DSA Assignment 9

1.

```cpp
#include <iostream>
using namespace std;

void heapify(int arr[], int n, int i) {
    int largest = i;
    int left = 2*i + 1;
    int right = 2*i + 2;

    if (left < n && arr[left] > arr[largest])
        largest = left;

    if (right < n && arr[right] > arr[largest])
        largest = right;

    if (largest != i) {
        swap(arr[i], arr[largest]);
        heapify(arr, n, largest);
    }
}

void heapSortIncreasing(int arr[], int n) {
    for (int i = n/2 - 1; i >= 0; i--)
        heapify(arr, n, i);

    for (int i = n-1; i >= 0; i--) {
        swap(arr[0], arr[i]);
        heapify(arr, i, 0);
    }
}

void heapSortDecreasing(int arr[], int n) {
    // Build MIN-HEAP
    for (int i = n/2 - 1; i >= 0; i--) {
        int smallest = i;
        int left = 2*i + 1;
        int right = 2*i + 2;

        if (left < n && arr[left] < arr[smallest]) smallest = left;
        if (right < n && arr[right] < arr[smallest]) smallest = right;

        if (smallest != i) {
            swap(arr[i], arr[smallest]);
            i = n; // restart heap build
        }
    }

    // Extract min
    for (int i = n-1; i >= 0; i--) {
        swap(arr[0], arr[i]);
        int size = i;

        int root = 0;
        while (true) {
            int left = 2*root + 1;
```

```
            int right = 2*root + 2;
            int smallest = root;

            if (left < size && arr[left] < arr[smallest]) smallest = left;
            if (right < size && arr[right] < arr[smallest]) smallest = right;

            if (smallest != root) {
                swap(arr[root], arr[smallest]);
                root = smallest;
            }
            else break;
        }
    }
}

void printArray(int arr[], int n) {
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";
    cout << endl;
}

int main() {
    int arr[] = {12, 11, 13, 5, 6, 7};
    int n = 6;

    cout << "Original Array: ";
    printArray(arr, n);

    heapSortIncreasing(arr, n);
    cout << "Sorted Increasing: ";
    printArray(arr, n);

    int arr2[] = {12, 11, 13, 5, 6, 7};
    heapSortDecreasing(arr2, n);
    cout << "Sorted Decreasing: ";
    printArray(arr2, n);

    return 0;
}
```

OUTPUT

```
Original Array: 12 11 13 5 6 7
Sorted Increasing: 5 6 7 11 12 13
Sorted Decreasing: 13 12 11 7 6 5
```

2.

```
#include <iostream>
using namespace std;

class PriorityQueue {
public:
    int arr[100];
    int size;

    PriorityQueue() {
```

```
        size = 0;
    }

    void heapifyUp(int index) {
        while (index > 0) {
            int parent = (index - 1) / 2;
            if (arr[parent] < arr[index]) {
                swap(arr[parent], arr[index]);
                index = parent;
            } else break;
        }
    }

    void heapifyDown(int index) {
        while (true) {
            int left = 2*index + 1;
            int right = 2*index + 2;
            int largest = index;

            if (left < size && arr[left] > arr[largest])
                largest = left;
            if (right < size && arr[right] > arr[largest])
                largest = right;

            if (largest != index) {
                swap(arr[index], arr[largest]);
                index = largest;
            } else break;
        }
    }

    void insert(int x) {
        arr[size] = x;
        size++;
        heapifyUp(size - 1);
    }

    int getMax() {
        if (size == 0) return -1;
        return arr[0];
    }

    int extractMax() {
        if (size == 0) return -1;

        int maxVal = arr[0];
        arr[0] = arr[size - 1];
        size--;

        heapifyDown(0);

        return maxVal;
    }

    void print() {
        for (int i = 0; i < size; i++)
            cout << arr[i] << " ";
        cout << endl;
```

```
    }
};

int main() {
    PriorityQueue pq;

    pq.insert(30);
    pq.insert(10);
    pq.insert(50);
    pq.insert(40);

    cout << "Priority Queue (Heap): ";
    pq.print();

    cout << "Max Element: " << pq.getMax() << endl;

    cout << "Extracting Max: " << pq.extractMax() << endl;
    cout << "After Extraction: ";
    pq.print();

    return 0;
}
```

OUTPUT

```
Priority Queue (Heap): 50 40 30 10
Max Element: 50
Extracting Max: 50
After Extraction: 40 10 30
```