**Aim**: Implement Heap tree and Heap sort.

**Objective:**

1.To learn how the Max-Heap data structure works, including its insertion and deletion operations.

2. To implement heap operations (heapify, insertion, and deletion) in C++.

3. To evaluate the time and space complexity of heap operations.

**Tools Used:** Visual Studio Code.

**Concept**:

A heap tree is a special binary tree used to efficiently manage data. It helps in tasks like sorting and creating priority queues.

Two key properties of a heap tree:

1. Complete Binary Tree: All levels of the tree are filled except possibly the last, which is filled from left to right.
2. Heap Property:
   * Max-Heap: The parent node is always greater than or equal to its children, and the largest value is at the root.
   * Min-Heap: The parent node is always smaller than or equal to its children, and the smallest value is at the root.

Sorting with a Max-Heap (Heap Sort)

Heap Sort organizes numbers into ascending order using the following steps:

1. Build a Max-Heap:
   * Rearrange the array into a Max-Heap so the largest value is at the root.
2. Sort:
   * Swap the root (largest value) with the last element.
   * Reduce the heap size by 1.
   * Fix the heap property by applying Heapify to the root.
   * Repeat until all elements are sorted.

Example of Heap Sort

1. Start with an unsorted array: [4, 10, 3, 5, 1].
2. Build a Max-Heap: [10, 5, 3, 4, 1].
3. Swap the root with the last element: [5, 4, 3, 1, 10].
4. Fix the heap (Heapify) and repeat: [1, 3, 4, 5, 10].

**Solution:**

#include <iostream>

using namespace std;

class Heap

{

    int arr[100];

    int size;

public:

    Heap() : size(0) {}

    void insertElement()

    {

        int n;

        cout << "Enter the number of elements to insert into the heap: ";

        cin >> n;

        cout << "Enter " << n << " elements: ";

        for (int i = 0; i < n; ++i)

        {

            int element;

            cin >> element;

            arr[size++] = element;

        }

        cout << "Heap constructed successfully!\n";

    }

    void heapifyIterative(int arr[], int n, int i, bool isMaxHeap)

    {

        while (true)

        {

            int largestOrSmallest = i;

            int left = 2 \* i + 1;

            int right = 2 \* i + 2;

            if (isMaxHeap)

            {

                if (left < n && arr[left] > arr[largestOrSmallest])

                {

                    largestOrSmallest = left;

                }

                if (right < n && arr[right] > arr[largestOrSmallest])

                {

                    largestOrSmallest = right;

                }

            }

            else

            {

                if (left < n && arr[left] < arr[largestOrSmallest])

                {

                    largestOrSmallest = left;

                }

                if (right < n && arr[right] < arr[largestOrSmallest])

                {

                    largestOrSmallest = right;

                }

            }

            if (largestOrSmallest == i)

                break;

            swap(arr[i], arr[largestOrSmallest]);

            i = largestOrSmallest;

        }

    }

    void heapSort(bool isMaxHeap)

    {

        for (int i = size / 2 - 1; i >= 0; i--)

        {

            heapifyIterative(arr, size, i, isMaxHeap);

        }

        for (int i = size - 1; i > 0; i--)

        {

            swap(arr[0], arr[i]);

            heapifyIterative(arr, i, 0, isMaxHeap);

        }

    }

    void displayMinHeap()

    {

        cout << "Max Heap elements (big to small): ";

        heapSort(false);

        for (int i = 0; i < size; ++i)

        {

            cout << arr[i] << " ";

        }

        cout << "\n";

    }

    void displayMaxHeap()

    {

        cout << "min Heap elements (small to big): ";

        heapSort(true);

        for (int i = 0; i < size; ++i)

        {

            cout << arr[i] << " ";

        }

        cout << "\n";

    }

};

int main()

{

    Heap heap;

    int choice;

    do

    {

        cout << "\n--- HEAP MENU ---";

        cout << "\n1. Insert elements (Build Heap)";

        cout << "\n2. Display Min Heap";

        cout << "\n3. Display Max Heap";

        cout << "\n4. Exit";

        cout << "\nEnter your choice: ";

        cin >> choice;

        switch (choice)

        {

        case 1:

            heap.insertElement();

            break;

        case 2:

            heap.displayMaxHeap();

            break;

        case 3:

            heap.displayMinHeap();

            break;

        case 4:

            cout << "Exiting the program.\n";

            break;

        default:

            cout << "Invalid choice! Please try again.\n";

        }

    } while (choice != 4);

    return 0;

}

Output:

--- HEAP MENU ---

1. Insert elements (Build Heap)

2. Display Min Heap

3. Display Max Heap

4. Exit

Enter your choice: 1

Enter the number of elements to insert into the heap: 7

Enter 7 elements: 10

30

12

18

7

6

4

Heap constructed successfully!

--- HEAP MENU ---

1. Insert elements (Build Heap)

2. Display Min Heap

3. Display Max Heap

4. Exit

Enter your choice: 3

Max Heap elements (big to small): 30 18 12 10 7 6 4

--- HEAP MENU ---

1. Insert elements (Build Heap)

2. Display Min Heap

3. Display Max Heap

4. Exit

Enter your choice: 2

min Heap elements (small to big): 4 6 7 10 12 18 30

--- HEAP MENU ---

1. Insert elements (Build Heap)

2. Display Min Heap

3. Display Max Heap

4. Exit

Enter your choice: 4

Exiting the program.

PS D:\SPIT-FYMCA\DS PRAC\LAB 9>

# Observation:

# A heap tree is a type of binary tree that follows the heap property. There are two main types of heaps: max-heap and min-heap.

# In a max-heap, the value of each node is greater than or equal to the values of its children, ensuring that the largest value is always at the root.

# In a min-heap, the value of each node is smaller than or equal to the values of its children, so the smallest value is always at the root.