

HEAPS

Problem Solving with Computers-II

C++

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

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

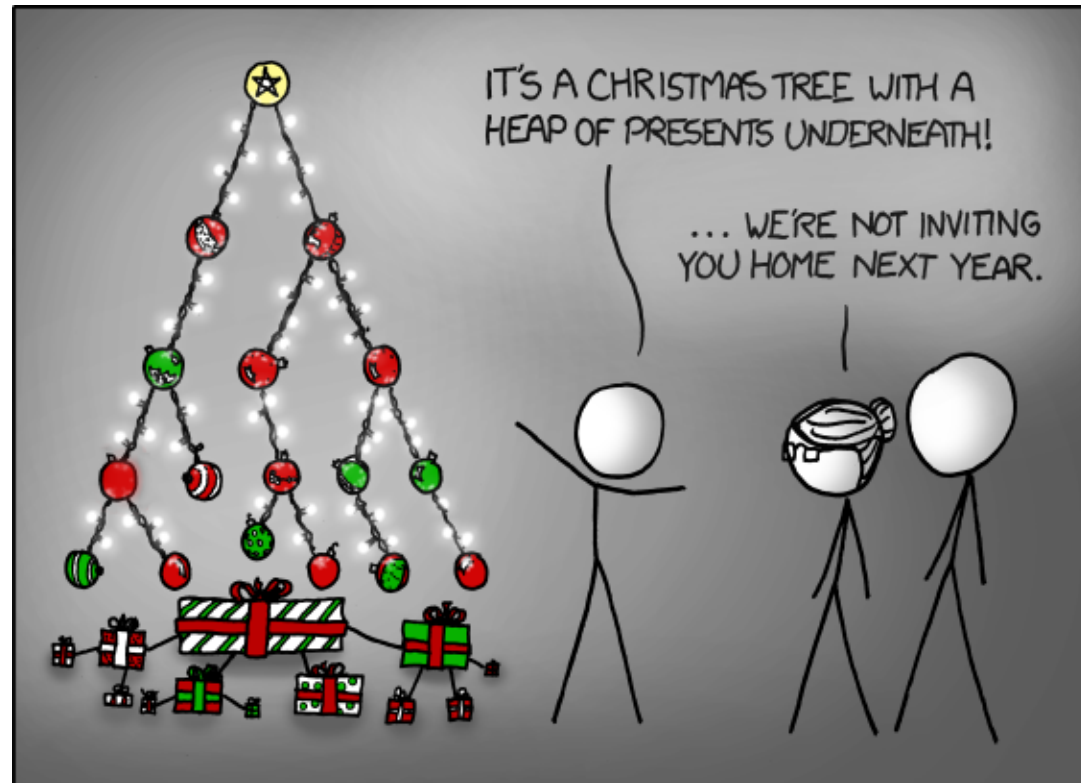


Reminders

- PA02 released, due Friday of Week 10 .
- Lab07 due Wed of Week 10
- zyBook, Chapter 8 activities due Monday of Finals week

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?



Heaps

Min-Heaps

Max-Heap

BST

- Insert :
- Min:
- Delete Min:
- Max
- Delete Max

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.

```
#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
- * `size()` //returns the number of elements in the PQ
- 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

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**

std::priority_queue template arguments

//Template parameters for a max-heap

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

//Template parameters for a min-heap

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


Comparison class

- Comparison class: A class that implements a function operator for comparing objects

```
class compareClass{  
    bool operator()(int& a, int & b) const {  
        return a>b;  
    }  
};
```

Comparison class

```
class compareClass{  
    bool operator()(int& a, int & b) const {  
        return a>b;  
    }  
};
```

```
int main(){  
    compareClass c;  
    cout<<c(10, 20)<<endl;  
}
```

What is the output of this code?

A. 1

B. 0

C. Error

STL Heap implementation: Priority Queues in C++

```
class cmp{
    bool operator()(int& a, int & b) const {
        return a>b;
    }
};

priority_queue<int, vector<int>, cmp> 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();
```

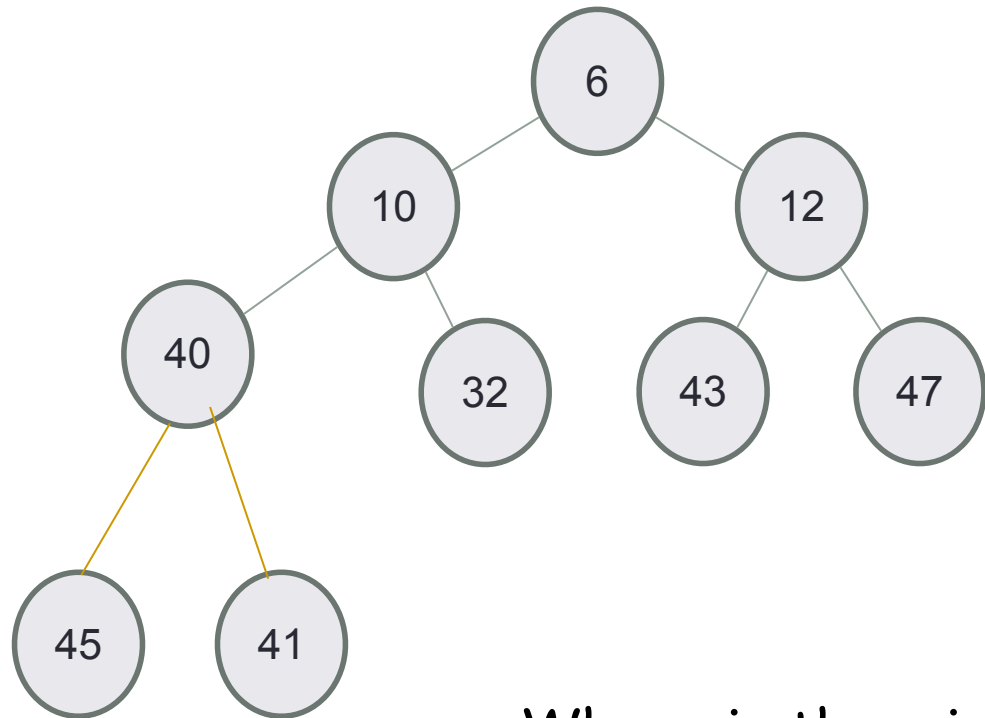
Output: _____

pq is a _____heap

Heaps as binary trees

- Rooted binary tree that is as complete as possible
- In a **min-Heap**, each node satisfies the following **heap property**:
 $\text{key}(x) \leq \text{key}(\text{children of } x)$

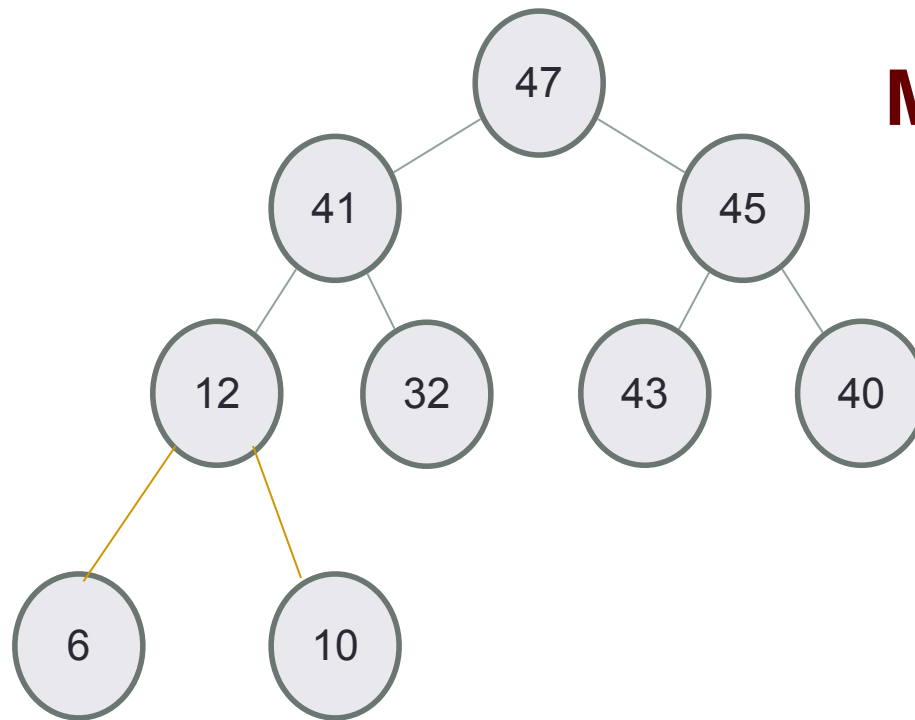
Min Heap with 9 nodes



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**:
 $\text{key}(x) \geq \text{key}(\text{children of } x)$

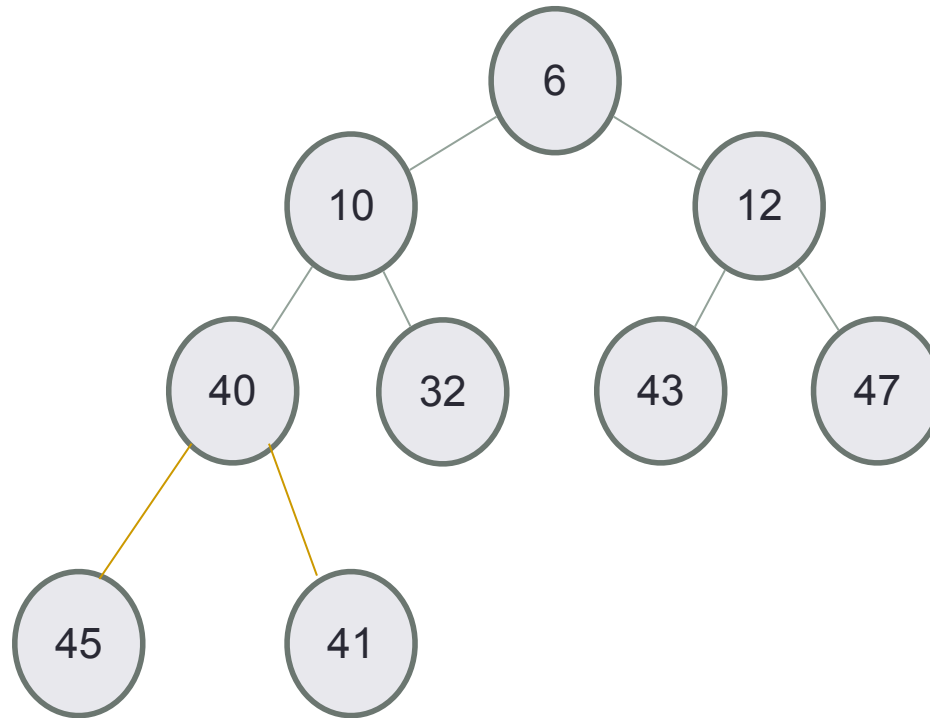


Max Heap with 9 nodes

Where is the maximum element?

Structure: Complete binary tree

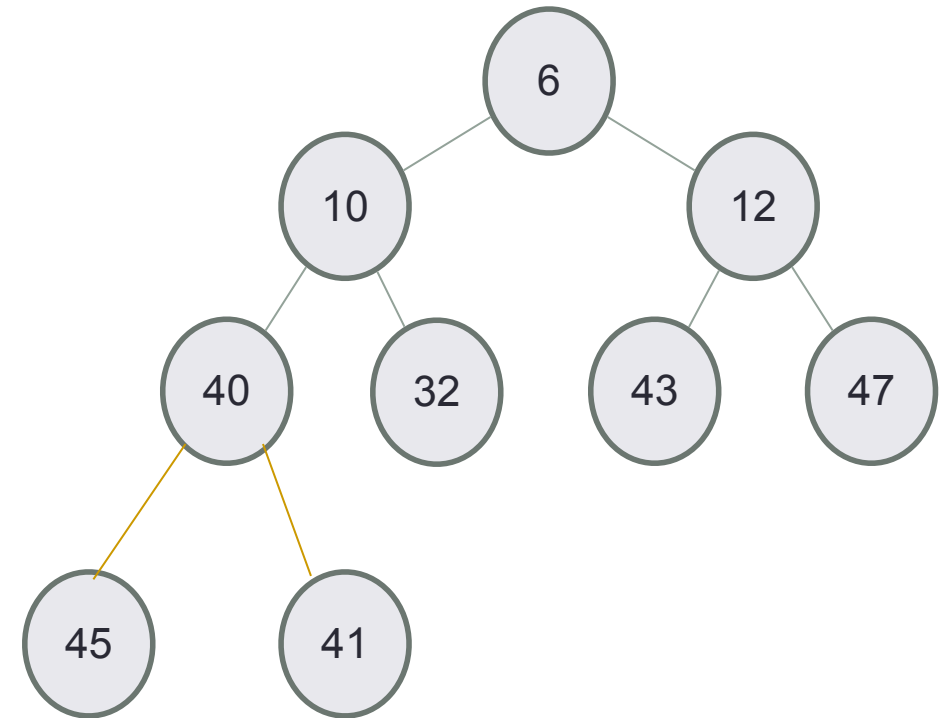
**A heap is a complete binary tree: Each level is as full as possible.
Nodes on the bottom level are placed as far left as possible**



Identifying heaps

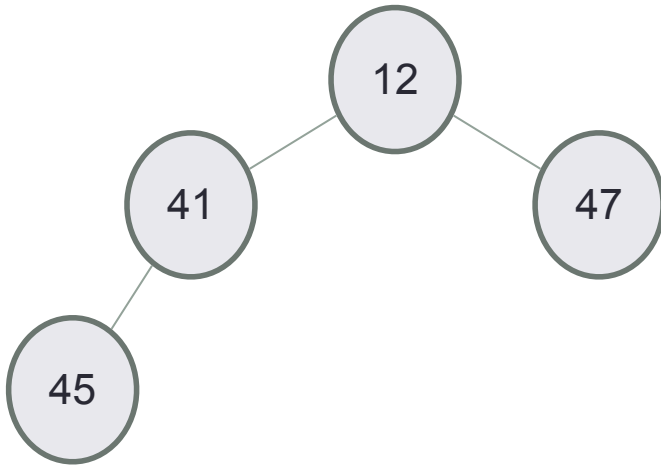
Starting with the following min-Heap which of the following operations will result in something that is NOT a min Heap

- 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

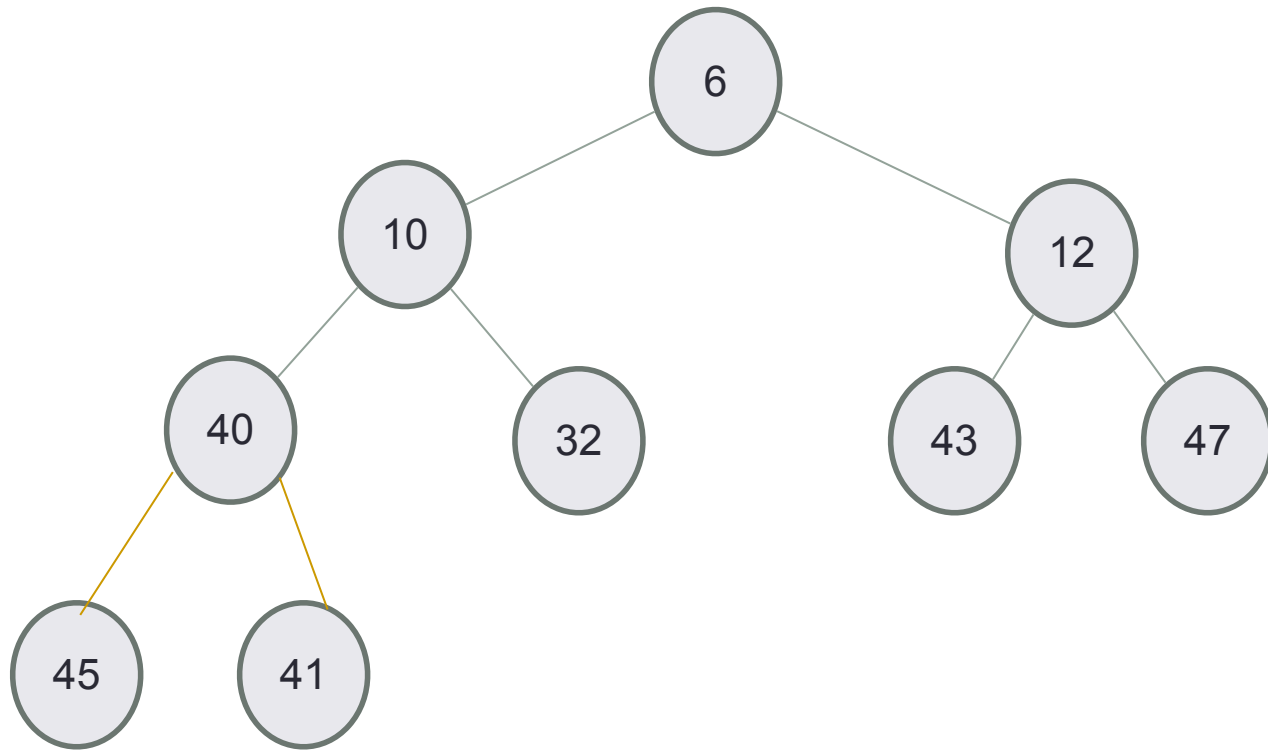


Insert 50 into a heap

- Insert $\text{key}(x)$ 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: $\text{while}(\text{key}(\text{parent}(x)) > \text{key}(x))$ swap the $\text{key}(x)$ with $\text{key}(\text{parent}(x))$

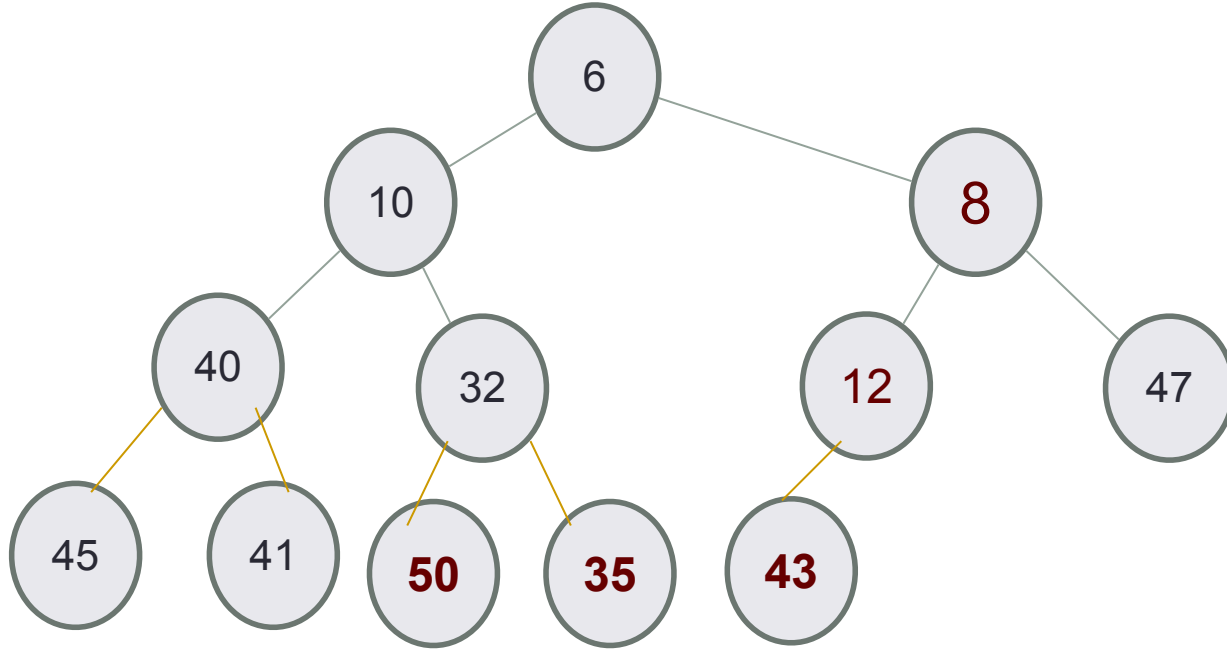


Insert 50, then 35, then 8



Delete min

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

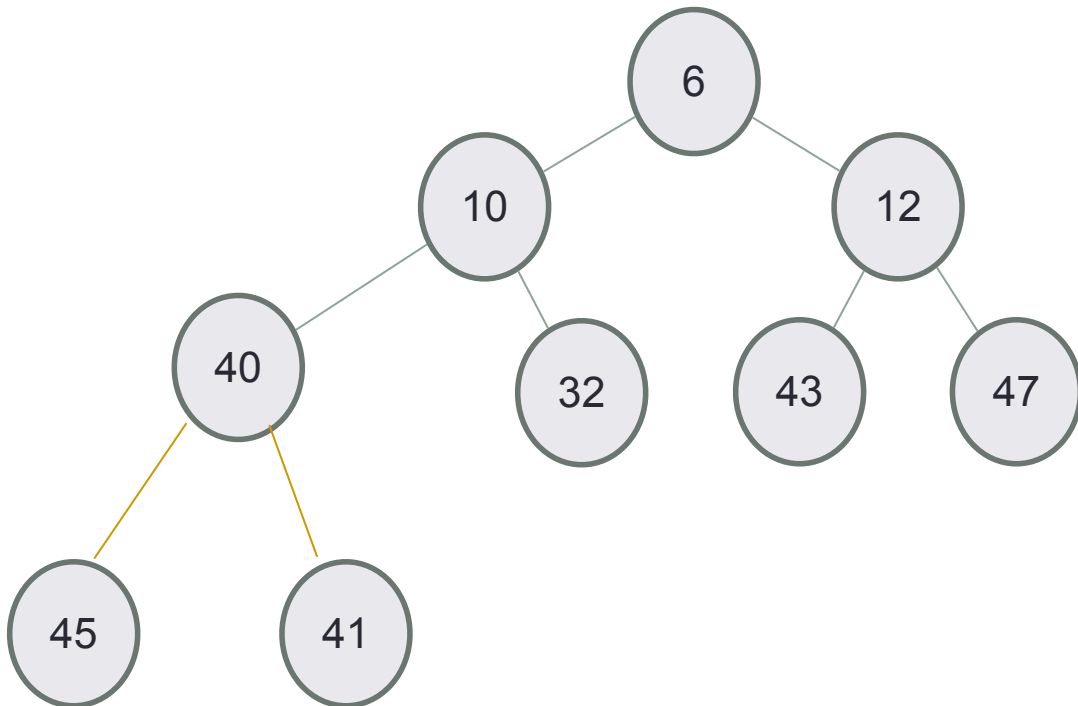


Under the hood of heaps

- 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										
Index	0	1	2	3	4	5	6	7	8	9

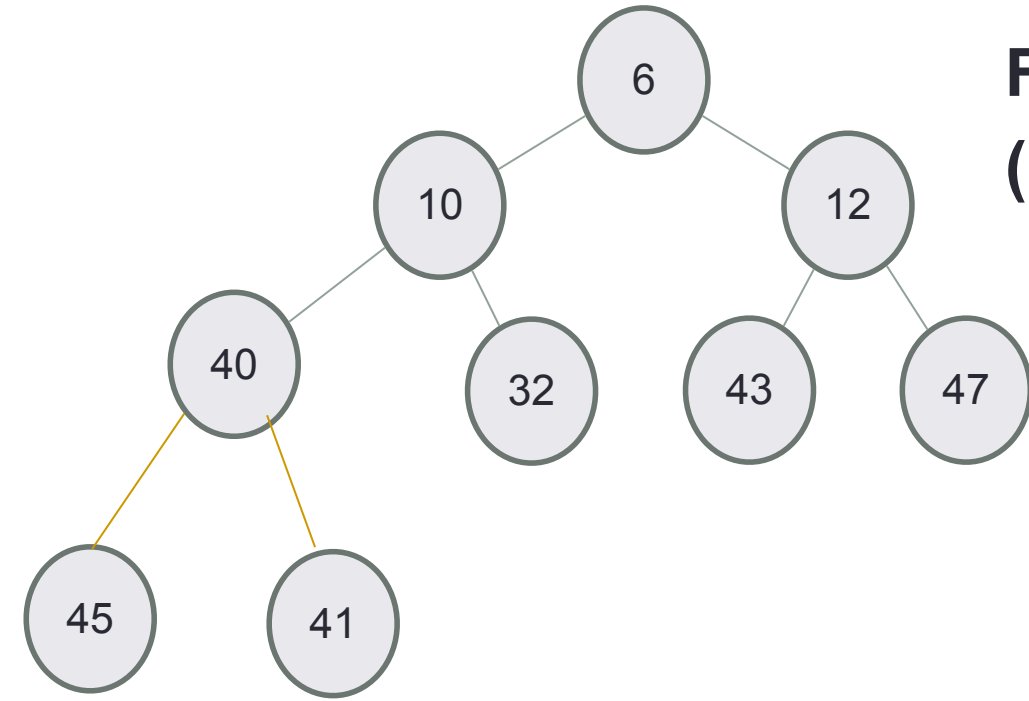


Using vector as the internal data structure of the heap has some advantages:

- 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 $(i-1)/2$



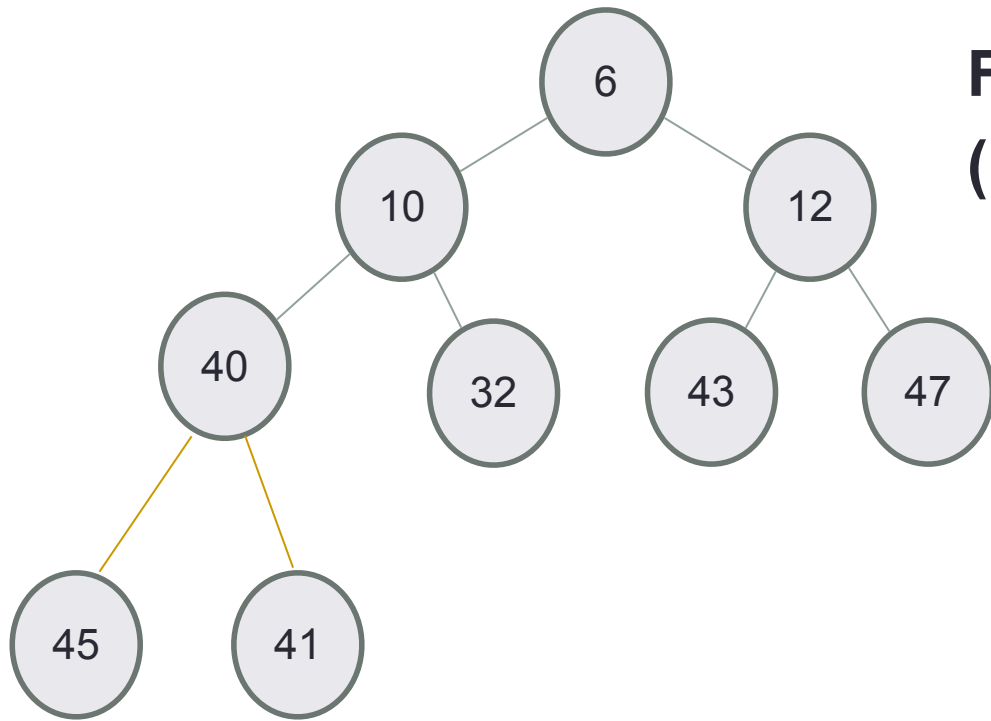
Value	6	10	12	40	32	43	47	45	41	
Index	0	1	2	3	4	5	6	7	8	

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

Insert 50, then 35

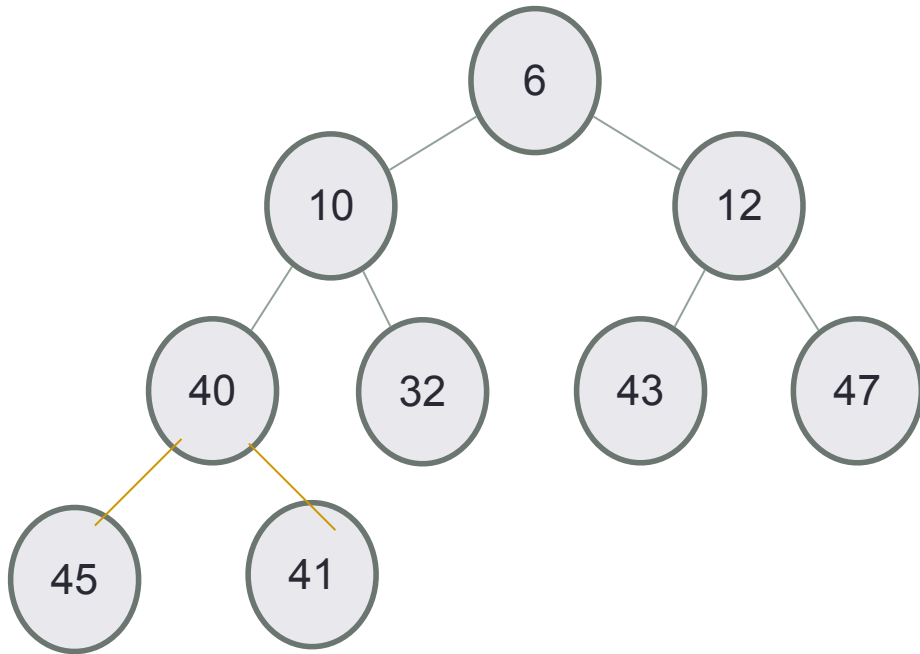


For a node at index i , index of the parent is $(i-1)/2$

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

Traversing down the tree

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

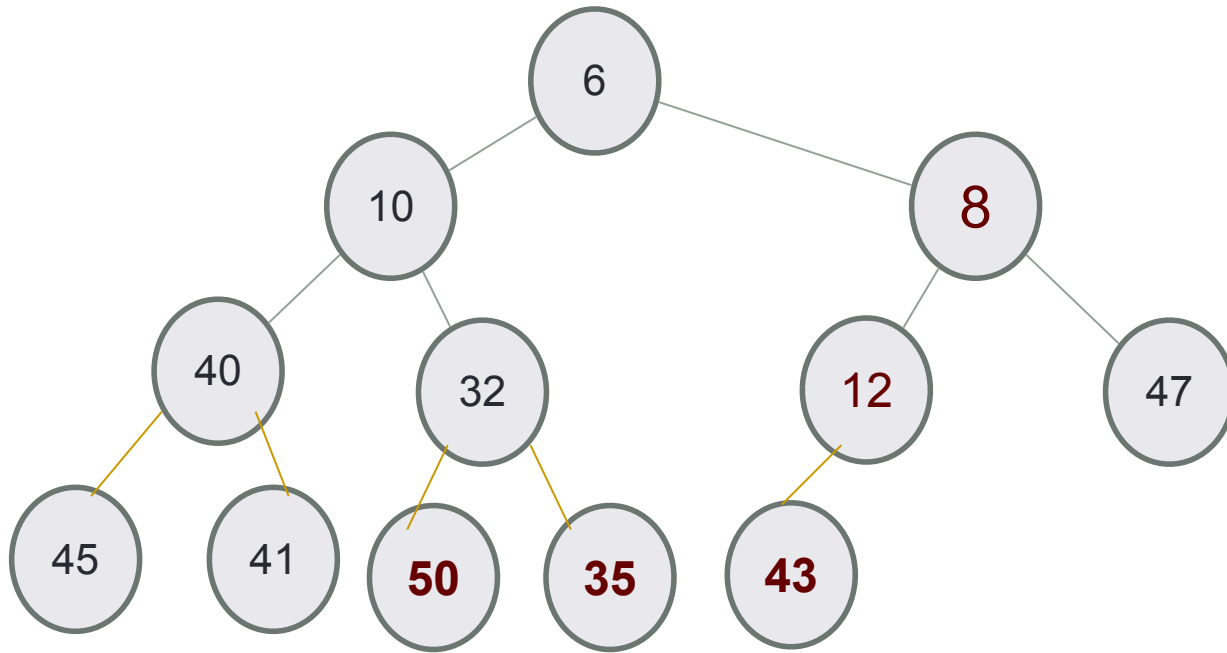


For a node at index i , what is the index of the left and right children?

- A. $(2*i, 2*i+1)$
- B. $(2*i+1, 2*i+2)$
- C. $(\log(i), \log(i)+1)$
- D. None of the above

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



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**