Heap

Problem: Clinic Waiting Room

You have to simulate a clinic waiting room. There is only one Doctor available. Patients will come in the Clinic. Doctor might have to prioritize patients based on the severity of their need. For example, doctor will choose to see the next "most critical" patient rather than the one who arrived first.

Problem: Clinic Waiting Room

Create a menu based system. Option 1 will add the patient in the Clinic waiting room. Option 2 will treat the most critical patient. Option 3 will view all the patients in the Waiting Room.

```
1. Add a Patient
2. Treat the most critical Patient
3. View all the Patients in the Waiting Room
4. Exit
Option->
```

Problem: Clinic Waiting Room

Option 1 will add the patient in the Clinic waiting room. We are just storing the name of the patient and his/her emergency Level.

```
Option-> 1
Enter the Patient Name: ABC
Enter the Patient Emergency Level: 5
Press Any Key to Continue
```

Problem: Clinic Waiting Room

Option 2 will treat the most critical patient in the Clinic waiting room.

```
Option-> 2
TTT with Emergency Level 9 has been treated
Press Any Key to Continue
```

Problem: Clinic Waiting Room

Option 3 will view all the patients in the Clinic waiting room. Patients can be viewed in any order.

```
Option-> 3
Index    Patient Name    EmergencyLevel
1        TTT             9
2        AAA             8
3        MMM             7
4        XYZ             3
5        QQQ             4
6        YYY             2
7        ABC             5
Press Any Key to Continue
```

Problem: Clinic Waiting Room

How will you implement the Solution?

Solution: Clinic Waiting Room

Let's create the structure of Patient.

```
struct Patient
{
    string name;
    int eLevel;
};
```

Solution 1: Clinic Waiting Room

One solution is to store the patients in an array or linked list for option 1, then search the largest element from the array or linked list and remove that element for option 2.

Solution 1: Clinic Waiting Room

One solution is to store the patients in an array or linked list for option 1, then search the largest element from the array or linked list and remove that element for option 2.

Such array or linked list is called Priority Queue.

Solution 1: Time Complexity

One solution is to store the patients in an array or linked list for option 1, then search the largest element from the array or linked list and remove that element for option 2.

Implementation	Insert	Delete
Array	O(1)	O(n)
Linked List	O(1)	O(n)

Solution 2: Clinic Waiting Room

Another solution is to sort the input based on the Emergency Level when a new patient is added, then remove the first patient from the array or linked list.

Solution 2: Clinic Waiting Room

Another solution is to sort the input based on the Emergency Level when a new patient is added, then remove the first patient from the array or linked list.

Such array or linked list is called Priority Queue.

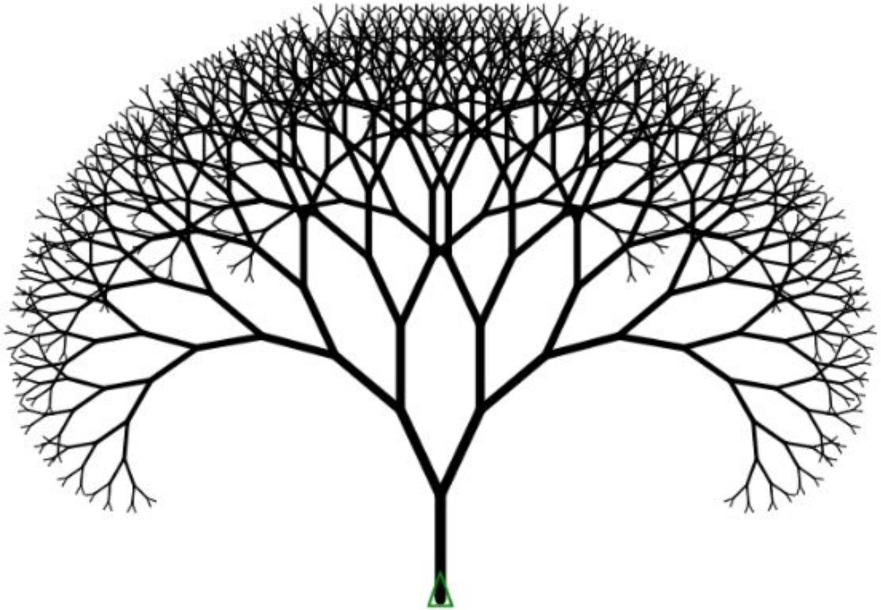
Solution 2: Time Complexity

Another solution is to sort the input based on the Emergency Level when a new patient is added, then remove the first patient from the array or linked list.

Implementation	Insert	Delete
Array	O(n)	O(1)
Linked List	O(n)	O(1)

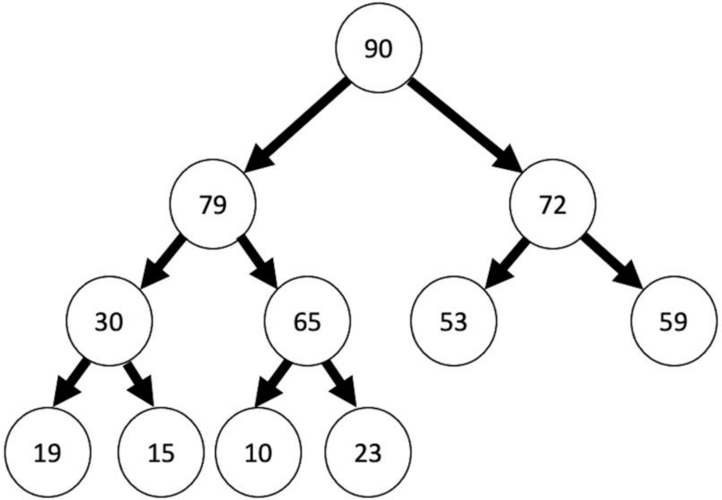
Solution 3: Clinic Waiting Room

The most efficient method for implementing the priority queue is through Binary Trees.



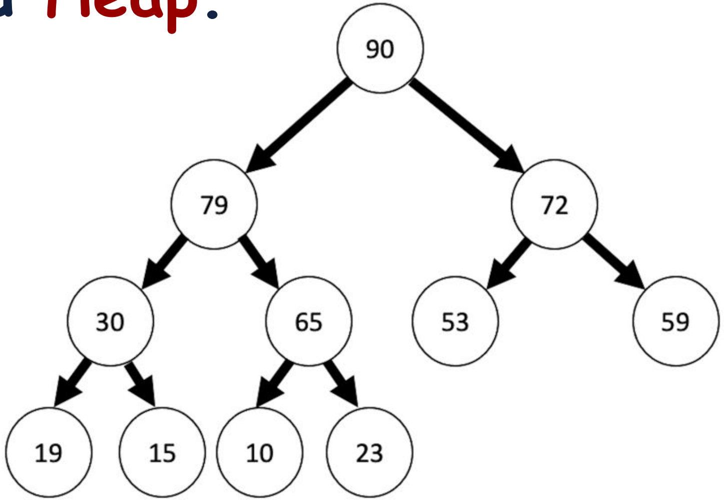
Solution 3: Binary Tree

The specific order of the binary tree to be used as the priority queue is such that the value of the node is greater than both of its children.



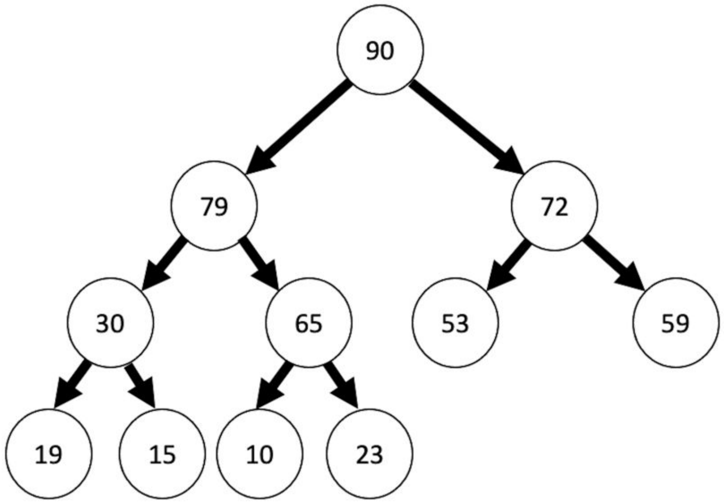
Heap

The specific order of the binary tree to be used as the priority queue is such that the value of the node is greater than both of its children. This is a new Data Structure called Heap.



Heap

The Heap in which the value of the node is greater than both of its children is more specifically called as **Max Heap**.



Heap

Now, let's see how to create the max heap if the patients with different priorities came.

Heap

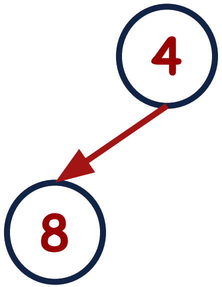
Input: 4,

Make 4 as root node.

Heap

Input: 4, 8,

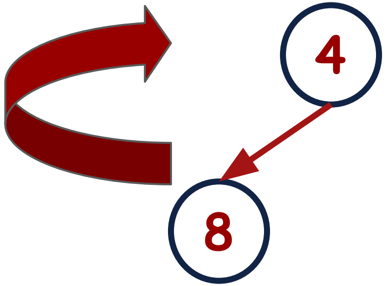
Place 8 where the next space is free.



Heap

Input: 4, 8,

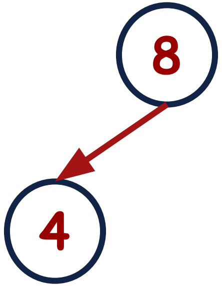
If the value is greater than the parent node then swap them.



Heap

Input: 4, 8,

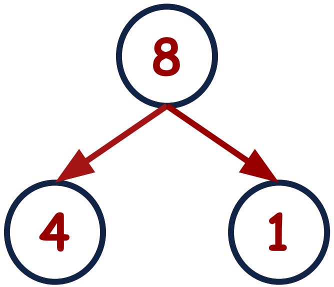
If the value is greater than the parent node then swap them.



Heap

Input: 4, 8, 1,

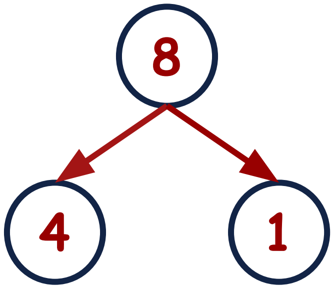
Place 1 where the next space is free.



Heap

Input: 4, 8, 1,

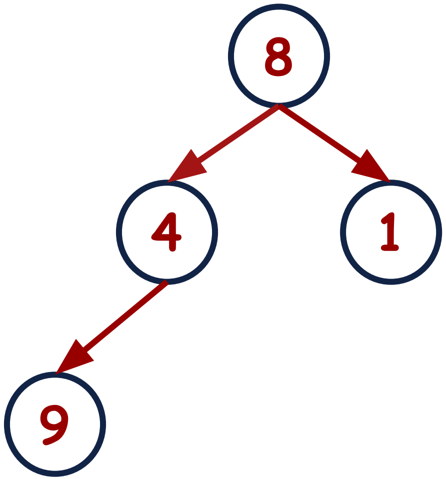
If the value is smaller than the parent node then do nothing.



Heap

Input: 4, 8, 1, 9,

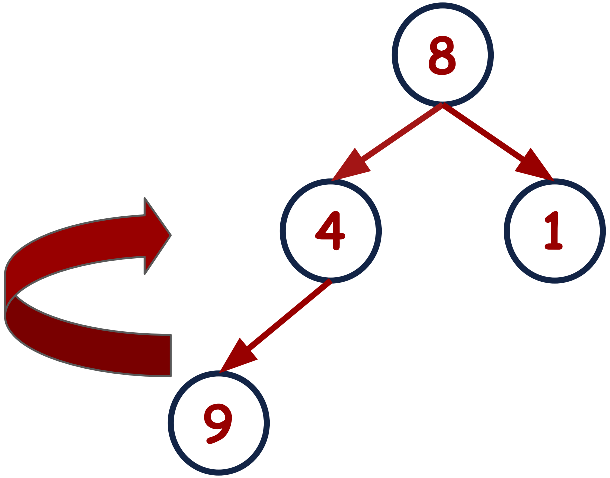
Place 9 where the next space is free.



Heap

Input: 4, 8, 1, 9,

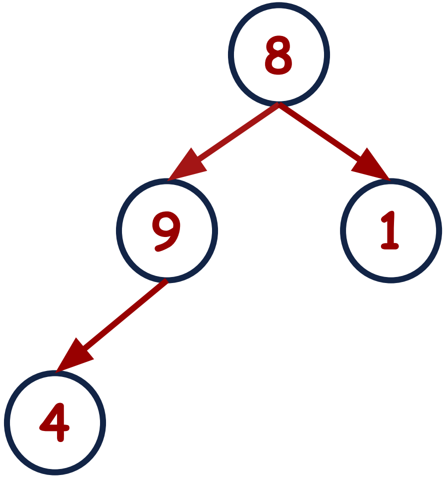
If the value is greater than the parent node then swap them.



Heap

Input: 4, 8, 1, 9,

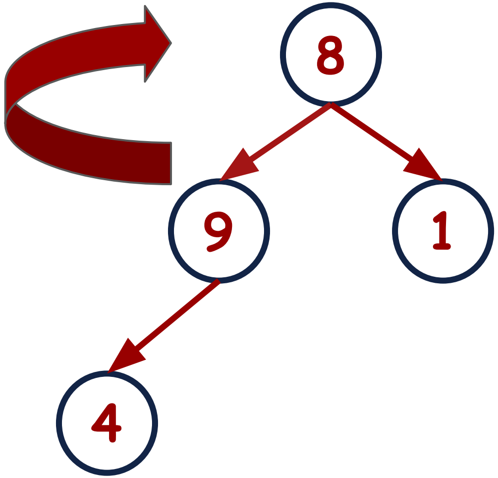
If the value is greater than the parent node then swap them.



Heap

Input: 4, 8, 1, 9,

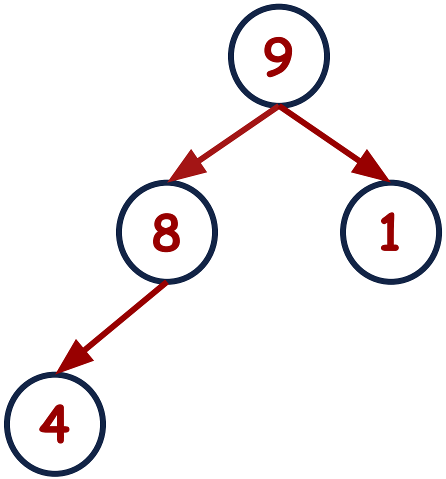
If the value is greater than the parent node then swap them.



Heap

Input: 4, 8, 1, 9,

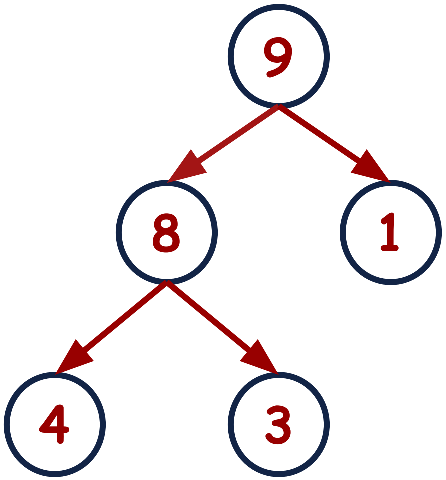
If the value is greater than the parent node then swap them.



Heap

Input: 4, 8, 1, 9, 3

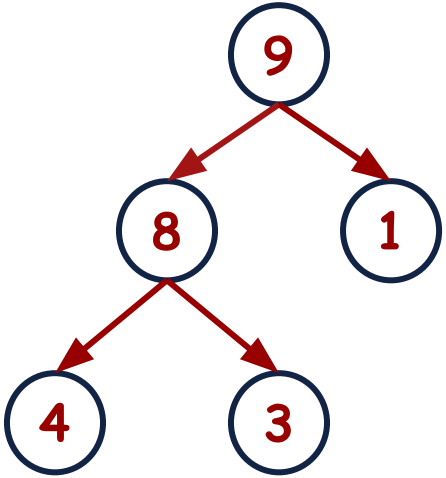
Place 3 where the next space is free.



Heap

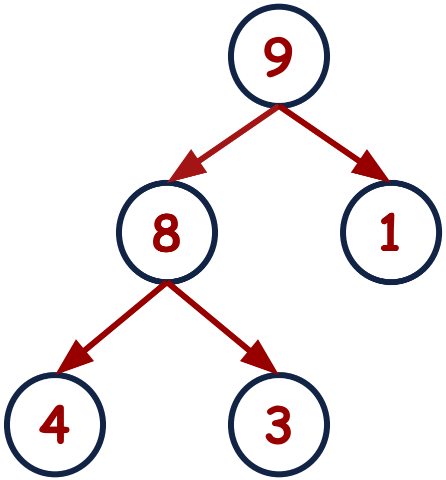
Input: 4, 8, 1, 9, 3

If the value is smaller than the parent node then do nothing.



Heap

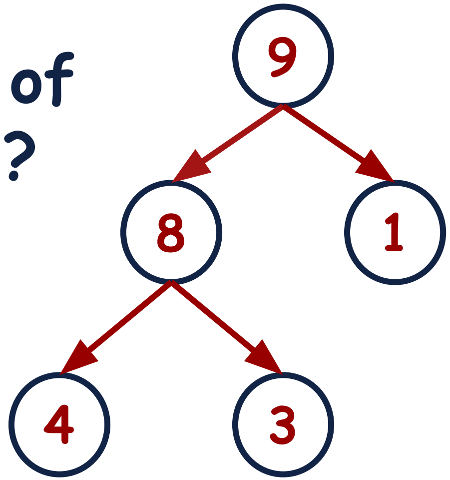
This process of creating the heap is called heapify.



Heap: Food for Thought

Note that all levels are completely filled except possibly the last level and the last level has all elements as left as possible.

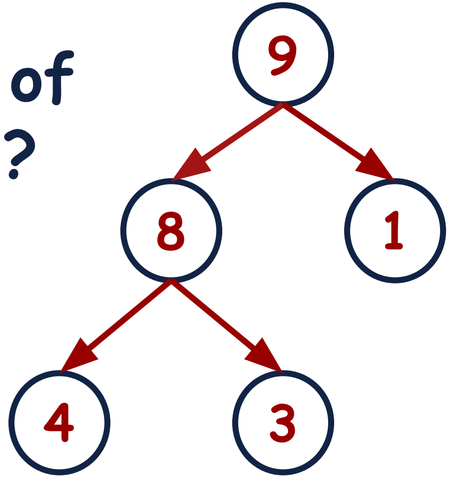
What is this type of Binary Tree called?



Heap: Complete Binary Tree

Note that all levels are completely filled except possibly the last level and the last level has all elements as left as possible.

What is this type of Binary Tree called?



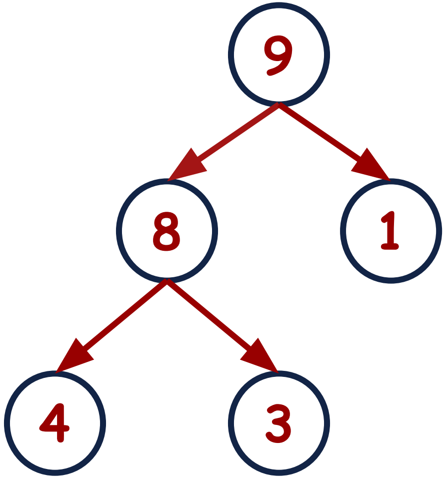
Heap (Insertion): Time Complexity

What is the Time Complexity of insertion in the Heap?

Implementation	Insert	Delete
Binary Heap	O(log(n))	

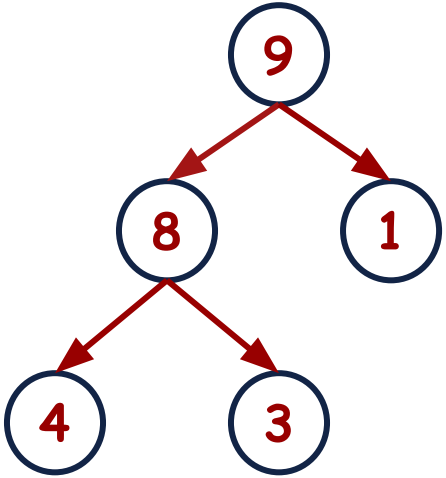
Heap: Deletion

Now, let's see how to delete the element with the highest priority.



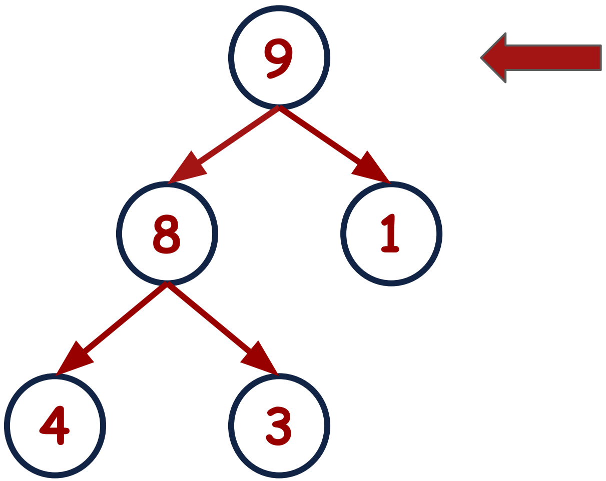
Heap: Deletion

Where is the element with highest priority present?



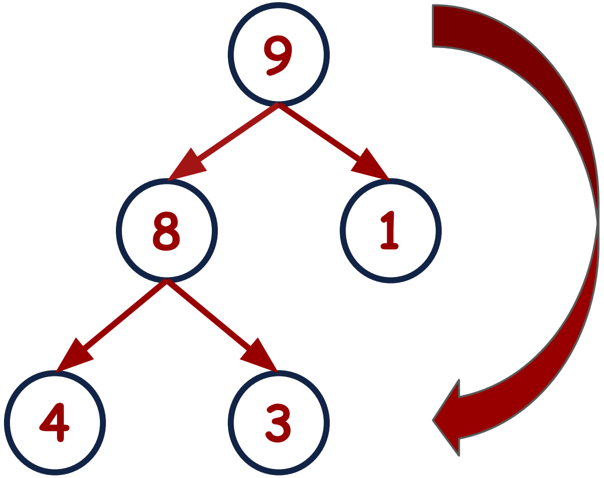
Heap: Deletion

Highest priority element is present on the root node.



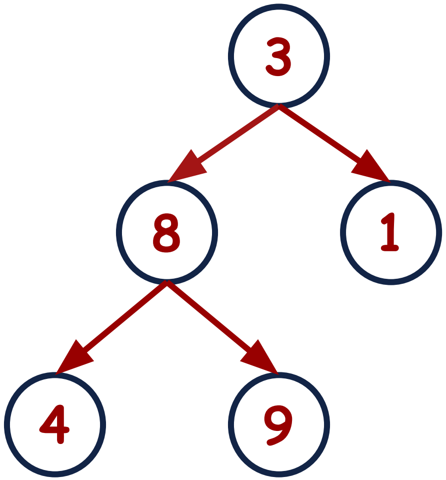
Heap: Deletion

Swap the value of the root node with the last element in the heap.



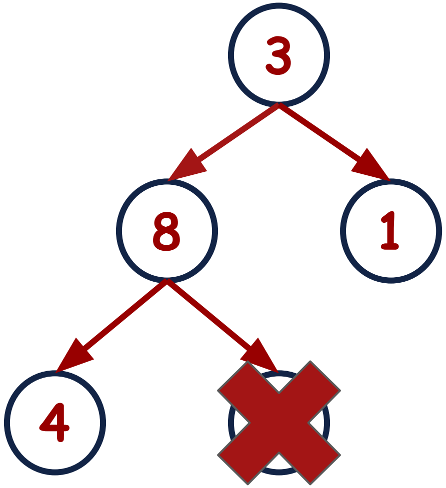
Heap: Deletion

Swap the value of the root node with the last element in the heap.



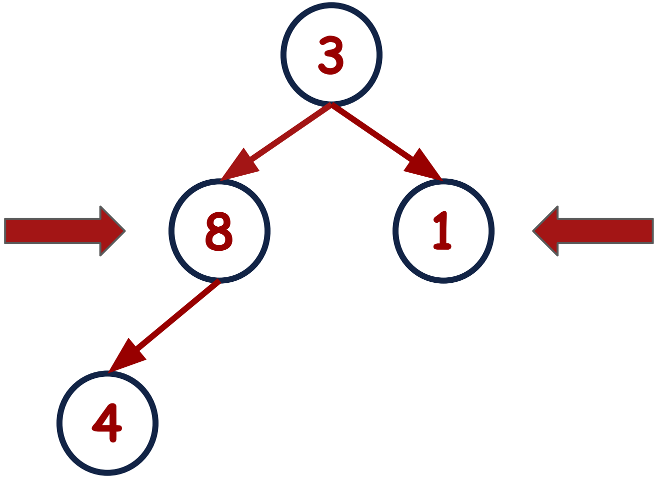
Heap: Deletion

Delete the last node.



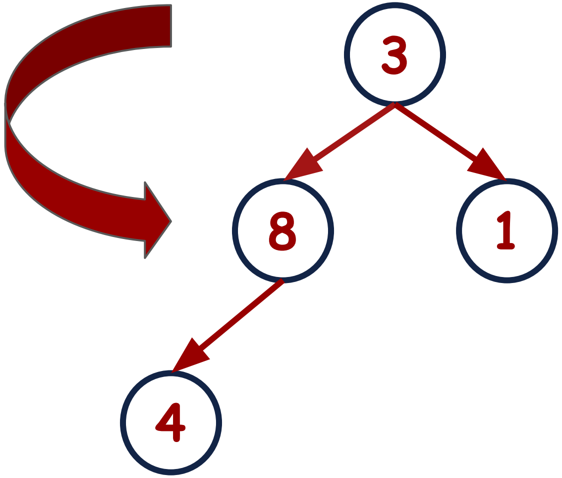
Heap: Deletion

Compare the root node value with its children.



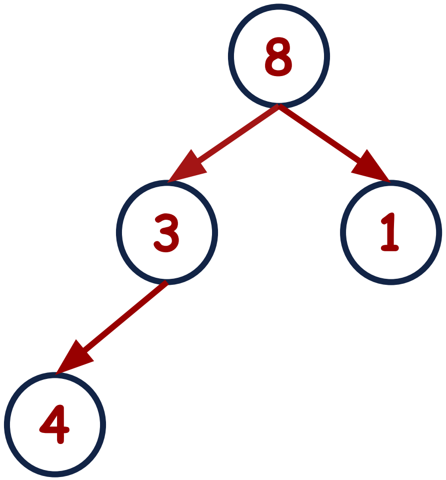
Heap: Deletion

Swap with the largest value from its children.



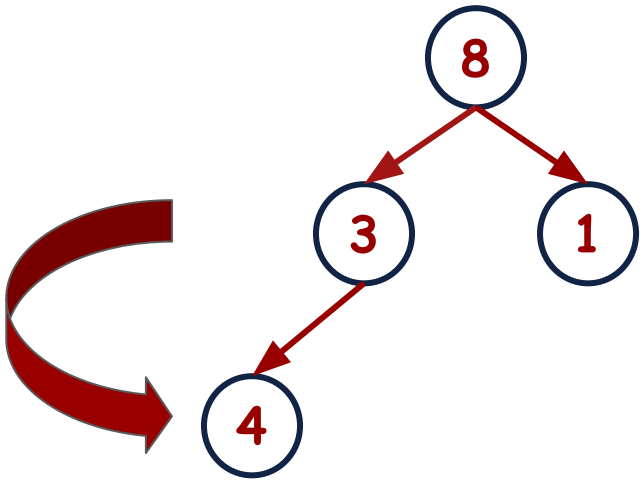
Heap: Deletion

Swap with the largest value from its children.



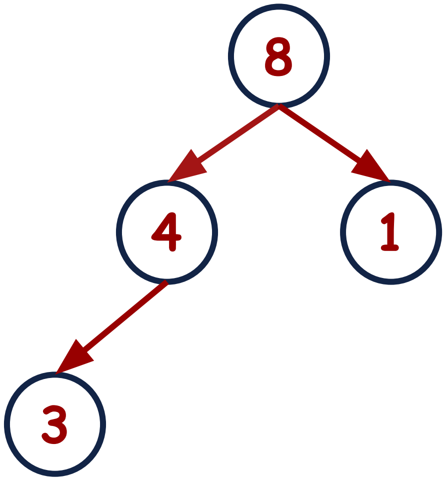
Heap: Deletion

Swap with the largest value from its children.



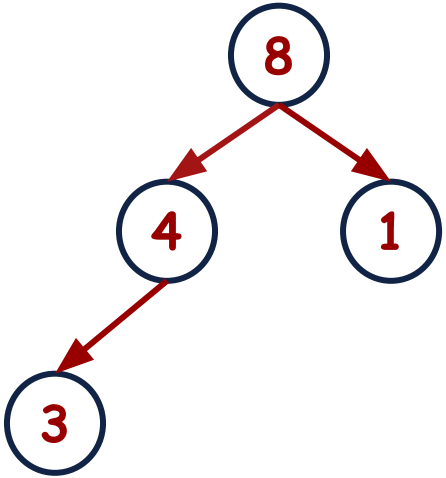
Heap: Deletion

Swap with the largest value from its children.



Heap: Deletion

After the deletion process, tree will become Max Heap again.



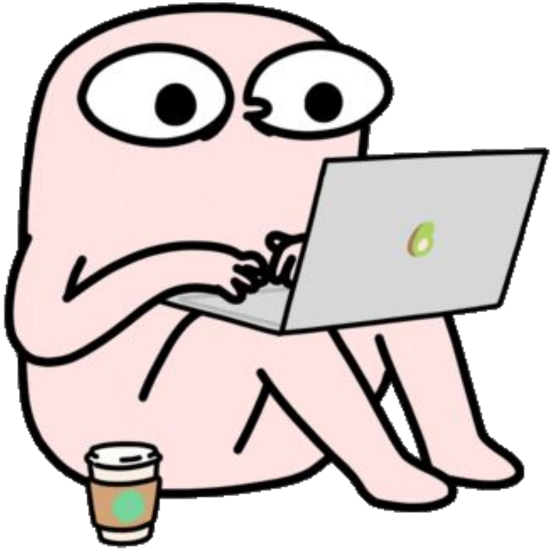
Heap (Deletion): Time Complexity

What is the Time Complexity of Deletion in the Heap?

Implementation	Insert	Delete
Binary Heap	O(log(n))	O(log(n))

Heap: Implementation

Let's implement the Heap.



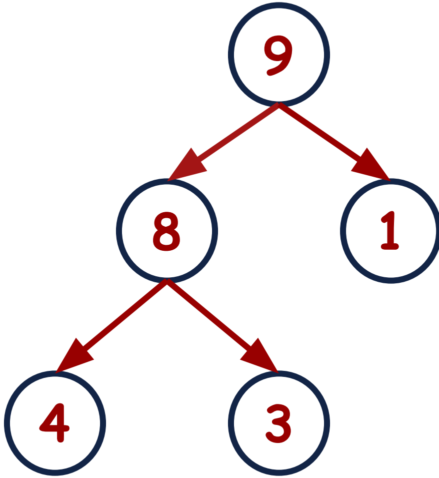
Heap: Implementation

A Binary Heap is a Complete Binary Tree. This property of Binary Heap makes it suitable to be stored in an array or vector.

Heap: Implementation

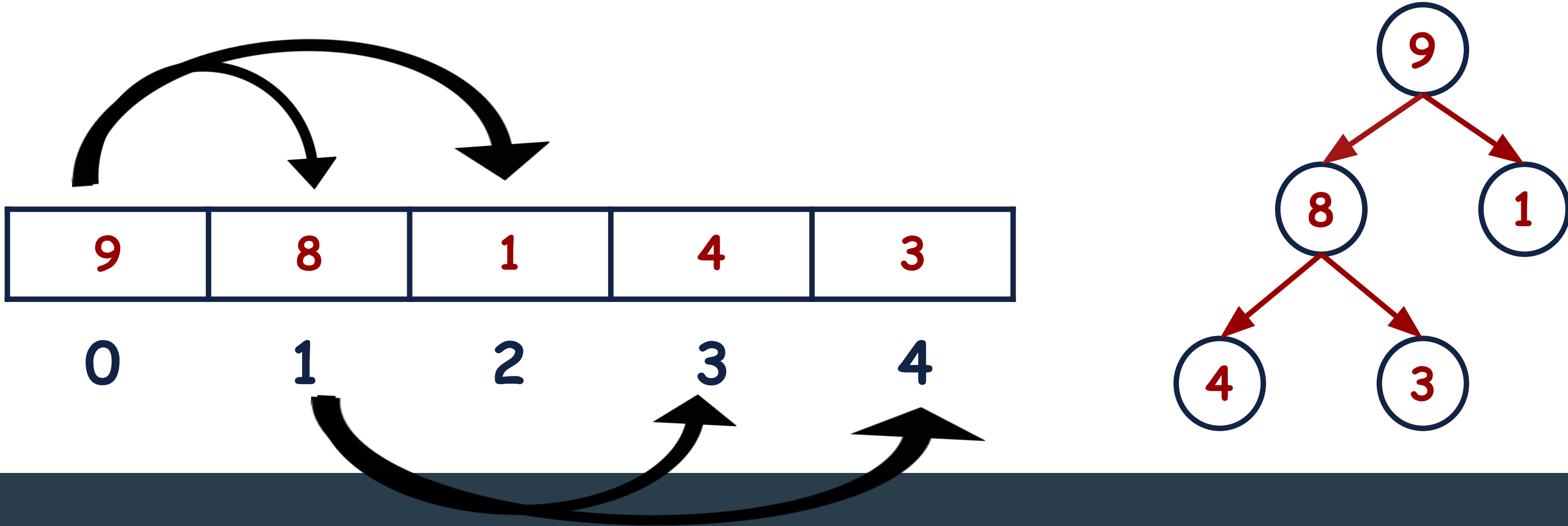
A Binary Heap is a Complete Binary Tree. This property of Binary Heap makes it suitable to be stored in an array or vector.

9	8	1	4	3
0	1	2	3	4



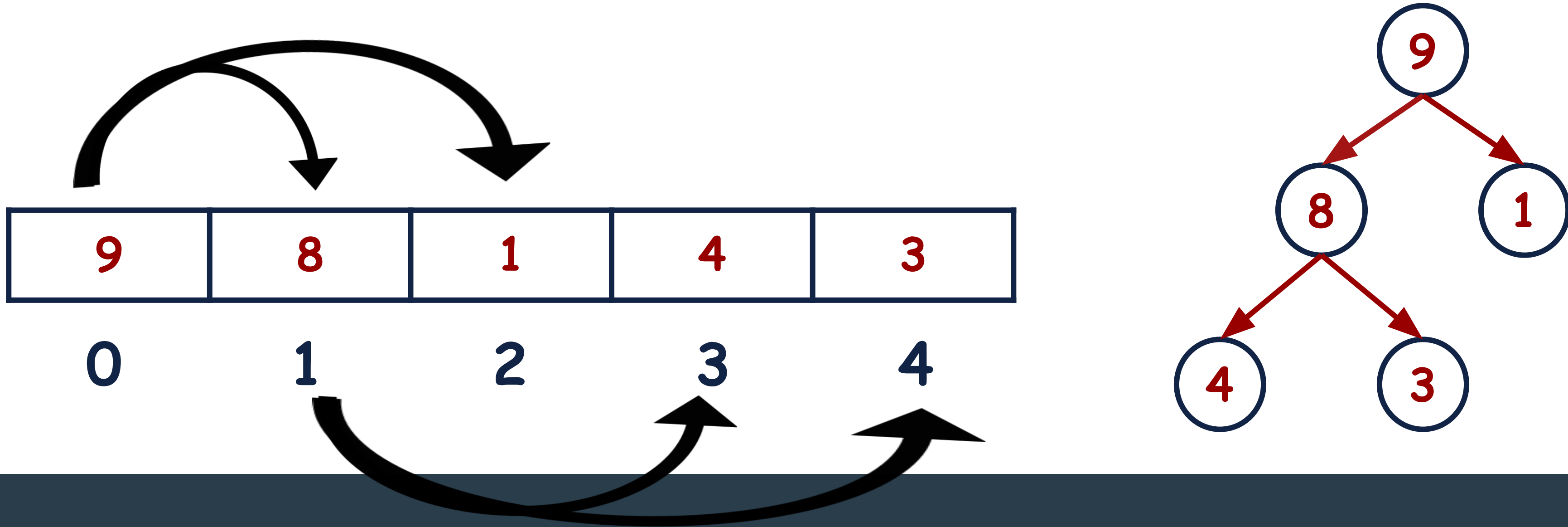
Heap: Implementation

A Binary Heap is a Complete Binary Tree. This property of Binary Heap makes it suitable to be stored in an array or vector.



Heap: Implementation

General Formulas: Let i be the current Index.



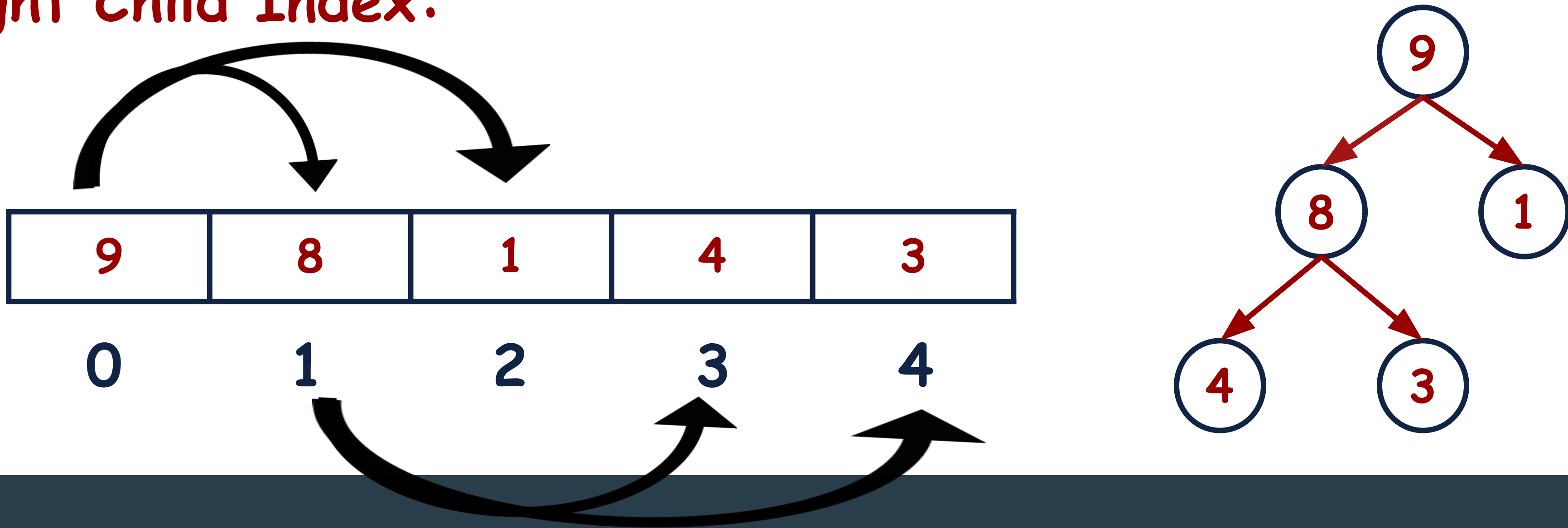
Heap: Implementation

General Formulas: Let i be the current Index.

Parent Index:

Left Child Index:

Right Child Index:



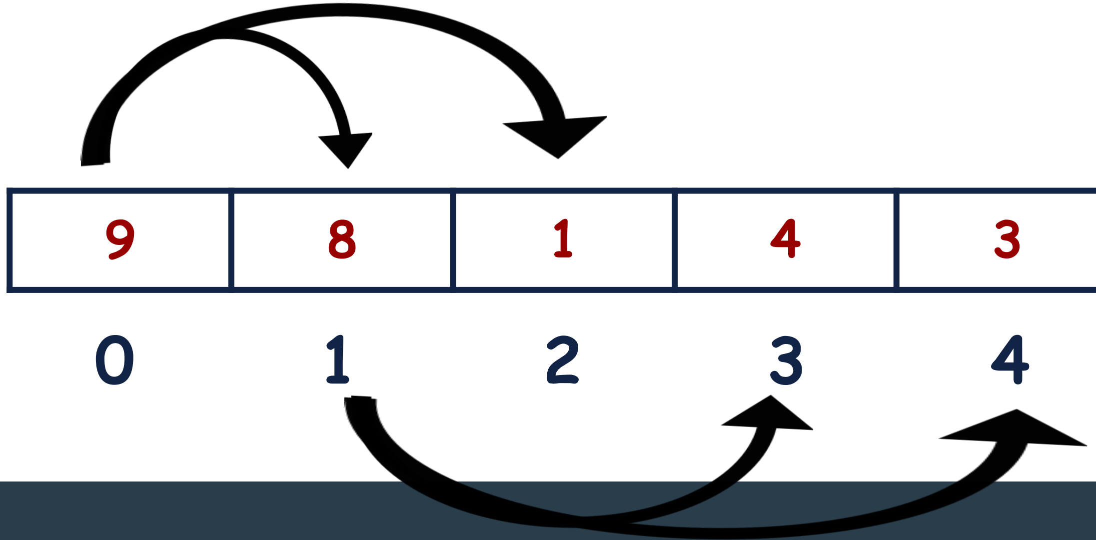
Heap: Implementation

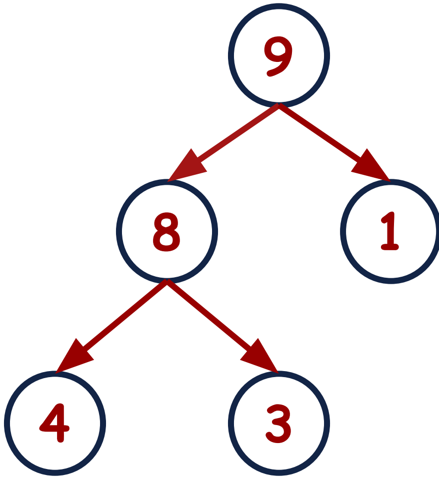
General Formulas: Let i be the current Index.

Parent Index: (i-1)/2

Left Child Index: (2*i) + 1

Right Child Index: (2*i) + 2





Heap: Implementation

```
int MAX = 10;

class Heap
{
    int heapArr[10];
    int size;

public:
    Heap()
    {
        size = 0;
    }
```

```
int parentIndex(int i)
{
    return (i - 1) / 2;
}
int leftChildIndex(int i)
{
    return (2 * i) + 1;
}
int rightChildIndex(int i)
{
    return (2 * i) + 2;
}
void swap(int a, int b)
{
    int temp = heapArr[a];
    heapArr[a] = heapArr[b];
    heapArr[b] = temp;
}
```

Heap (Insertion): Implementation

```
void insert(int value)
    {
        heapArr[size] = value;
        int index = size;
        size++;
        siftUp(index);
    }
```

```
void siftUp(int index)
{
    while (index > 0)
    {
        int pIdx = parentIndex(index);
        if (heapArr[index] > heapArr[pIdx])
        {
            swap(index, pIdx);
            index = pIdx;
        }
        else
        {
            break;
        }
    }
}
```

Heap (Deletion): Implementation

```
int deleteItem()
{
    if (size <= 0)
        cout << "UnderFlow!!! Queue is Empty";
    else if (size == 1)
    {
        size--;
        return heapArr[size];
    }
    else
    {
        int value = heapArr[0];
        heapArr[0] = heapArr[size - 1];
        size--;
        siftDown(0);
        return value;
    }
}
```

Heap (Deletion)

```
void siftDown(int index)
{
    int maxIndex;
    while (true)
    {
        int lIdx = leftChildIndex(index);
        int rIdx = rightChildIndex(index);
        if (rIdx >= size)
        {
            if (lIdx >= size)
                return;
            else
                maxIndex = lIdx;
        }
        else
        {
            if (heapArr[lIdx] >= heapArr[rIdx])
                maxIndex = lIdx;
            else
                maxIndex = rIdx;
        }
        if (heapArr[index] < heapArr[maxIndex])
            swap(index, maxIndex);
        else
            return;
    }
}
```

Learning Objective

Students should be able to use Heap data structure as Priority Queue.

Self Assessment

Consider a binary max-heap implemented using an array. Which one of the following array represents a binary max-heap?

A. 25, 14, 16, 13, 10, 8, 12
B. 25, 12, 16, 13, 10, 8, 14
C. 25, 14, 12, 13, 10, 8, 16
D. 25, 14, 13, 16, 10, 8, 12

Self Assessment

Consider the following max heap

50, 30, 20, 15, 10, 8, 16

Insert a new node with value 60.

Self Assessment

Consider the following max heap

50, 30, 20, 15, 10, 8, 16

Delete a node with value 50.

Self Assessment

Reading Links:

https://www.fluentcpp.com/2018/03/13/heaps-priority-queues-stl-part-1/

https://www.programiz.com/cpp-programming/priority-queue