**DS LAB 10**

**Task 01**

#include <iostream>

using namespace std;

class MaxHeap {

    public:

        int capacity;

        int size;

        int \*arr;

        MaxHeap (int c) : size(0), capacity(c) {

            arr = new int[capacity];

        }

        ~MaxHeap () {

            delete[] arr;

        }

        void heapify(int i) {

            int largest = i;

            int leftChild = (2 \* i) + 1;

            int rightChild = (2 \* i) + 2;

            if (leftChild < size && arr[leftChild] > arr[largest]) {

                largest = leftChild;

            }

            if (rightChild < size && arr[rightChild] > arr[largest]) {

                largest = rightChild;

            }

            if (largest != i) {

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

                heapify(largest);

            }

        }

        void insert(int value) {

            if (size == capacity) {

                cout << "The array is full beta." << endl;

                return;

            }

            int i = size;

            arr[i] = value;

            size++;

            while (i != 0 && arr[(i-1)/2] < arr[i]) {

                swap(arr[(i-1)/2], arr[i]);

                i = (i - 1) / 2;

            }

        }

        int extractMax() {

            if (size == 0) {

                cout << "Heap is empty. Cannot extract maximum element.";

            return -1;

            }

            int max = arr[0];

            arr[0] = arr[size - 1];

            size--;

            heapify(0);

            return max;

        }

        void deleteMax() {

            extractMax();

        }

        void printArray() {

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

                cout << arr[i] << " ";

            }

            cout << endl;

        }

};

class MinHeap {

    public:

        int capacity;

        int size;

        int \*arr;

        MinHeap (int c) : size(0), capacity(c) {

            arr = new int[capacity];

        }

        ~MinHeap () {

            delete[] arr;

        }

        void heapify(int i) {

            int smallest = i;

            int leftChild = (2 \* i) + 1;

            int rightChild = (2 \* i) + 2;

            if (leftChild < size && arr[leftChild] < arr[smallest]) {

                smallest = leftChild;

            }

            if (rightChild < size && arr[rightChild] < arr[smallest]) {

                smallest = rightChild;

            }

            if (smallest != i) {

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

                heapify(smallest);

            }

        }

        void insert(int value) {

            if (size == capacity) {

                cout << "The array is full beta." << endl;

                return;

            }

            int i = size;

            arr[i] = value;

            size++;

            while (i != 0 && arr[(i-1)/2] > arr[i]) {

                swap(arr[(i-1)/2], arr[i]);

                i = (i - 1) / 2;

            }

        }

        int extractMin() {

            if (size == 0) {

                cout << "Heap is empty. Cannot extract minimum element.";

            return -1;

            }

            // Store the root element

            int min = arr[0];

            // Replace the root with the last element

            arr[0] = arr[size - 1];

            size--;

            // Heapify the root

            heapify(0);

            return min;

        }

        void deleteMin() {

            extractMin();

        }

        void printArray() {

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

                cout << arr[i] << " ";

            }

            cout << endl;

        }

};

int main () {

    MaxHeap h1(10);

    h1.insert(7);

    h1.insert(1);

    h1.insert(6);

    h1.insert(2);

    h1.insert(5);

    h1.insert(9);

    h1.insert(10);

    h1.insert(2);

    cout << "------- Max Heap -------" << endl;

    h1.printArray();

    cout << "Inserting 99.." << endl;

    h1.insert(99);

    h1.printArray();

    cout << "Delete Max" << endl;

    h1.deleteMax();

    h1.printArray();

    MinHeap h2(10);

    h2.insert(7);

    h2.insert(1);

    h2.insert(6);

    h2.insert(2);

    h2.insert(5);

    h2.insert(9);

    h2.insert(10);

    h2.insert(2);

    cout << "\n------- Min Heap -------" << endl;

    h2.printArray();

    cout << "Inserting 99.." << endl;

    h2.insert(99);

    h2.printArray();

    cout << "Delete Min" << endl;

    h2.deleteMin();

    h2.printArray();

    return 0;

}

A screenshot of a computer

Description automatically generated

**Task 02**

#include <iostream>

#include <algorithm>

using namespace std;

class Heap {

    public:

        int \*arr;

        int size;

        Heap (int arr[], int size) {

            this->arr = arr;

            this->size = size;

        }

        void minHeapify(int i) {

            int smallest = i;

            int leftChild = (2 \* i) + 1;

            int rightChild = (2 \* i) + 2;

            if (leftChild < size && arr[leftChild] < arr[smallest]) {

                smallest = leftChild;

            }

            if (rightChild < size && arr[rightChild] < arr[smallest]) {

                smallest = rightChild;

            }

            if (smallest != i) {

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

                minHeapify(smallest);

            }

        }

        void buildMinHeap() {

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

                minHeapify(i);

            }

        }

        void minHeapSort() {

            buildMinHeap();

            int heapSize = size;

            for (int i = heapSize - 1; i > 0; i--) {

                swap(arr[0], arr[i]);

                heapSize--;

                minHeapify(0);

            }

        }

        void maxHeapify(int i) {

            int largest = i;

            int leftChild = (2 \* i) + 1;

            int rightChild = (2 \* i) + 2;

            if (leftChild < size && arr[leftChild] > arr[largest]) {

                largest = leftChild;

            }

            if (rightChild < size && arr[rightChild] > arr[largest]) {

                largest = rightChild;

            }

            if (largest != i) {

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

                maxHeapify(largest);

            }

        }

        void buildMaxHeap() {

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

                maxHeapify(i);

            }

        }

        void maxHeapSort() {

            buildMaxHeap();

            int heapSize = size;

            for (int i = heapSize - 1; i > 0; i--) {

                swap(arr[0], arr[i]);

                heapSize--;

                maxHeapify(0);

            }

        }

        void deleteRoot() {

            if (size == 0) {

                cout << "Heap is empty, cannot delete root." << endl;

                return;

            }

            arr[0] = arr[size - 1];

            size--;

            minHeapify(0);

        }

        void printArray() {

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

                cout << arr[i] << " ";

            }

            cout << endl;

        }

};

int main() {

    int arr[] = {35, 33, 42, 10, 14, 19, 27, 44, 26, 31};

    int n = sizeof(arr) / sizeof(arr[0]);

    Heap heap(arr, n);

    cout << "Original Array:" << endl;

    heap.printArray();

    cout << "\nMinHeap:" << endl;

    heap.buildMinHeap();

    heap.printArray();

    cout << "\nAfter Deleting Root from MinHeap:" << endl;

    heap.deleteRoot();

    heap.printArray();

    cout << "\nMaxHeap:" << endl;

    heap.buildMaxHeap();

    heap.printArray();

    heap.maxHeapSort();

    cout << "\nSorted Array:" << endl;

    heap.printArray();

    return 0;

}

A screenshot of a computer

Description automatically generated

**Task 03**

#include <iostream>

#include <algorithm>

using namespace std;

class Heap {

    public:

        int \*arr;

        int size;

        Heap(int arr[], int size) {

            this->arr = arr;

            this->size = size;

        }

        void minHeapify(int i) {

                int smallest = i;

                int leftChild = (2 \* i) + 1;

                int rightChild = (2 \* i) + 2;

                if (leftChild < size && arr[leftChild] < arr[smallest]) {

                    smallest = leftChild;

                }

                if (rightChild < size && arr[rightChild] < arr[smallest]) {

                    smallest = rightChild;

                }

                if (smallest != i) {

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

                    minHeapify(smallest);

                }

            }

        void buildMinHeap() {

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

                    minHeapify(i);

                }

            }

        void minHeapSort() {

                buildMinHeap();

                int heapSize = size;

                for (int i = heapSize - 1; i > 0; i--) {

                    swap(arr[0], arr[i]);

                    heapSize--;

                    minHeapify(0);

                }

            }

        void printArray() {

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

                cout << arr[i] << " ";

            }

            cout << endl;

        }

};

int main() {

    int arr[] = {4, 1, 3, 9, 7};

    int n = sizeof(arr) / sizeof(arr[0]);

    Heap heap(arr, n);

    cout << "Original Array:" << endl;

    heap.printArray();

    heap.minHeapSort();

    cout << "\nSorted Array:" << endl;

    heap.printArray();

    return 0;

}

A black screen with white text

Description automatically generated

**Task 04**

#include <iostream>

#include <cstdlib>

#include <ctime>

using namespace std;

class Task {

public:

    string name;

    int priority;

    Task(string name, int priority) : name(name), priority(priority) {}

    bool operator>(const Task &other) const {

        // For MaxHeap, higher priority (larger value) should be placed before

        return priority > other.priority;

    }

    bool operator<(const Task &other) const {

        // For MinHeap, lower priority (smaller value) should be placed before

        return priority < other.priority;

    }

    void printTask() const {

        cout << name << " (Priority: " << priority << ")";

    }

};

class MaxHeap {

public:

    int capacity;

    int size;

    Task \*\*arr;

    MaxHeap(int c) : size(0), capacity(c) {

        arr = new Task\*[capacity];

    }

    ~MaxHeap() {

        delete[] arr;

    }

    void heapify(int i) {

        int largest = i;

        int leftChild = (2 \* i) + 1;

        int rightChild = (2 \* i) + 2;

        if (leftChild < size && \*arr[leftChild] > \*arr[largest]) {

            largest = leftChild;

        }

        if (rightChild < size && \*arr[rightChild] > \*arr[largest]) {

            largest = rightChild;

        }

        if (largest != i) {

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

            heapify(largest);

        }

    }

    void insert(Task \*task) {

        if (size == capacity) {

            cout << "Heap is full." << endl;

            return;

        }

        int i = size;

        arr[i] = task;

        size++;

        while (i != 0 && \*arr[(i - 1) / 2] < \*arr[i]) {

            swap(arr[(i - 1) / 2], arr[i]);

            i = (i - 1) / 2;

        }

    }

    Task\* extractMax() {

        if (size == 0) {

            cout << "Heap is empty. Cannot extract maximum task." << endl;

            return nullptr;

        }

        Task\* max = arr[0];

        arr[0] = arr[size - 1];

        size--;

        heapify(0);

        return max;

    }

    void deleteMax() {

        Task\* task = extractMax();

        if (task) {

            delete task;

        }

    }

    void printArray() {

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

            arr[i]->printTask();

            cout << endl;

        }

    }

};

int main() {

    srand(time(0));

    MaxHeap taskHeap(10);

    // Insert some tasks with random priorities

    taskHeap.insert(new Task("Task 1", rand() % 10 + 1));

    taskHeap.insert(new Task("Task 2", rand() % 10 + 1));

    taskHeap.insert(new Task("Task 3", rand() % 10 + 1));

    taskHeap.insert(new Task("Task 4", rand() % 10 + 1));

    taskHeap.insert(new Task("Task 5", rand() % 10 + 1));

    cout << "------ Scheduled Tasks (Max Priority First) ------" << endl;

    taskHeap.printArray();

    // Process tasks in order of priority

    cout << "\nScheduling tasks..." << endl;

    while (true) {

        Task\* task = taskHeap.extractMax();

        if (!task) break;

        cout << "Scheduled: ";

        task->printTask();

        cout << endl;

        delete task;  // Free memory after task is scheduled

    }

    return 0;

}

A computer screen shot of a task

Description automatically generated

**Task 05**

**Task 05**

#include <iostream>

#include <algorithm>

using namespace std;

class Heap {

    public:

        int \*arr;

        int size;

        Heap (int arr[], int size) {

            this->arr = arr;

            this->size = size;

        }

        void maxHeapify(int i) {

            int largest = i;

            int leftChild = (2 \* i) + 1;

            int rightChild = (2 \* i) + 2;

            if (leftChild < size && arr[leftChild] > arr[largest]) {

                largest = leftChild;

            }

            if (rightChild < size && arr[rightChild] > arr[largest]) {

                largest = rightChild;

            }

            if (largest != i) {

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

                maxHeapify(largest);

            }

        }

        void buildMaxHeap() {

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

                maxHeapify(i);

            }

        }

        void maxHeapSort() {

            buildMaxHeap();

            int heapSize = size;

            for (int i = heapSize - 1; i > 0; i--) {

                swap(arr[0], arr[i]);

                heapSize--;

                maxHeapify(0);

            }

        }

        void printArray() {

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

                cout << arr[i] << " ";

            }

            cout << endl;

        }

};

int main() {

    int arr[] = {1, 2, 10, 3, 5, 4, 9};

    int n = sizeof(arr) / sizeof(arr[0]);

    Heap heap(arr, n);

    cout << "Original Array:" << endl;

    heap.printArray();

    heap.maxHeapSort();

    cout << "\nSorted Array:" << endl;

    heap.printArray();

    return 0;

}

A screenshot of a computer program

Description automatically generated

**Task 06**

#include <iostream>

using namespace std;

class MaxHeap {

    public:

        int \*arr;

        int size;

        int capacity;

        MaxHeap(int cap) {

            size = 0;

            capacity = cap;

            arr = new int[capacity];

        }

        ~MaxHeap() {

            delete[] arr;

        }

        void heapify(int i) {

            int largest = i;

            int leftChild = 2 \* i + 1;

            int rightChild = 2 \* i + 2;

            if (leftChild < size && arr[leftChild] > arr[largest]) {

                largest = leftChild;

            }

            if (rightChild < size && arr[rightChild] > arr[largest]) {

                largest = rightChild;

            }

            if (largest != i) {

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

                heapify(largest);

            }

        }

        void insert(int value) {

            if (size == capacity) {

                cout << "Heap is full!" << endl;

                return;

            }

            arr[size] = value;

            size++;

            int i = size - 1;

            while (i != 0 && arr[(i - 1) / 2] < arr[i]) {

                swap(arr[(i - 1) / 2], arr[i]);

                i = (i - 1) / 2;

            }

        }

        int extractMax() {

            if (size <= 0) {

                return -1;

            }

            int max = arr[0];

            arr[0] = arr[size - 1];

            size--;

            heapify(0);

            return max;

        }

        int getSize() {

            return size;

        }

};

int minOperationsToMakeNonIncreasing(int\* a, int n) {

    int operations = 0;

    for (int i = 1; i < n; i++) {

        if (a[i] > a[i - 1]) {

            operations += a[i] - a[i - 1];

            a[i] = a[i - 1];

        }

    }

    return operations;

}

int main() {

    int N = 4;

    int array[] = {3, 1, 2, 1};

    int result = minOperationsToMakeNonIncreasing(array, N);

    cout << "Minimum operations required: " << result << endl;

    return 0;

}

A black screen with white text

Description automatically generated