**Question#1**

**Code:**

#include<iostream>

using namespace std;

class Heap {

private:

int\* heapArr;

int size;

int count;

void swap(int& a, int& b) {

int c = a;

a = b;

b = c;

}

void heapifyUp(int index) {

while (index > 1 && heapArr[index] > heapArr[index / 2]) {

swap(heapArr[index], heapArr[index / 2]);

index = index / 2;

}

}

void heapifyDown(int index) {

int largest = index;

int leftChild = 2 \* index;

int rightChild = 2 \* index + 1;

if (leftChild < count && heapArr[leftChild] > heapArr[largest]) {

largest = leftChild;

}

if (rightChild < count && heapArr[rightChild] > heapArr[largest]) {

largest = rightChild;

}

if (largest != index) {

swap(heapArr[index], heapArr[largest]);

heapifyDown(largest);

}

}

public:

Heap(int val) {

size = val;

heapArr = new int[size + 1];

count = 1;

}

bool isFull() {

if (count == size) {

return true;

}

return false;

}

bool isEmpty() {

if (count == 1) {

return true;

}

return false;

}

void insert(int val) {

if (isFull()) {

cout << "Heap is Full. Can Not Insert" << endl;

return;

}

heapArr[count] = val;

heapifyUp(count);

count++;

}

int removeMax() {

if (isEmpty()) {

cout << "Heap is Empty. Can Not Remove" << endl;

return -1;

}

int maxVal = heapArr[1];

heapArr[1] = heapArr[--count];

heapifyDown(1);

return maxVal;

}

void display() {

if (isEmpty()) {

cout << "No patients in the queue." << endl;

return;

}

for (int i = 1; i < count; i++) {

cout << "Priority : " << heapArr[i] << endl;

}

}

void merge(Heap& other) {

for (int i = 1; i < other.count; i++) {

if (!isFull()) {

insert(other.heapArr[i]);

}

else {

cout << "Cannot Merge. Heap is Full" << endl;

break;

}

}

}

~Heap() {

delete[] heapArr;

}

};

int main() {

int size;

cout << "Enter The Size of The Emergency Room Queue : ";cin >> size;

Heap emergencyRoom(size);

int choice;

do {

cout << "Emergency Room Menu :" << endl;

cout << "1. Add Patient" << endl;

cout << "2. Admit Highest Priority Patient" << endl;

cout << "3. Display Patients" << endl;

cout << "4. Merge Queues" << endl;

cout << "5. Exit" << endl;

cout << "Enter Your Choice : ";cin >> choice;

switch (choice) {

case 1: {

int priority;

cout << "Enter Patient Priority : ";cin >> priority;

emergencyRoom.insert(priority);

break;

}

case 2: {

int admitted = emergencyRoom.removeMax();

if (admitted != -1) {

cout << "Admitted Patient With Priority : " << admitted << endl;

}

break;

}

case 3:

emergencyRoom.display();

break;

case 4: {

int otherSize;

cout << "Enter The Size of The Other Queue to Merge : "; cin >> otherSize;

Heap otherQueue(otherSize);

cout << "Enter Priorities For Other Queue : " << endl;

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

int priority;

cout << "Enter Priority : "; cin >> priority;

otherQueue.insert(priority);

}

emergencyRoom.merge(otherQueue);

break;

}

case 5:

cout << "Exiting..." << endl;

break;

default:

cout << "Invalid Input" << endl;

}

} while (choice != 5);

return 0;

}

**Output:**

**A screenshot of a computer

Description automatically generated**

**Question#2**

**Code:**

#include <iostream>

using namespace std;

class MinHeap {

private:

int\* heapArr;

int size;

int count;

void swap(int& a, int& b) {

int c = a;

a = b;

b = c;

}

void heapifyUp(int index) {

while (index > 1 && heapArr[index] < heapArr[index / 2]) {

swap(heapArr[index], heapArr[index / 2]);

index = index / 2;

}

}

void heapifyDown(int index) {

int smallest = index;

int leftChild = 2 \* index;

int rightChild = 2 \* index + 1;

if (leftChild < count && heapArr[leftChild] < heapArr[smallest]) {

smallest = leftChild;

}

if (rightChild < count && heapArr[rightChild] < heapArr[smallest]) {

smallest = rightChild;

}

if (smallest != index) {

swap(heapArr[index], heapArr[smallest]);

heapifyDown(smallest);

}

}

public:

MinHeap(int val) {

size = val;

heapArr = new int[size + 1];

count = 1;

}

void insert(int val) {

if (count == size + 1) {

cout << "Heap is Full. Cannot Insert." << endl;

return;

}

heapArr[count] = val;

heapifyUp(count);

count++;

}

int extractMin() {

if (count == 1) {

cout << "Heap is Empty" << endl;

return -1;

}

int minVal = heapArr[1];

heapArr[1] = heapArr[--count];

heapifyDown(1);

return minVal;

}

int getKthMin(int k) {

MinHeap tempHeap(size);

for (int i = 1; i < count; i++) {

tempHeap.insert(heapArr[i]);

}

int kthMin;

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

kthMin = tempHeap.extractMin();

}

return kthMin;

}

void display() {

for (int i = 1; i <= size; i++) {

cout << heapArr[i] << " ";

}

cout << endl;

}

~MinHeap() {

delete[] heapArr;

}

};

int main() {

int tasks[10] = { 15, 3, 8, 10, 2, 7, 6, 4, 9, 1 };

MinHeap taskHeap(10);

for (int i = 0; i < 10; i++) {

taskHeap.insert(tasks[i]);

}

cout << "Min Heap : ";

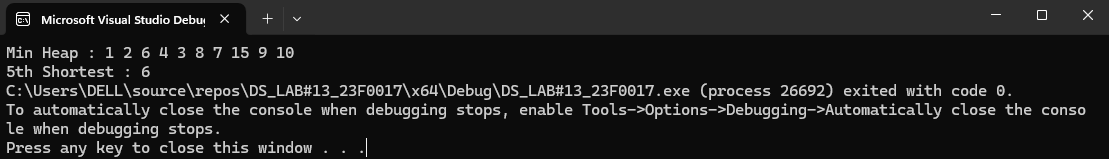
taskHeap.display();

cout << "5th Shortest : " << taskHeap.getKthMin(5);

return 0;

}

**Output:**

****

**Question#3**

**Code:**

#include <iostream>

using namespace std;

class Heap {

private:

int\* heapArr;

int size;

int count;

void heapifyDown(int index) {

int largest = index;

int leftChild = 2 \* index;

int rightChild = 2 \* index + 1;

if (leftChild <= count && heapArr[leftChild] > heapArr[largest]) {

largest = leftChild;

}

if (rightChild <= count && heapArr[rightChild] > heapArr[largest]) {

largest = rightChild;

}

if (largest != index) {

swap(heapArr[index], heapArr[largest]);

heapifyDown(largest);

}

}

public:

Heap(int val) {

size = val;

heapArr = new int[size + 1];

count = 0;

}

void buildHeap(int arr[]) {

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

heapArr[i + 1] = arr[i];

}

count = size;

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

heapifyDown(i);

}

}

void heapsort() {

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

swap(heapArr[1], heapArr[i]);

count--;

heapifyDown(1);

}

}

void display() {

for (int i = 1; i <= size; i++) {

cout << heapArr[i] << " ";

}

cout << endl;

}

~Heap() {

delete[] heapArr;

}

};

int main() {

int salaries[10] = { 55000, 75000, 120000, 95000, 60000, 70000, 80000, 100000, 85000, 95000 };

int size = 10;

Heap salaryHeap(size);

salaryHeap.buildHeap(salaries);

cout << "Sorted Salaries (Heapsort) : ";

salaryHeap.heapsort();

salaryHeap.display();

return 0;

}

**Output:**

**A screenshot of a computer

Description automatically generated**