TASK 1:

#include <iostream>

using namespace std;

class heap{

private:

int\* arr;

int size, count;

void mini\_heapify\_down(int root) {

int smallest = root;

int left = (root \* 2) + 1;

int right = (root \* 2) + 2;

if (left < count && arr[left] < arr[smallest]){

smallest = left;

}

else if (right < count && arr[right] < arr[smallest]){

smallest = right;

}

if (smallest != root){

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

mini\_heapify\_down(smallest);

}

}

void mini\_heapify\_up(int root) {

int smallest = root;

int parent = (root - 1) / 2;

if (arr[parent] > arr[smallest]){

smallest = parent;

}

if (smallest != root){

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

mini\_heapify\_up(smallest);

}

}

public:

heap(int s){

size = s;

arr = new int[size];

count = 0;

}

bool isEmpty(){

return (count == -1);

}

bool isFull(){

return (count == size);

}

int getMini(){

if (isEmpty()){

return -1;

}

return arr[0];

}

int extractMin(){

count--;

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

mini\_heapify\_down(0);

return arr[count];

}

void insert(int value){

if (isFull()){

cout << "Heap is Full" << endl;

}

else{

count++;

arr[count - 1] = value;

mini\_heapify\_up(count - 1);

}

}

void display(){

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

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

}

}

};

int main(){

int size, val;

int choice;

int n;

cout << "Enter size: ";

cin >> size;

heap H(size);

do {

cout << "1- Insert" << endl;

cout << "2- Display" << endl;

cout << "3- Extraction" << endl;

cout << "4- Get Minimum" << endl;

cin >> choice;

switch (choice) {

case 1:

cout << "Enter Values: " << endl;

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

cin >> val;

H.insert(val);

}

break;

case 2:

cout << endl << "\nHeap is : ";

H.display();

cout << endl;

break;

case 3:

H.extractMin();

cout << endl << "Heap after extraction : ";

H.display();

cout << endl;

break;

case 4:

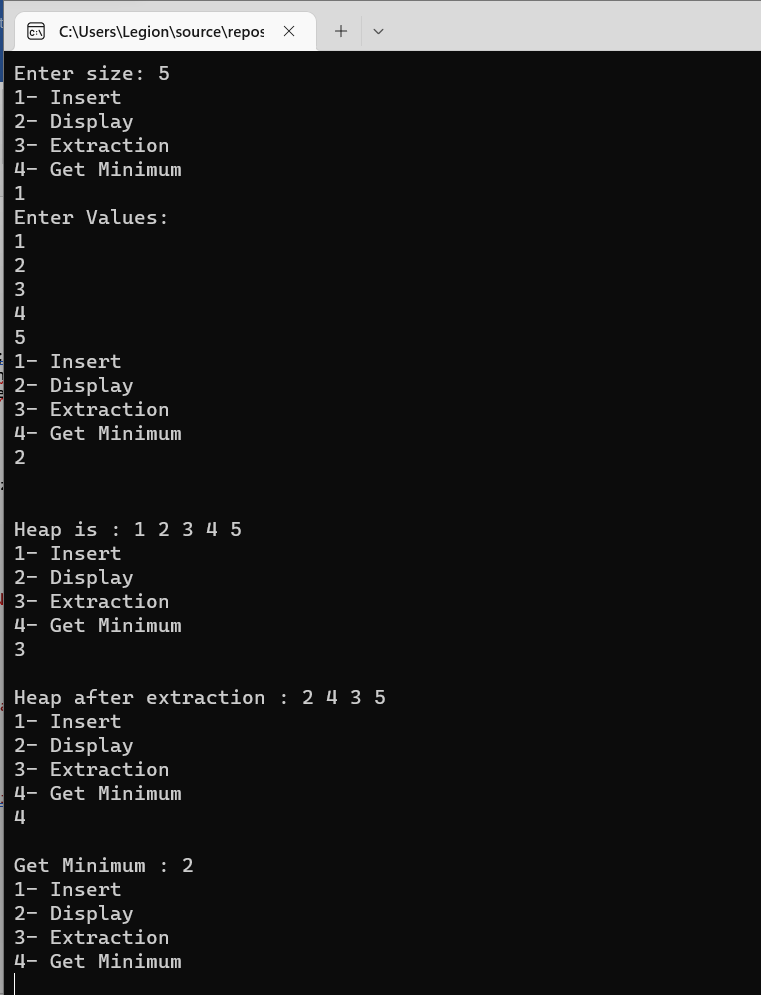
cout << endl << "Get Minimum : " << H.getMini() << endl;

break;

}

} while (choice != 7);

}



TASK 2:

#include <iostream>

using namespace std;

void func(int arr[], int size, int root){

int largest = root; int l = 2 \* root + 1; int r = 2 \* root + 2;

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

largest = l;

}

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

largest = r;

}

if (largest != root){

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

func(arr, size, largest);

}

}

void sorting(int arr[], int size){

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

func(arr, size, i);

}

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

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

func(arr, i, 0);

}

}

void displayArray(int arr[], int size){

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

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

} cout << endl;

}

int main(){

int array[] = {22,19,20,10,8};

int size = sizeof(array) / sizeof(array[0]);

cout << "Array is :" << endl;

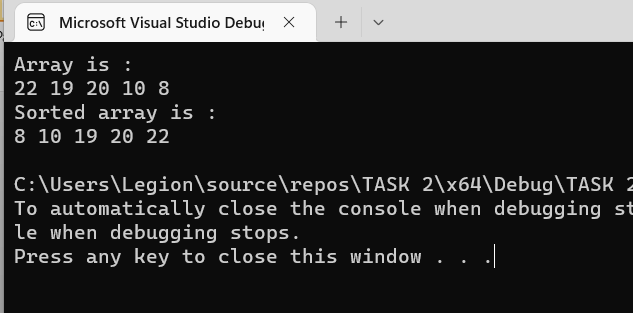
displayArray(array, size);

sorting(array, size);

cout << "Sorted array is :" << endl;

displayArray(array, size);

}



TASK 4:

#include <iostream>

using namespace std;

class AVLnode{

public:

int data;

AVLnode\* left;

AVLnode\* right;

AVLnode(){

left = right = NULL;

}

};

class AVL{

private:

AVLnode\* root;

int\* arr;

int arr\_size, arr\_count, arr\_rev;

int count;

void insert(int value, AVLnode\*& Node){

if (Node == NULL){

Node = new AVLnode;

Node->data = value;

count++;

}

else if (value < Node->data){

insert(value, Node->left);

}

else{

insert(value, Node->right);

}

/\*Node = Rotating(Node);\*/

}

AVLnode\* rrRotation(AVLnode\*& Node){

AVLnode\* temp;

temp = Node->right;

Node->right = temp->left;

temp->left = Node;

return temp;

}

AVLnode\* llRotation(AVLnode\*& Node){

AVLnode\* temp;

temp = Node->left;

Node->left = temp->right;

temp->right = Node;

return temp;

}

AVLnode\* rlRotation(AVLnode\*& Node){

AVLnode\* temp;

AVLnode\* temp2;

temp = Node->right;

temp2 = Node->right->left;

Node->right = temp2->left;

temp->left = temp2->right;

temp2->left = Node;

temp2->right = temp;

return temp2;

}

AVLnode\* lrRotation(AVLnode\*& Node){

AVLnode\* temp;

AVLnode\* temp2;

temp = Node->left;

temp2 = Node->left->right;

Node->left = temp2->right;

temp->right = temp2->left;

temp2->right = Node;

temp2->left = temp;

return temp2;

}

AVLnode\* Rotating(AVLnode\* Node){

if (balance(Node) == 2){

if (balance(Node->left) == 1){

Node = llRotation(Node);

return Node;

}

else if (balance(Node->left) == -1){

Node = lrRotation(Node);

return Node;

}

}

else if (balance(Node) == -2){

if (balance(Node->right) == 1){

Node = rlRotation(Node);

return Node;

}

else if (balance(Node->right) == -1){

Node = rrRotation(Node);

return Node;

}

}

return Node;

}

public:

AVL(int s){

root = NULL;

count = 0;

arr\_size = s;

arr\_count = 0;

arr = new int[s];

}

bool isEmpty(){

return(root == NULL);

}

void insertion(int value){

insert(value, root);

}

int balance(AVLnode\* Node){

if (Node == NULL){

return 0;

}

return (height(Node->left) - height(Node->right));

}

AVLnode\* getRoot(){

return root;

}

void inOrder\_traversal(AVLnode\* p){

if (p != NULL){

inOrder\_traversal(p->left);

arr[arr\_count] = p->data;

arr\_count++;

cout << p->data << " ";

inOrder\_traversal(p->right);

}

}

void preOrder\_traversal(AVLnode\* p){

if (p != NULL){

p->data = arr[arr\_rev];

arr\_rev++;

preOrder\_traversal(p->left);

preOrder\_traversal(p->right);

}

}

int height(AVLnode\* temp){

if (temp == NULL){

return -1;

}

int left = height(temp->left);

int right = height(temp->right);

if (left >= right){

return left + 1;

}

else{

return right + 1;

}

}

void heap\_display(AVLnode\* p){

if (p != NULL){

heap\_display(p->left);

cout << p->data << " ";

heap\_display(p->right);

}

}

};

int main(){

int size, value;

cout << "Enter size : ";

cin >> size;

AVL obj(size);

cout << "Insert value : " << endl;

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

cin >> value;

obj.insertion(value);

}

cout << "BST : ";

obj.inOrder\_traversal(obj.getRoot());

cout << endl;

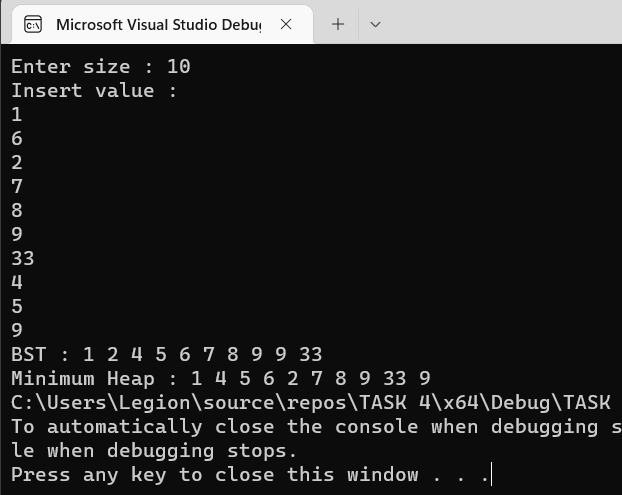
obj.preOrder\_traversal(obj.getRoot());

cout << "Minimum Heap : ";

obj.heap\_display(obj.getRoot());

return 0;

}



TASK 5:

#include<iostream>

using namespace std;

void swap(int\* a, int\* b){

int temp = \*a;

\*a = \*b;

\*b = temp;

}

void max(int arr[], int i, int num){

int left = 2 \* i + 1;

int right = 2 \* i + 2;

int largest = i;

if (left < num && arr[left] > arr[i]) {

largest = left;

}

if (right < num && arr[right] > arr[largest]) {

largest = right;

}

if (largest != i) {

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

max(arr, largest, num);

}

}

void display(int\* arr, int size){

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

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

}

void conversion(int arr[], int N) {

int i = (N - 2) / 2;

while (i >= 0) {

max(arr, i, N);

--i;

}

}

int main(){

int arr[10];

cout << "Enter Elements in array" << endl;

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

cin >> arr[i];

}

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

cout << "Min Heap : ";

display(arr, size);

cout << endl;

conversion(arr, size);

cout << "Max Heap : ";

display(arr, size);

}

