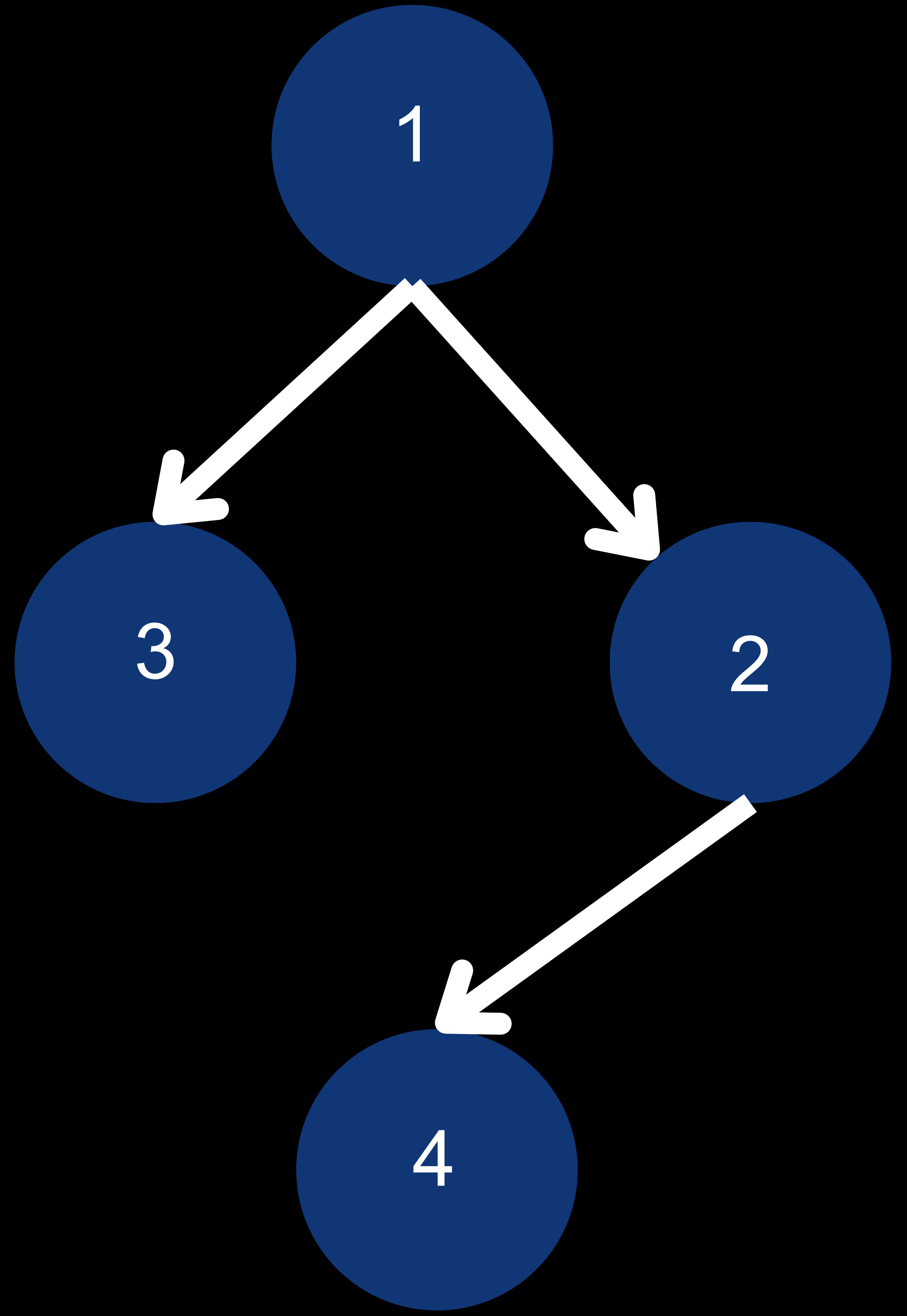
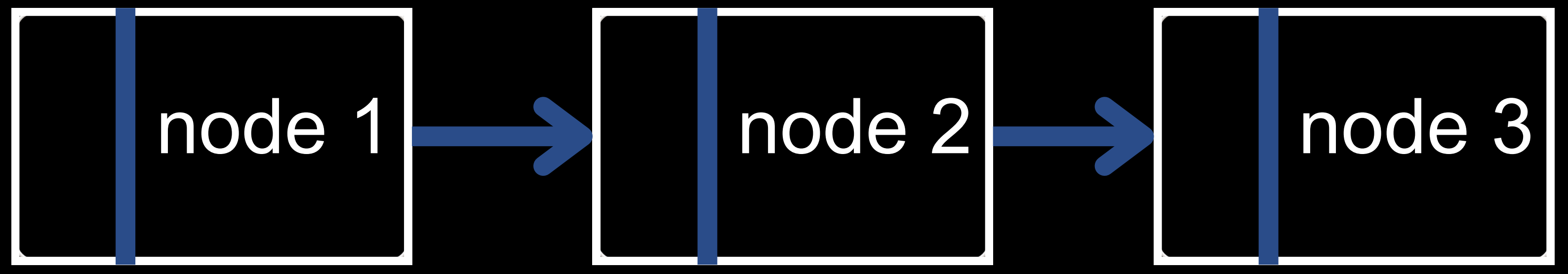
DATA STRUCTURE





essammohamedst1@gmail.com

Heaps

What is a Heap?

heap is just a complete binary tree with some Properties

- It is a Complete Binary Tree.
- It either follows max heap or min heap property.

Complete Binary Tree:

A heap must always be a complete binary tree, meaning all levels are filled from left to right, and if the last level is not full, the nodes should be as far left as possible.

rules of create a heap

1. Maintain the Binary Tree Structure

A heap must be a complete binary tree, which means:

All levels of the tree, except possibly the last, are fully filled.

The last level is filled from left to right with no gaps.

heap does not necessarily follow the property that the left child is less than the right child This property is not required for heaps.

2. Satisfy the Heap Property

we have two property of heap:

Max-Heap: Every parent node must be greater than or equal to its children.

Min-Heap: Every parent node must be less than or equal to its children

Max-Heap

A max-heap is a type of binary tree where:

Parent Nodes are Greater:

Each parent node's value is greater than or equal to the values of its children. This property

ensures that the largest element is always at the root of the heap

Complete Binary Tree:

A max-heap must be a complete binary tree, meaning:

All levels of the tree are fully filled except possibly the last level.

If the last level is not full, the nodes must be filled from left to right.

Min-Heap

A min-heap is a type of binary tree where:

Parent Nodes are Smaller:

Each parent node's value is less than or equal to the values of its children. This

property ensures that the smallest element is always at the root of the hea

Complete Binary Tree:

A min-heap must be a complete binary tree, meaning:

All levels of the tree are fully filled except possibly the last level.

If the last level is not full, nodes must be filled from left to right.

3. Add Elements Following the Structure Rule

When adding a new element:
Place the new element in the next available position (at the end of the last level, maintaining the complete binary tree structure).

Restore the heap property "heapifying":

max-heap: If the new element is greater than its parent, swap it with the parent (upward adjustment).

min-heap:
If the new element is smaller than its parent, swap it with the parent (upward adjustment).

4. Remove Elements While Maintaining the Heap Property

When removing the root element (usually the largest in a max-heap or smallest in a min-heap):

Replace the root with the last element in the heap (to maintain the complete binary tree structure).
Remove the last element.

Restore the heap propertyby "heapifying":
Compare the new root with its children.

max-heap: Swap the root with the larger child if the root is smaller than its children.

min-heap: Swap the root with the smaller child if the root is larger than its children.

Summary of Rules

Maintain a complete binary tree structure.

Enforce the heap property (max-heap or min-heap).

Use array representation for efficient storage.

Insert elements at the end and restore heap property via up-heapify.

Remove the root and restore heap property via down-heapify.

Use the Heapify Algorithm to build a heap from an array.

Optimize performance with O(logn)O(\log n)O(logn) insertion/deletion and

O(n)O(n)O(n) heap building

Array Index Relationships

For any node at index i in the array:

Left Child: The index is $2i + 1$.

Right Child: The index is $2i + 2$.

Parent: The index is $(i - 1) / 2$ (rounded down).

Steps to Create a Heap from an Array

Identify the Last Non-Leaf Node:
The last non-leaf node is at index (n // 2) - 1, where n is the size of the array. This is because nodes after this index are leaf nodes and do not need to be heapified

Perform Down-Heapify (Bubble Down):
Starting from the last non-leaf node, move upwards to the root

For each node, compare it with its children:
For Max-Heap: If the node is smaller than one of its children, swap it with the larger child.
For Min-Heap: If the node is greater than one of its children, swap it with the smaller child.
Repeat this process for the child nodes until the heap property is restored

Algorithm for Building a Heap

Find the last non-leaf node: lastNonLeafNode = (n // 2) - 1.
Heapify each node:
Start from the last non-leaf node and go up to the root (index 0).
For each node, call the heapify function.
Heapify Function (For Node at index i)
Identify its children:
Left child: 2i + 1
Right child: 2i + 2
Compare the node with its children:
For Max-Heap: Find the largest among the node and its children.
For Min-Heap: Find the smallest among the node and its children.
If the node violates the heap property, swap it with the largest (or smallest) child and recursively heapify the affected subtree.

Heap operations

1. Insert Operation

Algorithm:

Add the new element at the end of the heap (array).

Bubble-up (Up-Heapify):

Compare the new element with its parent.

If the new element is larger (for max-heap) or smaller (for min-heap) than its

parent, swap them.

Repeat the comparison with the new parent's parent, until the heap property is satisfied.

2. Delete Operation

Algorithm:

Replace the root element with the last element in the heap.

Down-Heapify (Bubble Down)the new root to restore the heap property.

Compare the root with its children.

Swap with the larger child (max-heap) or the smaller child (min-heap).

Repeat this process for the affected subtree.

3. Heapify Operation

Algorithm:

Start with a node and compare it with its left and right children.

For Max-Heap: Swap with the largest child if the node is smaller than the largest child.

For Min-Heap: Swap with the smallest child if the node is larger than the smallest child.

Repeat the process for the affected subtree.

4. Build Heap Operation

To build a heap from an unsorted array, use the Heapify function starting from the last non-leaf node and move upwards to the root.

Algorithm:

Identify the index of the last non-leaf node: (n // 2) - 1.

For each node from the last non-leaf node to the root (index 0), call the heapify function.

Heap sort

Steps in Heap Sort:

Build a Max-Heap: Arrange the elements of the array in a way that each parent node is greater than its children.

Extract Maximum: The maximum element is at the root of the heap (index 0) Swap it with the last element of the heap

Reheapify: After the swap, reduce the heap size and reheapify the heap to maintain the max-heap property.

Repeat: Repeat the extraction and reheapify steps until all elements are sorted.

Thanks

All code and examples can be found on GitHub https://github.com/ESSAMMOHAMED1