DATA STRUCTURES

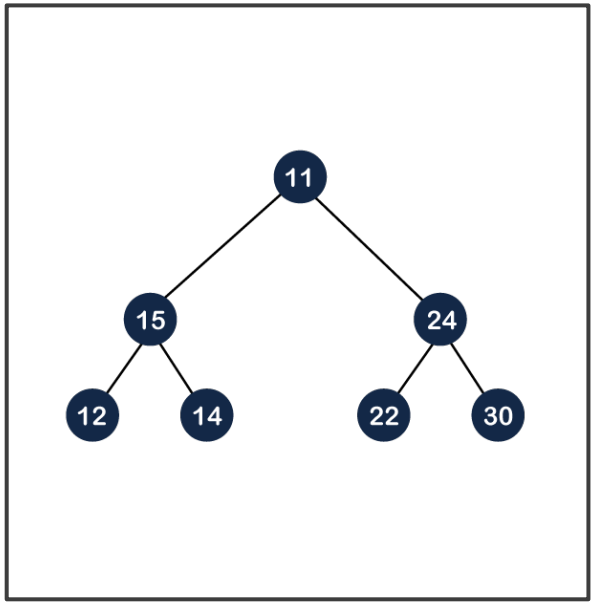
WENYE LI

CUHK-SZ

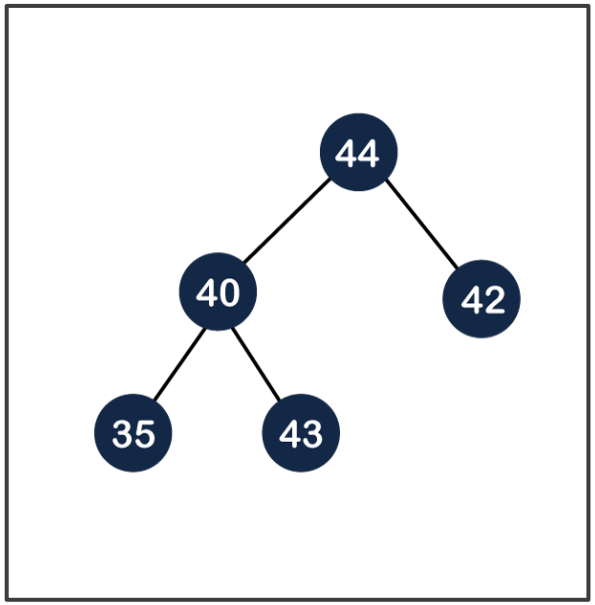
OUTLINE

- Concepts
- Implementations
- Examples

HEAP



- Recall: A complete binary tree is a binary tree in which all the levels except the last level, i.e., leaf node should be completely filled, and all the nodes should be left-justified.



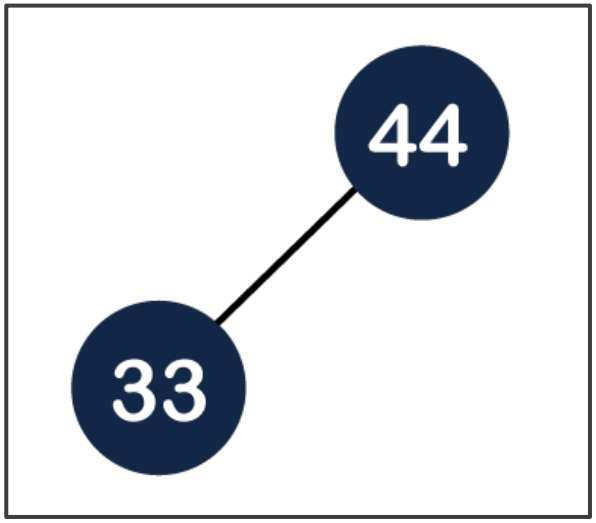
- What is Min Heap?
  - A complete binary tree.
  - The value of a parent node is less than or equal to its children.
- What is Max Heap?
  - A complete binary tree.
  - The value of a parent node is greater than or equal to its children.

INSERTION

- To create a max heap tree, consider the following two cases:
  - The property of the complete binary tree must be maintained.
  - The value of the parent node should be greater than the either of its child.

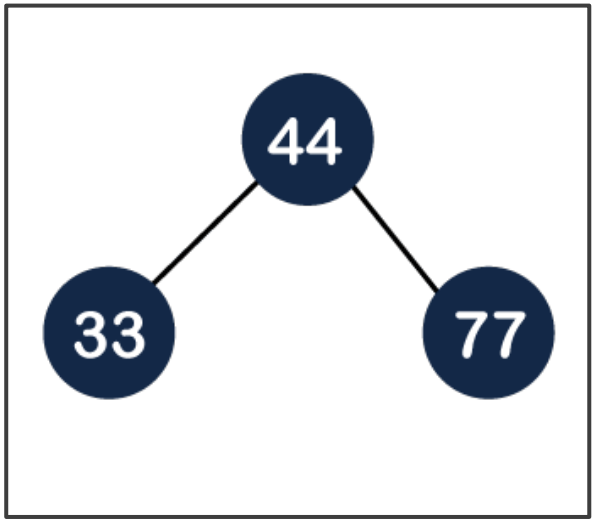
- Example: create a max heap with 44, 33, 77, 11, 55, 88, 66

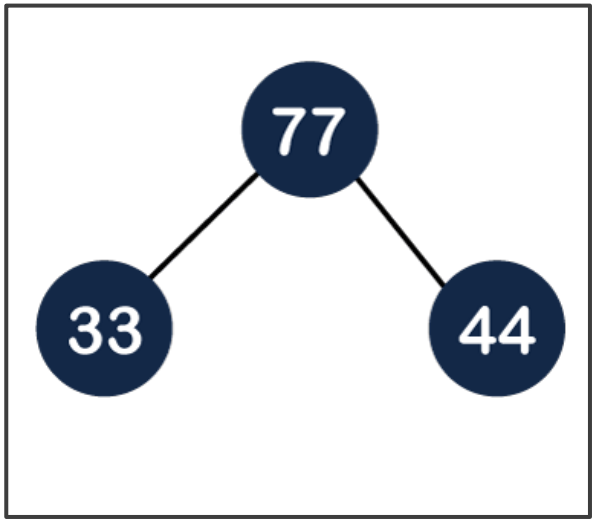
44, 33, 77, 11, 55, 88



- Step 1: First add 44 in the tree.
- Step 2: Insertion always starts from the left side. So add 33 to the left of 44.

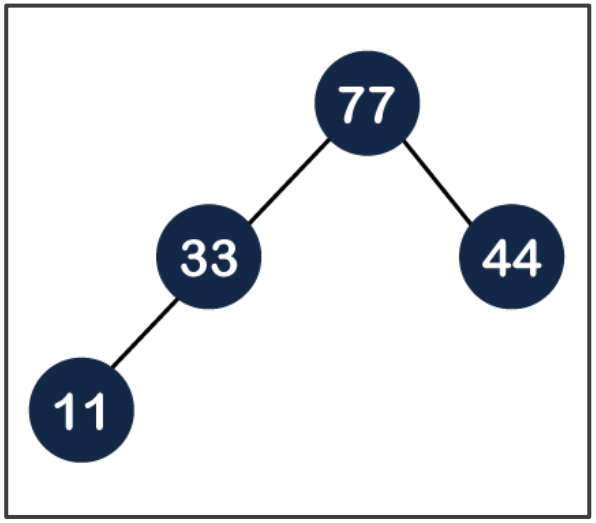
44, 33, 77, 11, 55, 88

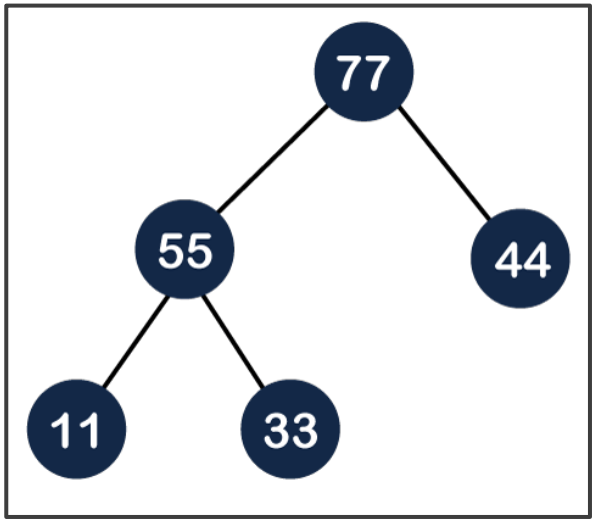




- Step 3: Add 77 to the right of the 44.
- Parent node 44 is less than the child 77. So swap these two values.

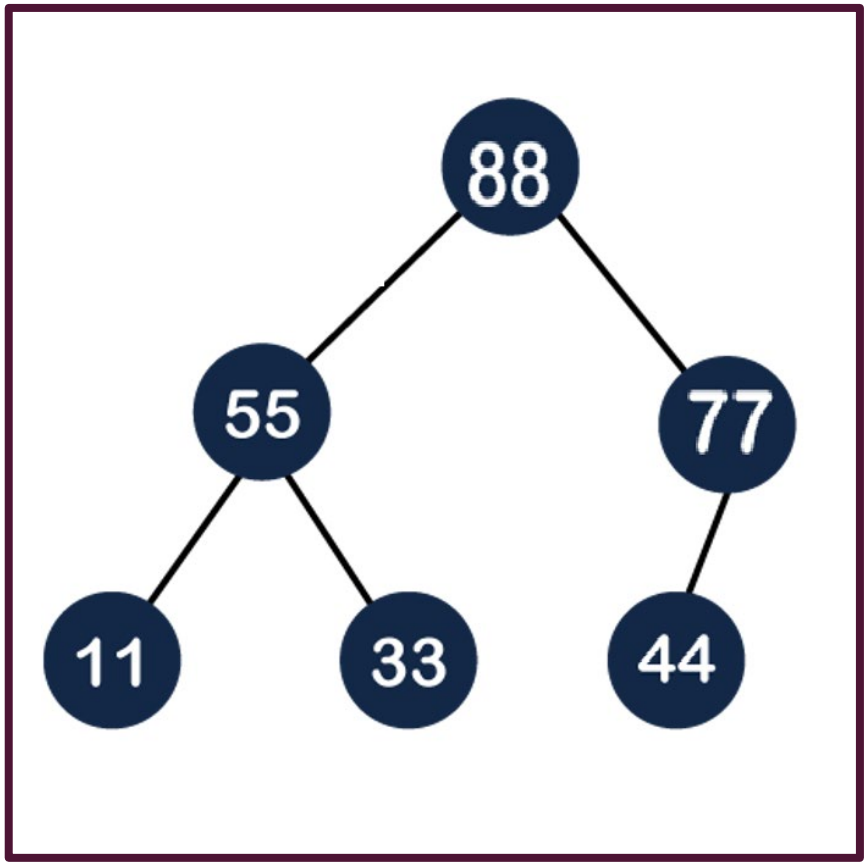
44, 33, 77, 11, 55, 88





- Step 4: Add the node 11 to the left of 33.
- Step 5: Add the node 55 to the right of 33.
- It does not satisfy the property of max heap because 33<55. So swap these two values.

44, 33, 77, 11, 55, 88



- Step 6: Add 88 to the left of 44.
- It does not satisfy the property of max heap because 44<88. So swap these two values.
- It is violating the max heap property because 88>77. So swap these two values.

DELETION

- The standard deletion operation on Heap is to delete the element present at the root node of the Heap.
  - If it is a Max Heap, the standard deletion operation will delete the maximum element.
  - If it is a Min Heap, it will delete the minimum element.

- Deleting an element at any intermediary position in the heap can be costly, so we can simply replace the element to be deleted by the last element and delete the last element of the Heap.
  - Replace the root or element to be deleted by the last element.
  - Delete the last element from the Heap.
  - Since, the last element is now placed at the position of the root node. So, it may not follow the heap property. Therefore, **heapify** the last node placed at the position of root.

```java
// Java program for implementing insertion in Heaps
public class InsertionHeap {
    // Function to heapify ith node in a Heap
    // of size n following a Bottom-up approach
    static void heapify(int[] arr, int n, int i) {
        // Find parent
        int parent = (i - 1) / 2;
        if (arr[parent] > 0) {
            // For Max-Heap
            // If current node is greater than its parent
            // Swap both of them and call heapify again
            // for the parent
            if (arr[i] > arr[parent]) {
                // swap arr[i] and arr[parent]
                int temp = arr[i];
                arr[i] = arr[parent];
                arr[parent] = temp;
                // Recursively heapify the parent node
                heapify(arr, n, parent);
            }
        }
    }
    static int insertNode(int[] arr, int n, int Key) {
        // Increase the size of Heap by 1
        n = n + 1;
        // Insert the element at end of Heap
        arr[n - 1] = Key;
        // Heapify the new node following a
        // Bottom-up approach
        heapify(arr, n, n - 1);
        // return new size of Heap
        return n;
    }
    static void printArray(int[] arr, int n) {
        for (int i = 0; i < n; ++i) System.out.println(arr[i] + " ");
        System.out.println();
    }
    public static void main(String args[]) {
        // Array representation of Max-Heap
        // 10
        //    /  \
        // 5    3
        //  / \
        // 2   4

        // maximum size of the array
        int MAX = 1000;
        int[] arr = new int[MAX];

        // initializing some values
        arr[0] = 10;
        arr[1] = 5;
        arr[2] = 3;
        arr[3] = 2;
        arr[4] = 4;

        // Current size of the array
        int n = 5;

        // the element to be inserted
        int Key = 15;

        // The function inserts the new element to the heap and
        // returns the new size of the array
        n = insertNode(arr, n, Key);

        printArray(arr, n);
        // Final Heap will be:
        //       15
        //    /   \
        //   5     10
        //  / \   /
        // 2   4 3
    }
}
```

```java
public class DeletionHeap {
    // Heapify a subtree rooted with node i as an index in arr[]. Nn: size of heap
    static void heapify(int arr[], int n, int i) {
        int largest = i; // Initialize largest as root
        int l = 2 * i + 1; // left = 2*i + 1
        int r = 2 * i + 2; // right = 2*i + 2
        // If left child is larger than root
        if (l < n && arr[l] > arr[largest]) largest = l;
        // If right child is larger than largest so far
        if (r < n && arr[r] > arr[largest]) largest = r;
        // If largest is not root
        if (largest != i) {
            int swap = arr[i];
            arr[i] = arr[largest];
            arr[largest] = swap;
            // Recursively heapify the affected sub-tree
            heapify(arr, n, largest);
        }
    }
    // Function to delete the root from Heap
    static int deleteRoot(int arr[], int n) {
        // Get the last element
        int lastElement = arr[n - 1];
        // Replace root with first element
        arr[0] = lastElement;
        // Decrease size of heap by 1
        n = n - 1;
        // heapify the root node
        heapify(arr, n, 0);
        // return new size of Heap
        return n;
    }
    static void printArray(int arr[], int n) {
        for (int i = 0; i < n; ++i) System.out.print(arr[i] + " ");
        System.out.println();
    }
    public static void main(String args[]) {
        //     10
        //    /  \
        //   5    3
        //  / \
        // 2   4
        int arr[] = { 10, 5, 3, 2, 4 };
        int n = arr.length;
        n = deleteRoot(arr, n);
        printArray(arr, n);
    }
}
```

Output:

```
5 4 3 2
```

THANKS