Module 6

Advanced Data Structures

Content

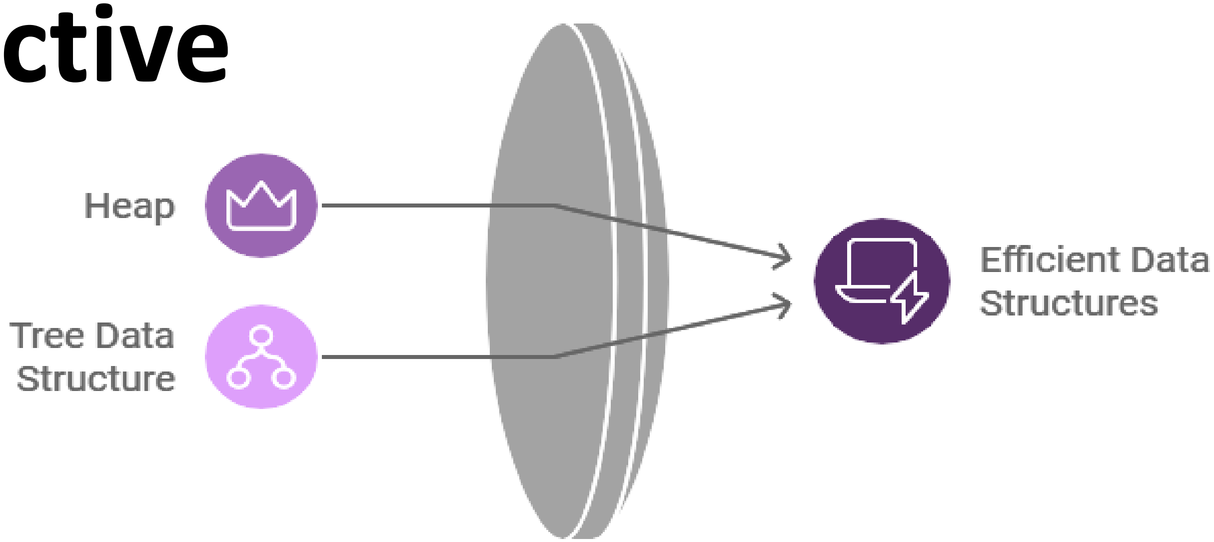
Part 1: Heap Structure

- Definition
- Operations
- Applications

Part 2: Tries Structure

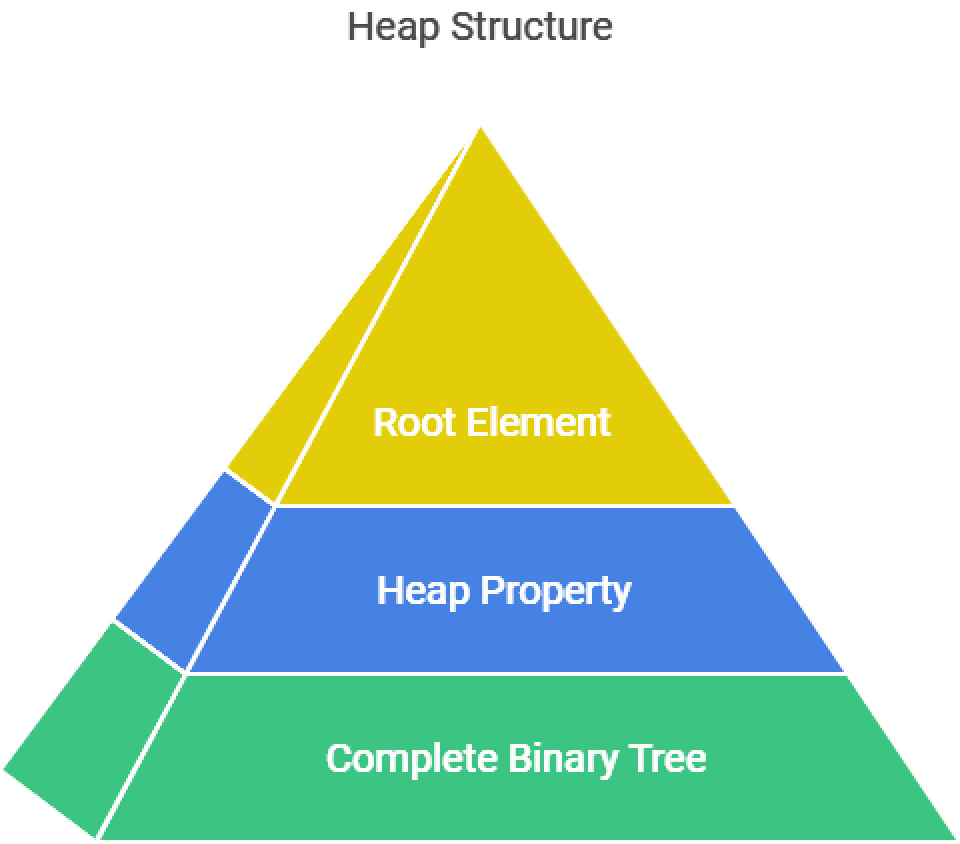
- Definition
- Terminologies
- Types
- Operations
- Applications

Objective

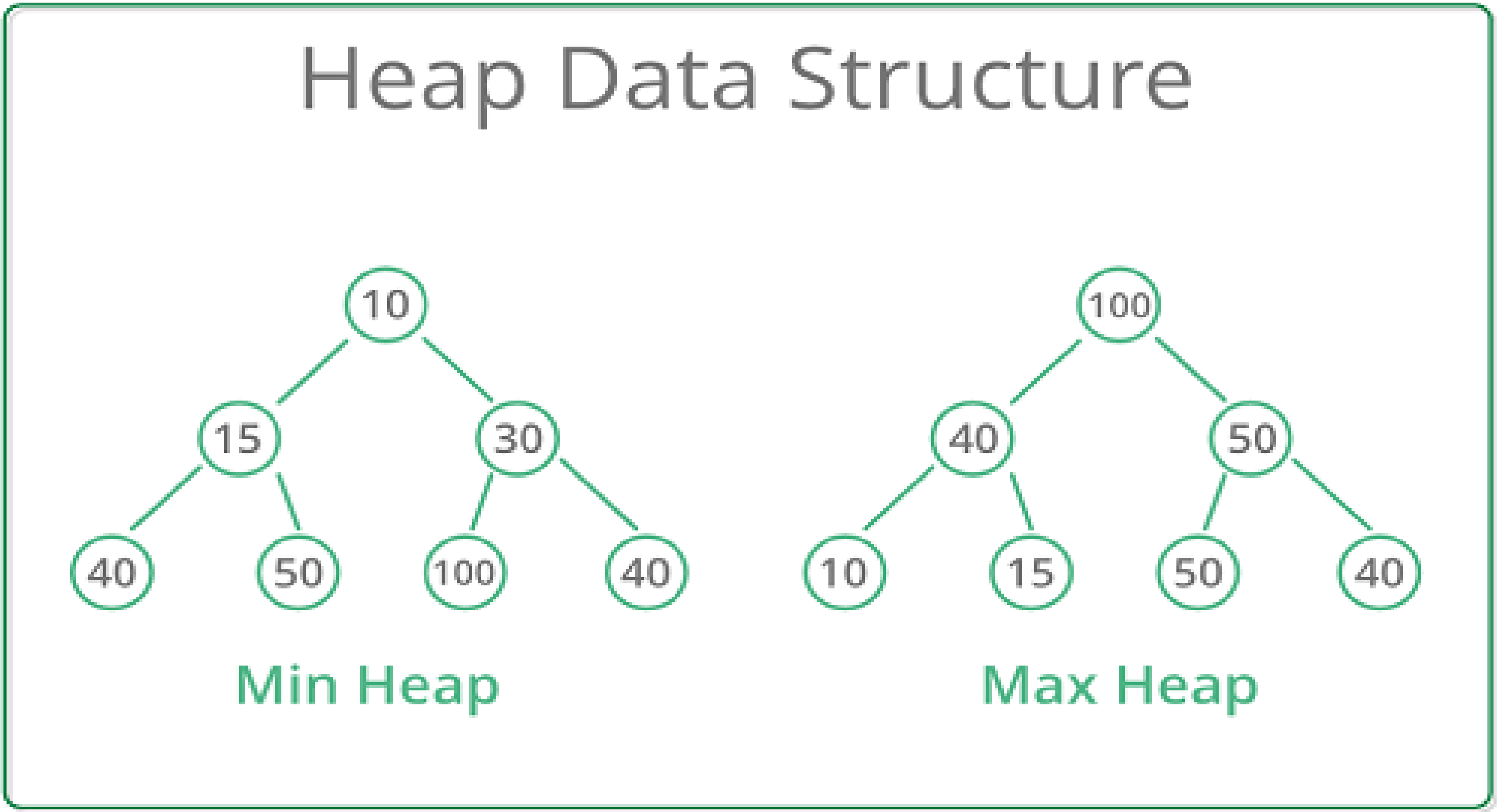


Part 1: Heap Structure

I. Definition and operations

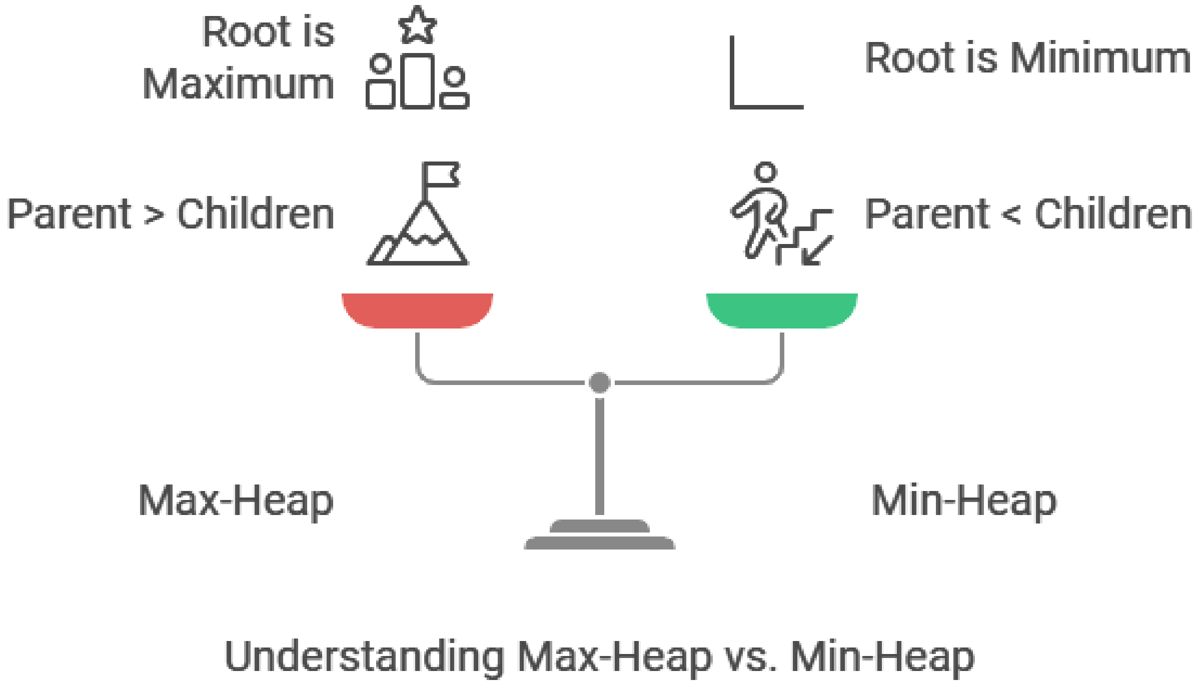


I. Definition and operations



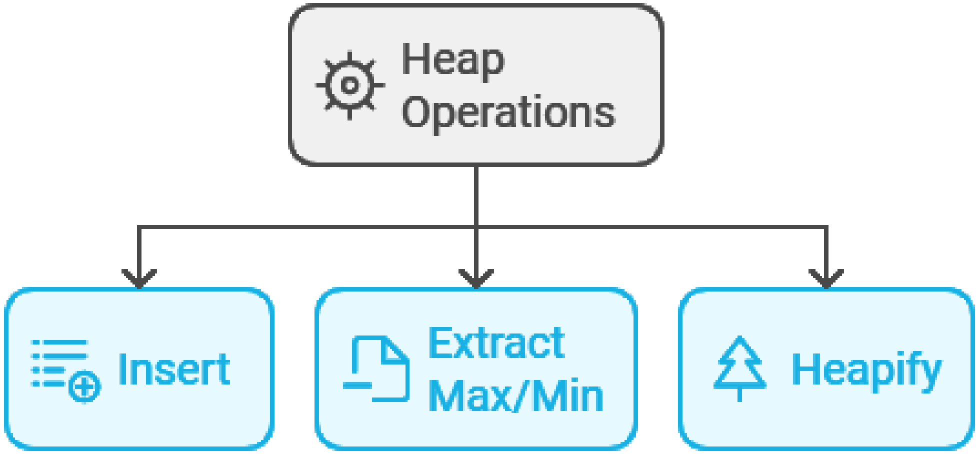
I. Definition and operations

a) **What is Heap Data Structure?**



I. Definition and operations

b) Heap Operations



II. Applications of heaps

Example 1:

Input: N = 5

arr[] = {4,1,3,9,7}

Output: 1 3 4 7 9

Explanation:

After sorting elements

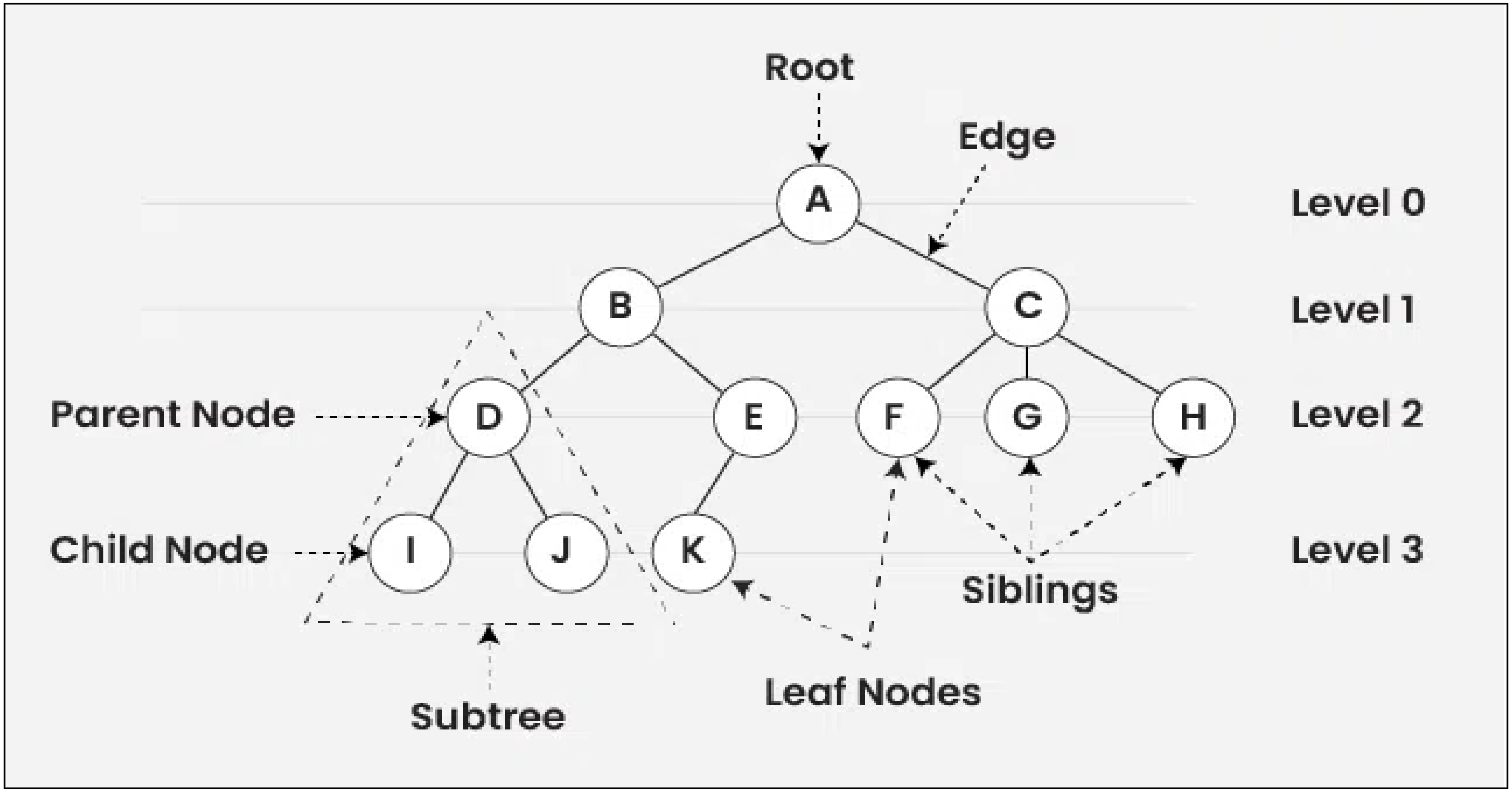
- using heap sort, elements will be
- in order as 1,3,4,7,9.

II. Applications of heaps

```
using System;  class HeapSort
{  public void Sort(int[] array)
   {   int n = array.Length; //(rearrange array)
      for (int i = n / 2 - 1; i >= 0; i--)
         Heapify(array, n, i); // One 2 one extract ele.
      for (int i = n - 1; i >= 0; i--)
      {   int temp = array[0]; // Move current to end
         array[0] = array[i];   array[i] = temp;
         Heapify(array, i, 0);     }
   } // Call max heapify on the reduced heap
   void Heapify(int[] array, int n, int i)
   {   int largest = i; // Initialize largest as root
      int left = 2 * i + 1; // left = 2*i + 1
      int right = 2 * i + 2; // right = 2*i + 2
      if (left < n && array[left] > array[largest])
          largest = left; // If right child >  largest
         if (right < n && array[right] > array[largest])
            largest = right;
         if (largest != i)       // If largest is not root
         {   int swap = array[i];  array[i] = array[largest];
            array[largest] = swap;    Heapify(array, n,
largest);   } // heapify the affected sub-tree
      static void PrintArray(int[] array)
      {   int n = array.Length; //fun to print array of size n
      for (int i = 0; i < n; ++i) Console.Write(array[i] + " ");
         Console.WriteLine();              }
      public static void Main()   // Driver program
      {   int[] array = { 12, 11, 13, 5, 6, 7 };
         HeapSort heapSort = new HeapSort();
heapSort.Sort(array); Console.WriteLine("Sorted
array is"); PrintArray(array);   }  }
```

Part 2: Tries Structure

I. Definition and Operations

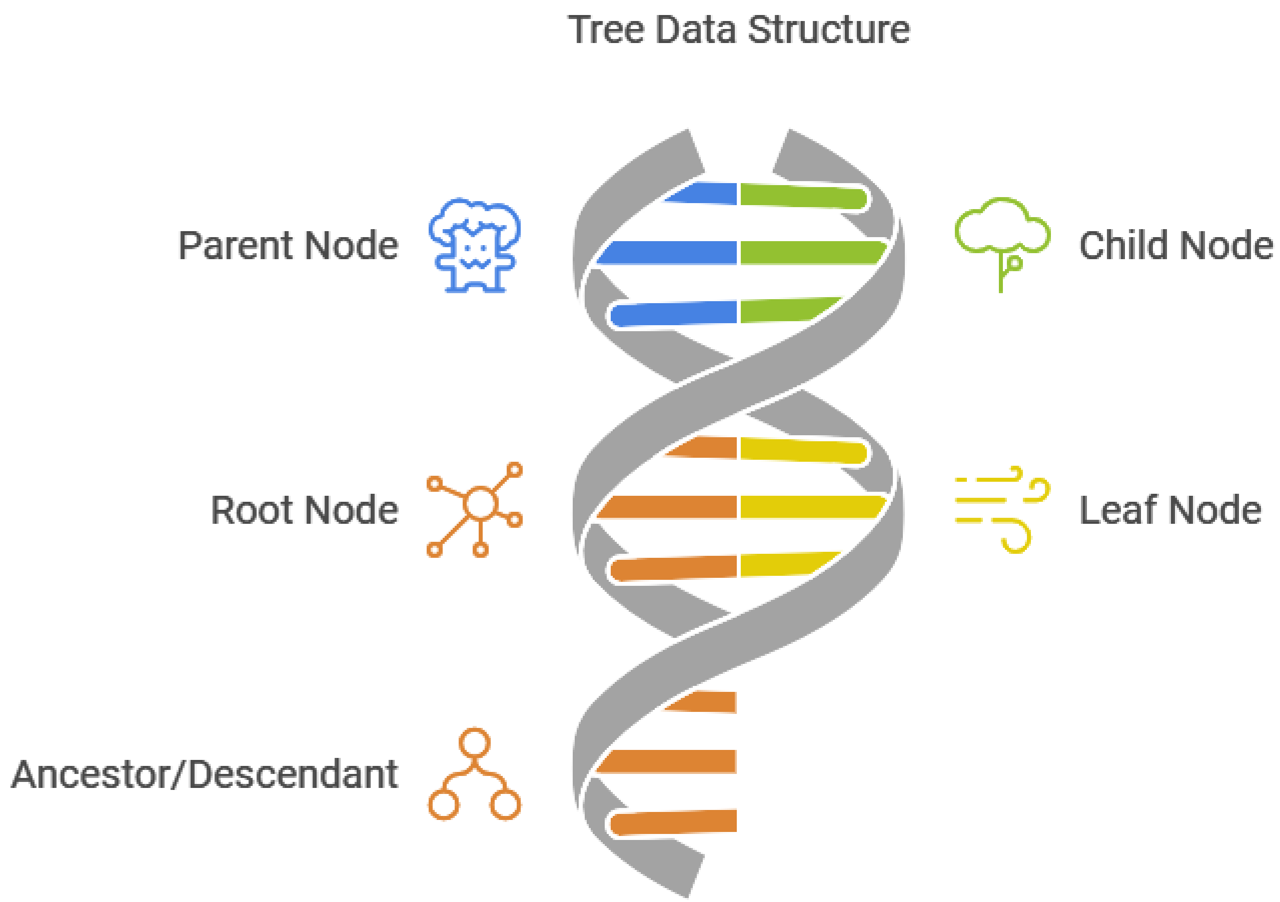


I. Definition and Operations

- Central Node
- Structural Nodes
- Sub-Nodes
- Roots
- Branches
- Leaves

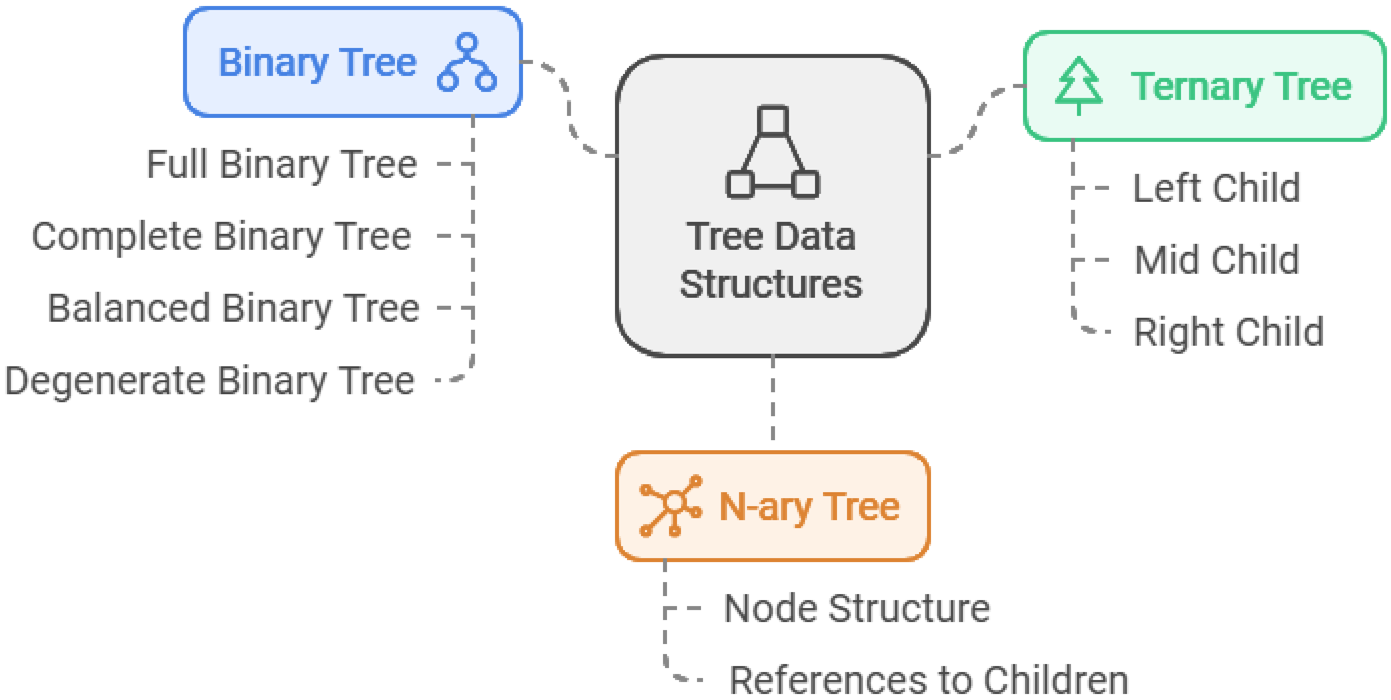
I. Definition and Operations

a) Basic Terminologies in Tree:



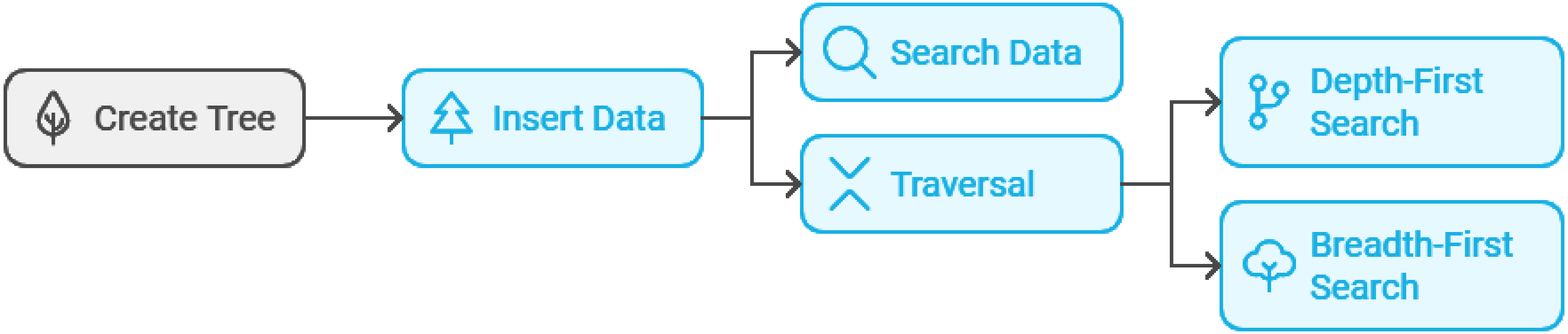
I. Definition and Operations

b) Types of Tree data structures:



I. Definition and Operations

c) Basic Operations of Tree Data Structure:



II. Applications of Tries

```
using System;
using System.Collections.Generic;
class Program
{   static void PrintParents(int node, List<List<int>> adj, int parent)
    {   if (parent == 0)
        {   Console.WriteLine($"{node} -> Root");       }
        else
        {   Console.WriteLine($"{node} -> {parent}");   }
        foreach (int cur in adj[node])
        {   if (cur != parent)
            {   PrintParents(cur, adj, node);   }       }
    }
static void PrintChildren(int Root, List<List<int>> adj)
    {   Queue<int> q = new Queue<int>();
        q.Enqueue(Root);
        bool[] vis = new bool[adj.Count];
        while (q.Count > 0)
        {   int node = q.Dequeue();  vis[node] = true;
            Console.Write($"{node} -> ");
            foreach (int cur in adj[node])
            {   if (!vis[cur])
                {   Console.Write($"{cur} ");
                    q.Enqueue(cur);             }
            }   Console.WriteLine();      }   }
 static void PrintLeafNodes(int Root, List<List<int>> adj)
    {  for (int i = 0; i < adj.Count; i++)
        {   if (adj[i].Count == 1 && i != Root)
            {   Console.Write($"{i} ");         }
        }       Console.WriteLine();
    }
```

II. Applications of Tries

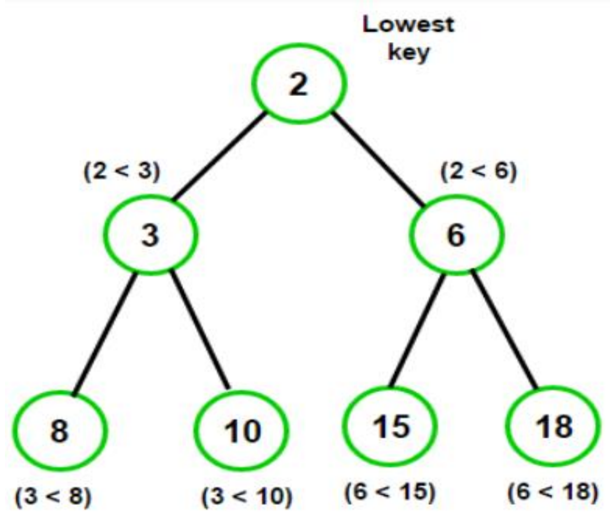
```
static void PrintDegrees(int Root, List<List<int>> adj)
  {   for (int i = 1; i < adj.Count; i++)
      {   Console.Write($"{i}: ");
          if (i == Root)
          {     Console.WriteLine(adj[i].Count);          }
          else
          {     Console.WriteLine(adj[i].Count - 1);  }  }
  }
  static void Main(string[] args)
  {   int N = 7;
      int Root = 1;
      List<List<int>> adj = new List<List<int>>();
      for (int i = 0; i <= N; i++)
      {       adj.Add(new List<int>());       }
      adj[1].AddRange(new int[] { 2, 3, 4 });
      adj[2].AddRange(new int[] { 1, 5, 6 });
      adj[4].Add(7);
Console.WriteLine("The parents of each node are:");
        PrintParents(Root, adj, 0);
 Console.WriteLine("The children of each node are:");
 PrintChildren(Root, adj);
 Console.WriteLine("The leaf nodes of the tree are:");
 PrintLeafNodes(Root, adj);
 Console.WriteLine("The degrees of each node are:");
 PrintDegrees(Root, adj);    }
}
```

Quizzes

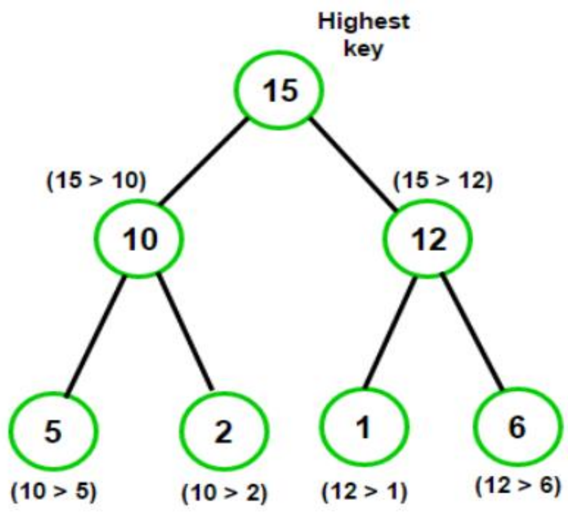
ACLEDA UNIVERSITY

1. What is Heap Data Structure?
2. What is the Heap Sort?
3. What are the basic Heap Operations?
4. What type of the following heap a, b:

(a)



(b)



5. What is the difference between Min Heap and Max heap?
6. Explain Heapify and where it is used in heap operations.

Quizzes

7. What is a tree data structure?
8. For the expression (7-(4*5))+(9/3) which of the following is the post-order tree traversal?
9. For the expression (7-(4*5))+(9/3) which of the following is the pre-order tree traversal?
10. For the expression (7-(4*5))+(9/3) which of the following is the In-order tree traversal?
11. Construct a binary tree by using post-order and in-order sequences given below.
    In-order: N, M, P, O, Q; Post-order: N, P, Q, O, M
12. Which (Pre, In, Post Order) traversal's pseudo code is written here?

```
order(node)
Q → Queue()
Q.push(node)
while !Q.empty():
    current_node = Q.pop()
    print current_node.value
    if current_node.left is not NULL:
        Q.push(current_node.left)
    if current_node.right is not NULL:
        Q.push(current_node.right)
```


THANK
YOU!