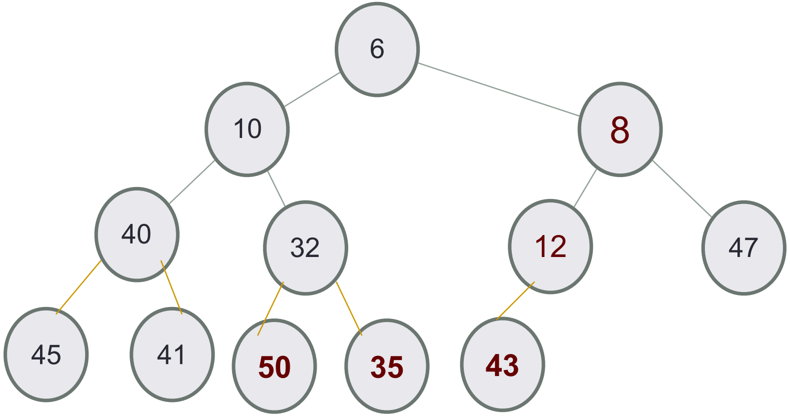
B. Node 12

C. None 43

D. None - Node 8 will be the root

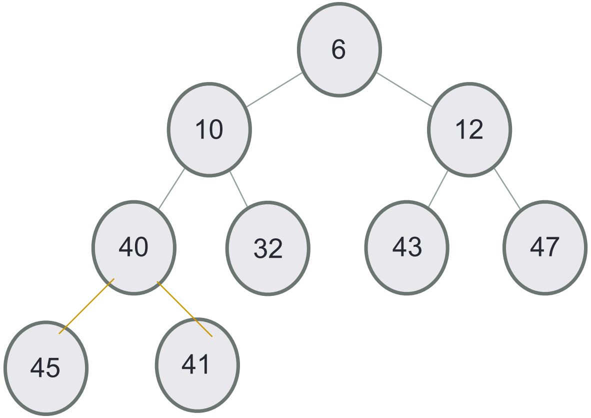
Delete min

- 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



Traversing down the tree

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



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

Next lecture

- **More on STL implementation of heaps (priority queues)**
- **Queues**