PRIORITY QUEUES (HEAPS)

Problem Solving with Computers-II

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.

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

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

std::priority_queue template arguments

```
template <
      class T,
      class Container= vector<T>,
      class Compare = less <T>
      > class priority_queue;
```

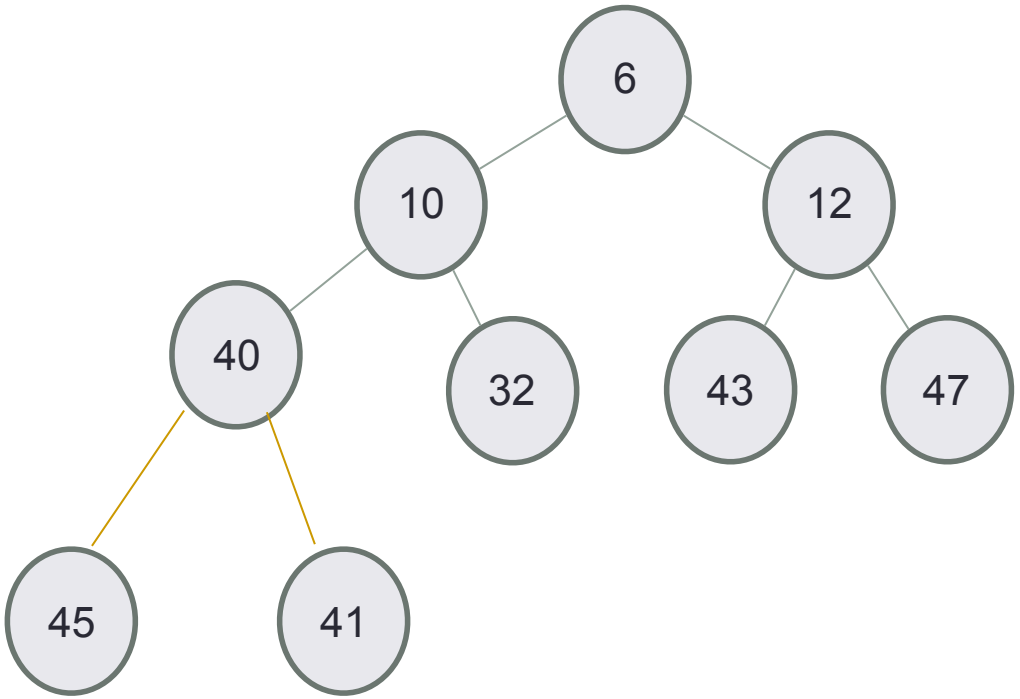
The template for priority_queue takes 3 arguments:

1. Type elements contained in the queue.
2. Container class used as the internal store for the priority_queue, the default is **vector<T>**
3. Class that provides priority comparisons, the default is **less**

```
priority_queue<int, vector<int>, std::greater<int>> pq;
```

Implementing heaps using array or vector

Value									
Index	0	1	2	3	4	5	6	7	8



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**

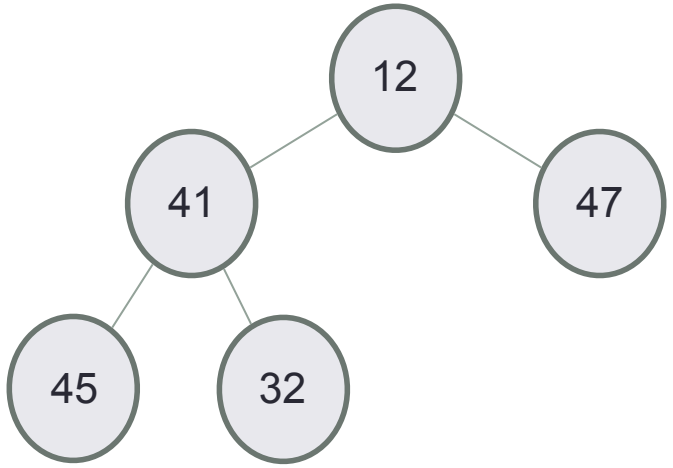
Insert into a heap

- **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….**

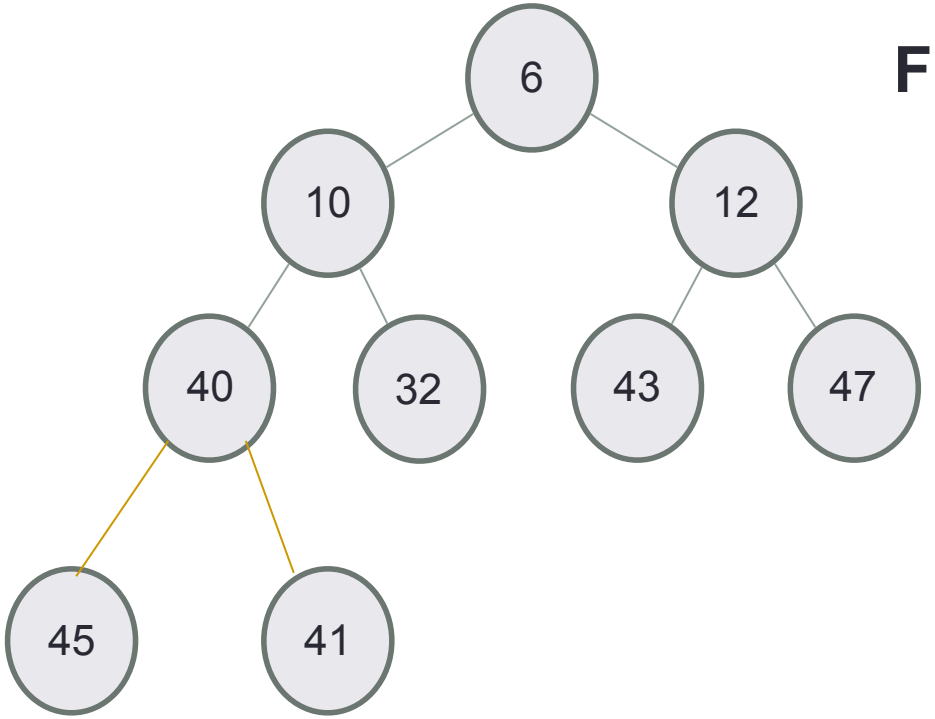
Insert the elements {12, 41, 47, 45, 32} in a min-Heap

Insert 32 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))**



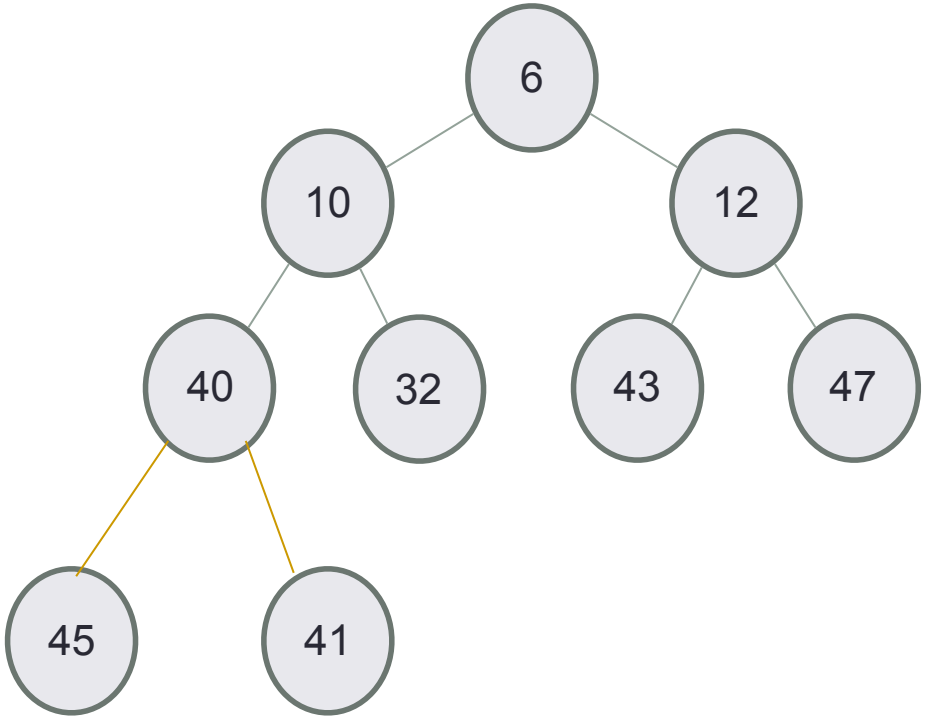
Traversing up the “tree”



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

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

Insert 50, then 35



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

Insert 8 into a heap

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

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

A. Node 6

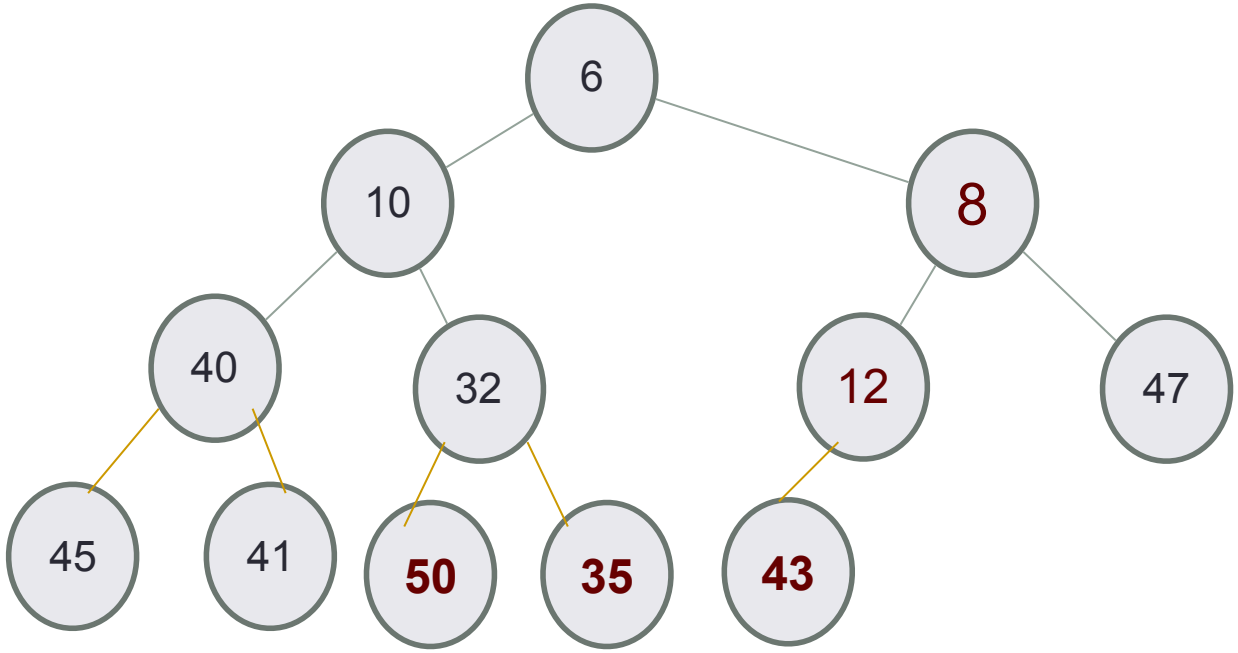
B. Node 12

C. None 43

D. None - Node 8 will be the root

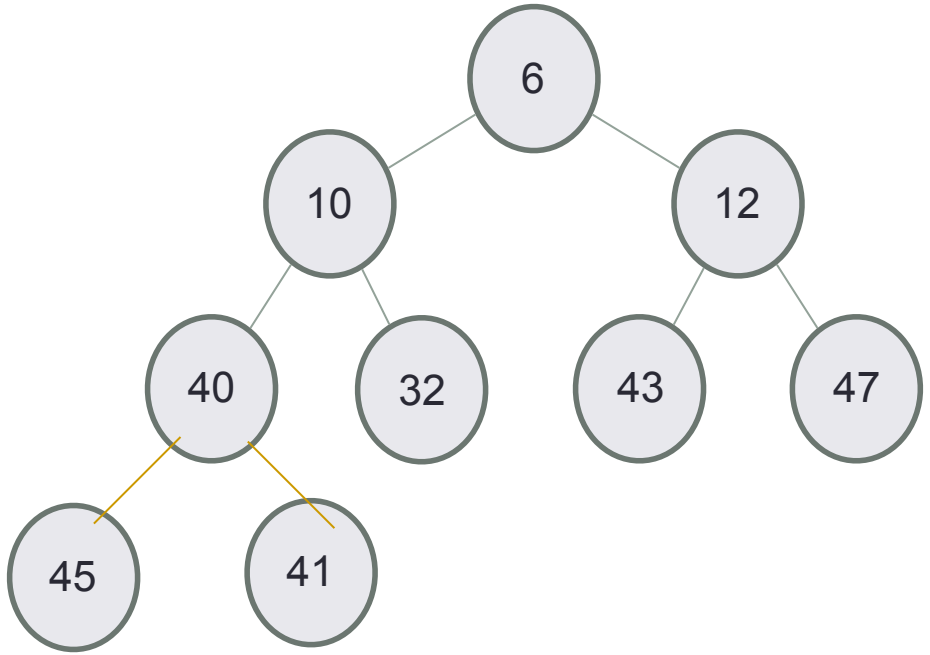
Delete min

- **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

std::priority_queue template arguments

```
//Define a max-heap
priority_queue<int, vector<int>, std::less<int>> pq;
```

But what is the third template parameter really?

But what is the third template parameter really?

```
priority_queue<int, vector<int>, std::less<int>> pq;

 template <class T>
 class less{
        bool operator()(T& a, T & b) const {
              return a<b;
         }
 };

less<int> ls;
if(ls(a,b))
  cout<<a<<“is less than ”<< b;
```

The default std::less is a comparator class that provides priority comparisons