HEAPS

Problem Solving with Computers-II

GitHub

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
- What are the operations supported?
- What are the running times?

There are two variants of the Heap datastructure - min-Heap, max-Heap

Each supports the following operations

push () // insert a key

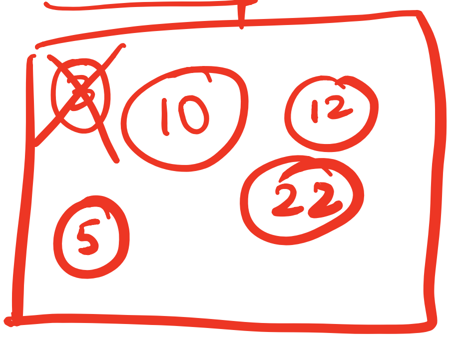
top () // For a min-Heap, top() returns the min key

// For a max-Heap, top() returns the max key

pop () // delete the key on top() (either min or max)

empty() // check if the heap is empty

min-Heap as a black-box



push(5)
push(22)
push(10)
top() // returns 5
push(12)
push(3)
top() // returns 3
pop() // deletes 3
top() // returns 5

Running time for heap with N keys

push() → O(log N) → same as insert in a balanced BST, but in practice heap-push is faster

top() → O(1) → better than a balanced BST

pop() - O(log N) → same as delete min on a BST

Heaps

N keys in each data structure

	Min-Heaps	Max-Heap	BST	balanced BST
Insert :	O(logN)	O(logN)	O(N)	O(logN)
Min:	→ O(1)	—	O(N)	O(logN)
Delete Min:	O(logN)	—	O(N)	O(logN)
Max	—	O(1)	O(N)	O(logN)
Delete Max	—	O(logN)	O(N)	O(logN)

Applications:

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

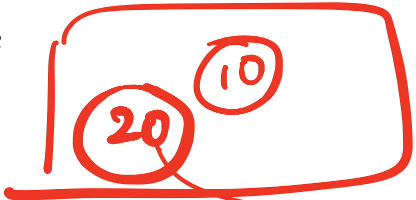
Choose heap if you are doing repeated insert/delete/(min OR max) operations

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.

heap keys

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



push(10)
push(20)
top() // returns 20

Methods:

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

key values are used as priority by default

key with max priority is on "top"

- 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();
```

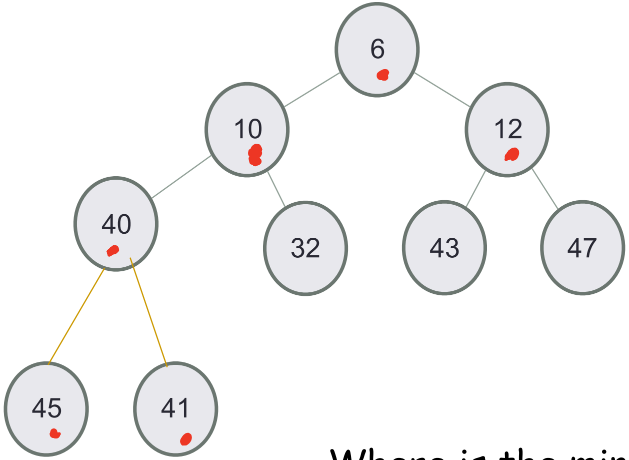
A. 10 2 80
B. 2 10 80
C. 80 10 2
D. 80 2 10
E. None of the above

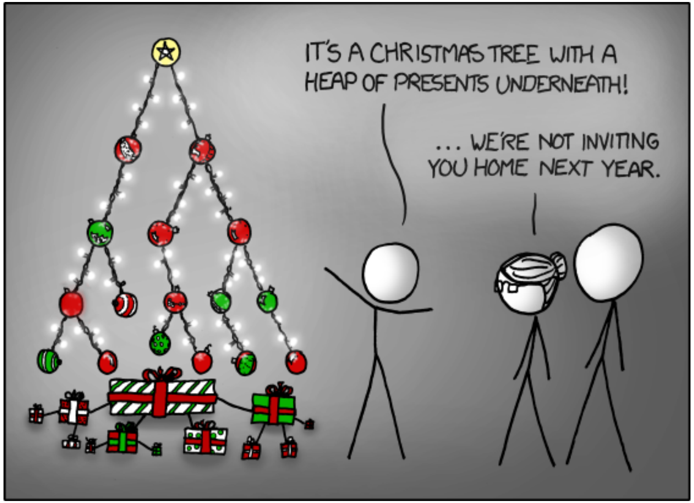
Heaps as binary trees

- 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



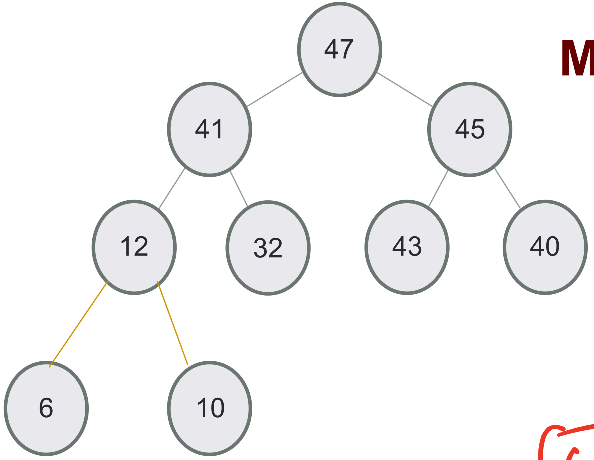


Where is the minimum element?

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)



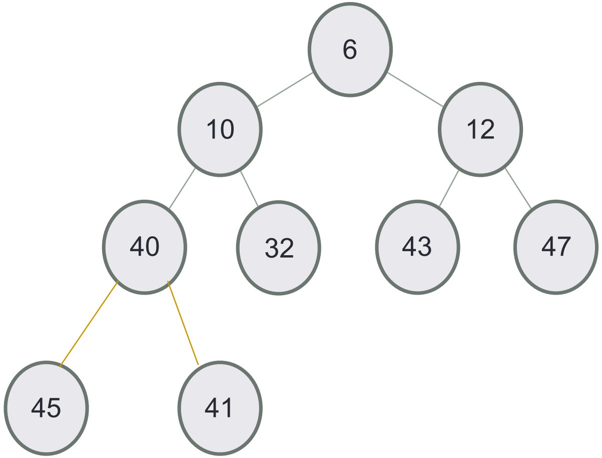
Max Heap with 9 nodes

Where is the maximum element?

50

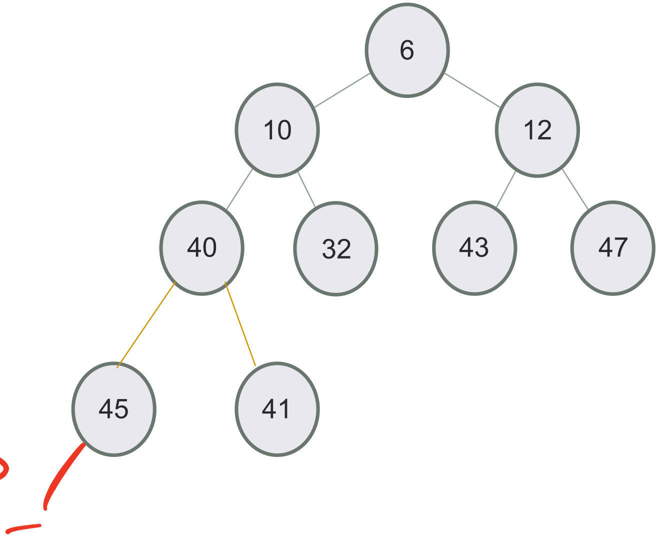
Structure: Complete binary tree

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



Identifying heaps

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

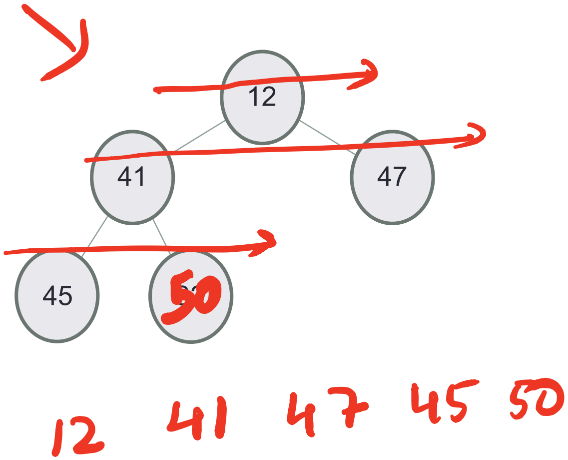


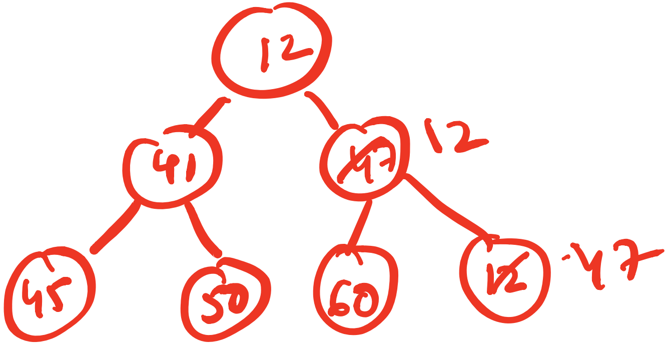
A. **Swap the nodes 40 and 32**
B. **Swap the nodes 32 and 43**
C. **Swap the nodes 43 and 40**
D. **Insert 50 as the left child of 45**
E. **C&D**

Violates the structure requirements

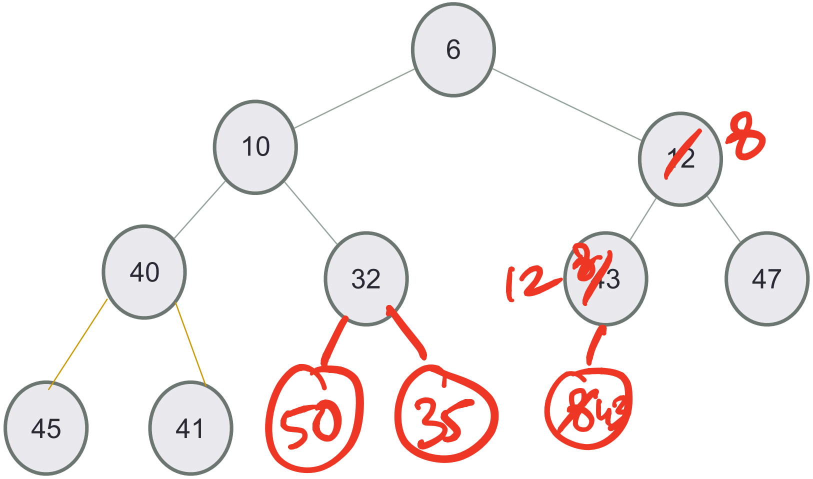
Insert 50 into a heap

- 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))





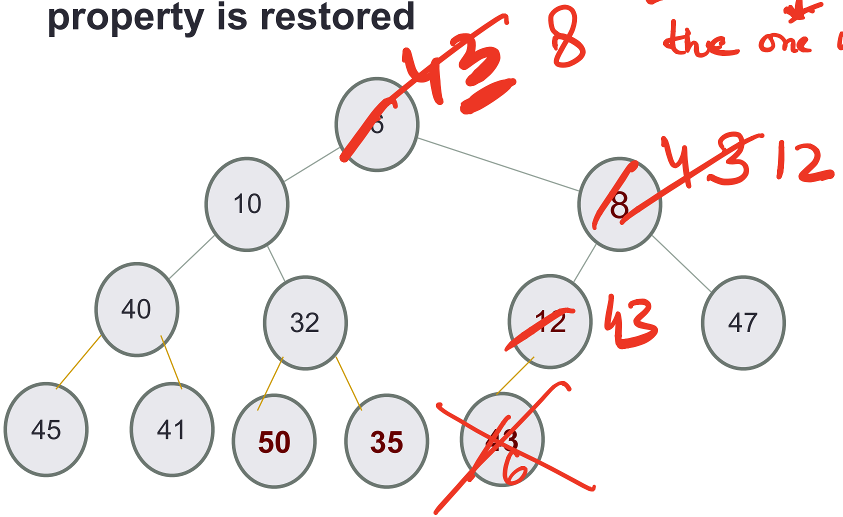
Insert 50, then 35, then 8



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

the one* with the min key



Perform pop() on this tree

swap 6 with 43

Bubble down 43

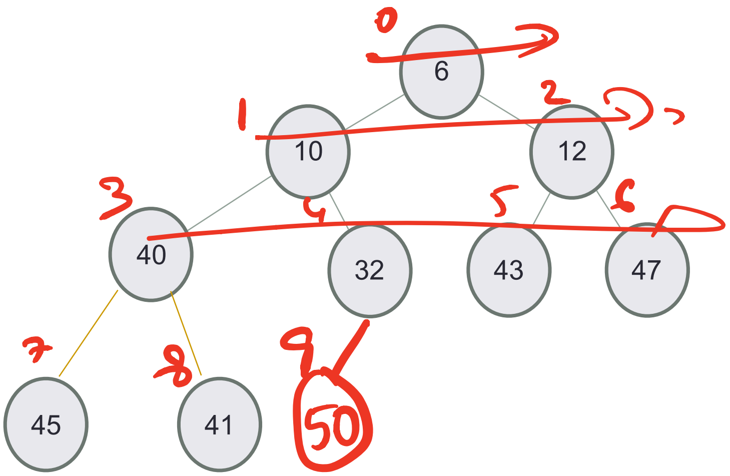
Under the hood of heaps

- 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

"Read out" the keys in the tree level by level, left to right. Start with the root.

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

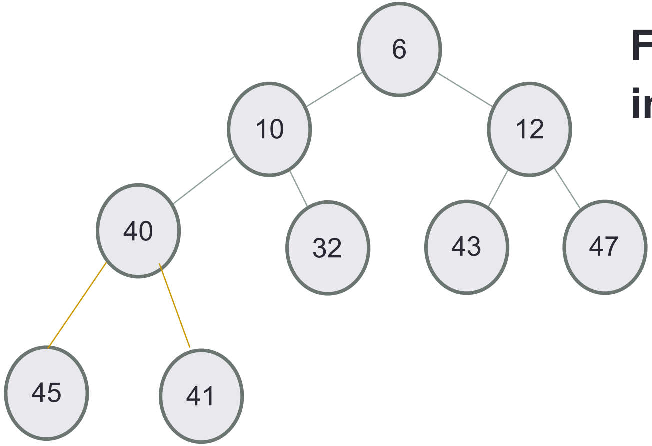


The entire heap-tree shown on the left can be represented as a vector. The parent-child relationships in the vector are implicit (we don't store pointers)

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 ~~node~~ key at index i, index of the parent is int(i-1/2)

In general for a key at index i of the vector, the index of its parent is $int\left(\frac{i-1}{2}\right)$

Value	6	10	12	40	32	43	47	45	41	
Key Index	0	1	2	3	4	5	6	7	8	... i
Index of parent	-	0	0	1	1	2	2	3	3	$int\left(\frac{i-1}{2}\right)$

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

In practice we never implement the binary tree in the usual way with nodes & pointers

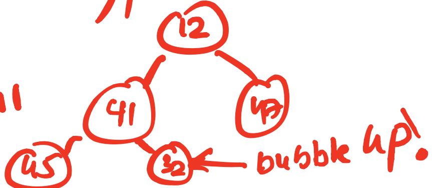
So, we never convert heap-tree to a vector instead work directly with the vector representation

32 > parent(32) stop bubbling up!

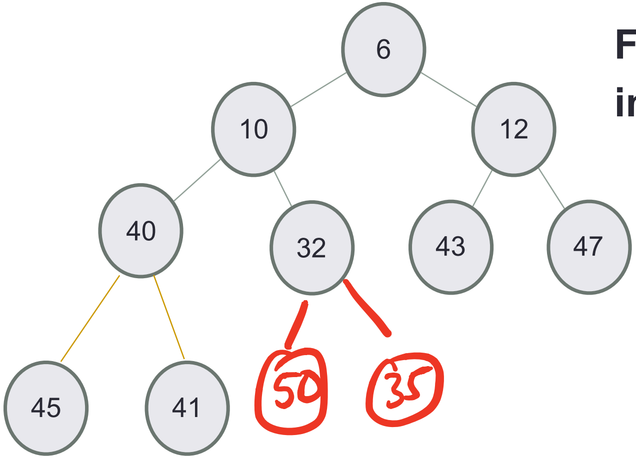
12	~~41~~ 32	47	45	~~32~~ 41
0	1	2	3	4

32 is at index 4
its parent is at index 1 (41)
swap 32 with 41

Hypothetical tree



Insert 50, then 35



For a node at index i, index of the parent is int(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	40	32	43	47	45	41	50	35	8
Index	0	1	2	3	4	5	6	7	8	9	10	11

Insert 8 at index 11 (8 is now the last node in the last level of the "tree")

Find 8's parent at index $(\frac{11-1}{2}) = 5$, → key 43

$8 < 43$ so swap, 8 & 43

Find 8's new parent at index $(\frac{5-1}{2}) = 2$ → key 12

swap 8 & 12

Find 8's new parent at index $(\frac{2-1}{2}) = 0$, key 6

$8 > 6$ → stop

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

A. Node 6

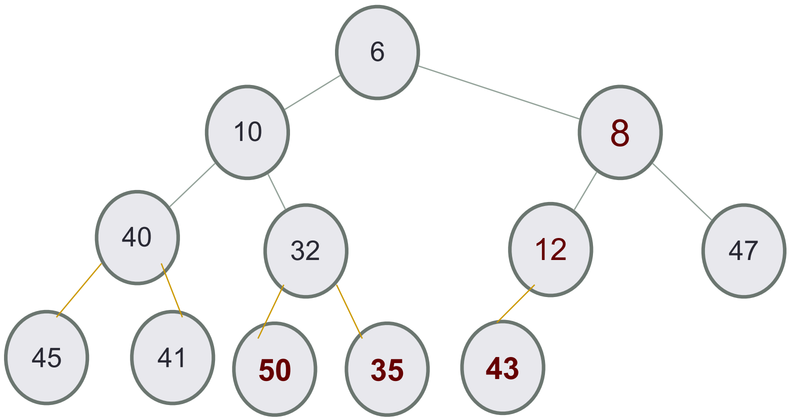
B. Node 12

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**

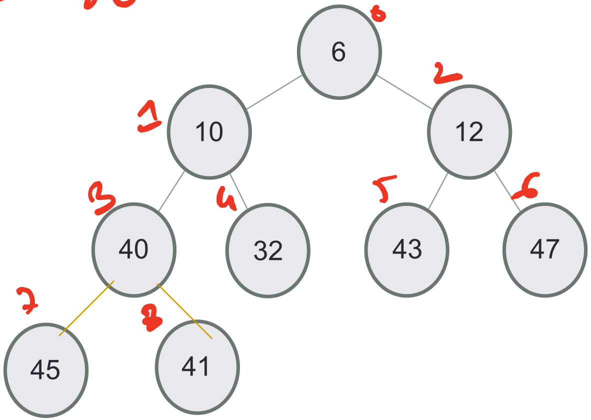


Traversing down the tree

When doing a pop, replace the min key (6) with the last key in the vector (41), Delete 6, by reducing the size of the vector by 1. Bubble down 41

Value	6	10	12	40	32	43	47	45	41	
Index	0	1	2	3	4	5	6	7	8	
Index of left child	1	3	5	7	-	-	-	-	-	
Index of right child	2	4	6	8	-	-	-	-	-	

to do this need to find the children in the vector.



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**