Q1

#include <iostream>

using namespace std;

template <typename t>

class queue

{

public:

t data;

queue<t> \*next;

queue(t d)

{

this->data = d;

this->next = NULL;

}

void push(queue<t> \*&q, t d)

{

queue<t> \*n = new queue<t>(d);

if (q == NULL)

{

q = n;

return;

}

else

{

queue<t> \*temp = q;

while (temp->next != NULL)

{

temp = temp->next;

}

temp->next = n;

return;

}

}

void pop(queue<t> \*&q)

{

if (q != NULL)

{

queue<t> \*te = q;

q = q->next;

delete te;

}

}

t top(queue<t> \*&q)

{

if (q != NULL)

{

return q->data;

}

return NULL;

}

bool isempty(queue<t> \*&q)

{

return q == NULL ? true : false;

}

};

class node

{

public:

int data;

node \*left;

node \*right;

node(int d)

{

this->data = d;

this->left = NULL;

this->right = NULL;

}

node \*insert\_into\_bst(node \*root, int data)

{

if (root == NULL)

return new node(data);

if (root->data < data)

{

root->right = insert\_into\_bst(root->right, data);

}

else if (root->data > data)

{

root->left = insert\_into\_bst(root->left, data);

}

else

{

cout << "\nduplicate data .\n";

return root;

}

}

void inorder(node \*root)

{

if (root == NULL)

return;

inorder(root->left);

cout << root->data << " ";

inorder(root->right);

}

void insert(node \*&root)

{

int data;

cout << "enter the data : (-1 to stop) ";

cin >> data;

while (data != -1)

{

root = insert\_into\_bst(root, data);

cout << "enter the data : (-1 to stop) ";

cin >> data;

}

}

void level\_order(node \*root)

{

if (root == NULL)

{

cout << "\nNULL";

return;

}

queue<node \*> \*q = NULL;

q->push(q, root);

q->push(q, NULL);

while (!q->isempty(q))

{

node \*f = q->top(q);

q->pop(q);

if (f == NULL)

{

if (!q->isempty(q))

{

q->push(q, NULL);

}

cout << endl;

}

else

{

cout << f->data << " ";

if (f->left)

{

q->push(q, f->left);

}

if (f->right)

{

q->push(q, f->right);

}

}

}

}

int min\_value(node \*root)

{

node \*r = root->right;

while (r->left != NULL)

{

r = r->left;

}

return r->data;

}

node \*deletion(node \* root, int d)

{

if (root == NULL)

return NULL;

if (root->data < d)

root->right = deletion(root->right, d);

else if (root->data > d)

root->left = deletion(root->left, d);

else if(root->data==d)

{

if (root->left == NULL)

{

node \*a = root->right;

delete root;

return a;

}

else if (root->right == NULL)

{

node \*a = root->left;

delete root;

return a;

}

else if (root->right != NULL && root->left != NULL)

{

int x = min\_value(root);

root->data = x;

root->right = deletion(root->right, x);

return root;

}

}

}

};

int main()

{

// queue<int> \*q = NULL;

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

// {

// q->push(q, i + 1);

// }

// while (!q->isempty(q))

// {

// cout << q->top(q);

// q->pop(q);

// }

node \*root = NULL;

root->insert(root);

cout << "\nlevel order \n";

root->level\_order(root);

int data;

cout << "\nenter the data for deletion : (-1 to stop) ";

cin >> data;

while (data != -1)

{

root = root->deletion(root, data);

cout << "\nlevel order after deletion :";

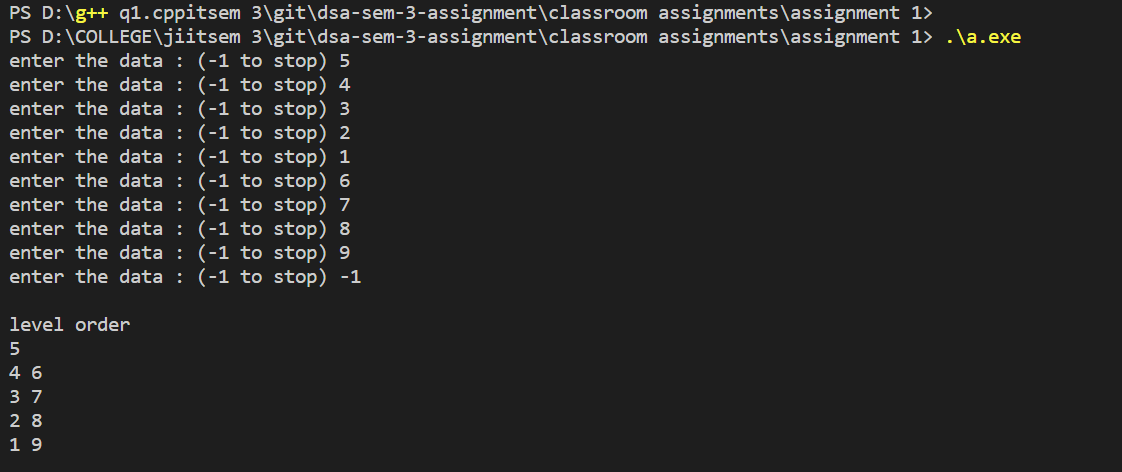
root->level\_order(root);

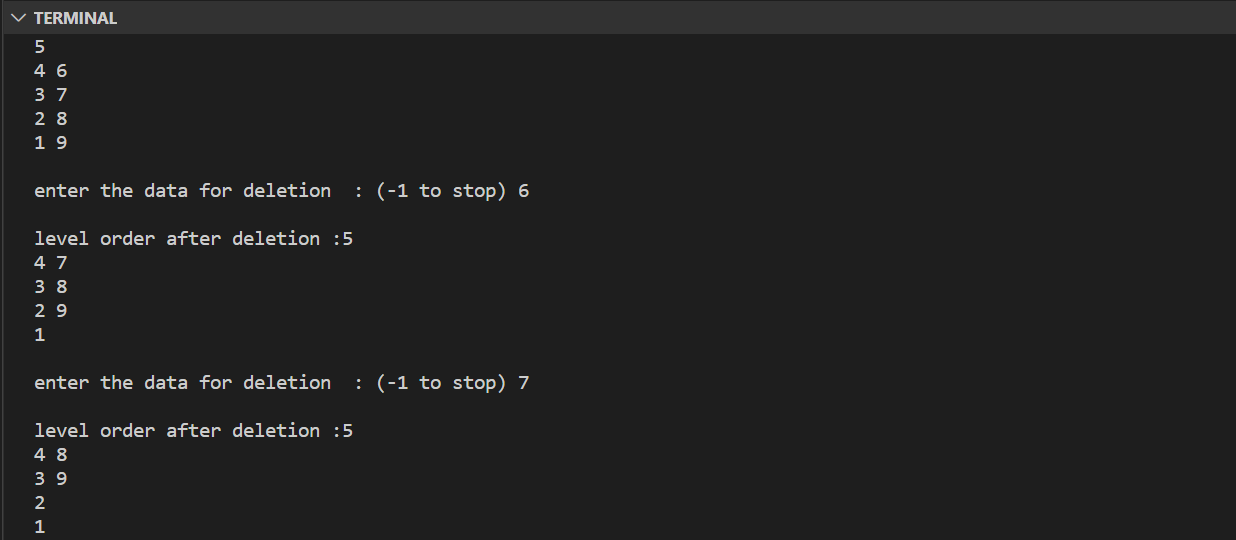
cout << "\nenter the data for deletion : (-1 to stop) ";

cin >> data;

}

}





Q2

#include <iostream>

using namespace std;

template <typename t>

class queue

{

public:

t data;

queue<t> \*next;

queue(t d)

{

this->data = d;

this->next = NULL;

}

void push(queue<t> \*&q, t d)

{

queue<t> \*n = new queue<t>(d);

if (q == NULL)

{

q = n;

return;

}

else

{

queue<t> \*temp = q;

while (temp->next != NULL)

{

temp = temp->next;

}

temp->next = n;

return;

}

}

void pop(queue<t> \*&q)

{

if (q != NULL)

{

queue<t> \*te = q;

q = q->next;

delete te;

}

}

t top(queue<t> \*&q)

{

if (q != NULL)

{

return q->data;

}

return NULL;

}

bool isempty(queue<t> \*&q)

{

return q == NULL ? true : false;

}

};

class node

{

public:

int data;

node \*left;

node \*right;

node(int d)

{

this->data = d;

this->left = NULL;

this->right = NULL;

}

void height(node \*root , int h,int & ans)

{

if(root==NULL) {ans=max(ans,h); return ;}

height(root->left,h+1,ans);

height(root->right,h+1,ans);

}

int height\_cal(node \* root)

{

int ans=0;

height(root,0,ans);

return ans;

}

int get\_balance\_factor(node \* root)

{

if(root==NULL) return -1;

return height\_cal(root->left)-height\_cal(root->right);

}

node \* left\_r(node \* x)

{

node \* y=x->left;

node \* b=y->right;

y->right=x;

x->left=b;

return y;

}

node \* right\_r(node \* x)

{

node \* y=x->right;

node \* b=y->left;

y->left=x;

x->right=b;

return y;

}

node \*insert\_into\_avl(node \*root, int data)

{

if (root == NULL)

return new node(data);

if (root->data < data)

{

root->right = insert\_into\_avl(root->right, data);

}

else if (root->data > data)

{

root->left = insert\_into\_avl(root->left, data);

}

else

{

cout << "\nduplicate data .\n";

return root;

}

int bf=get\_balance\_factor(root);

if(bf>1 && get\_balance\_factor(root->left)>=0)

{

return left\_r(root);

}

if(bf>1 && get\_balance\_factor(root->left)<0)

{

root->left=right\_r(root->left);

return left\_r(root);

}

if(bf<-1 && get\_balance\_factor(root->right)<=0)

{

return right\_r(root);

}

if(bf<-1 && get\_balance\_factor(root->right)>0)

{

root->right=left\_r(root->right);

return right\_r(root);

}

return root;

}

void inorder(node \*root)

{

if (root == NULL)

return;

inorder(root->left);

cout << root->data << " ";

inorder(root->right);

}

void insert(node \*&root)

{

int data;

cout << "enter the data : (-1 to stop) ";

cin >> data;

while (data != -1)

{

root = insert\_into\_avl(root, data);

cout << "enter the data : (-1 to stop) ";

cin >> data;

}

}

void level\_order(node \*root)

{

if (root == NULL)

{

cout << "\nNULL";

return;

}

queue<node \*> \*q = NULL;

q->push(q, root);

q->push(q, NULL);

while (!q->isempty(q))

{

node \*f = q->top(q);

q->pop(q);

if (f == NULL)

{

if (!q->isempty(q))

{

q->push(q, NULL);

}

cout << endl;

}

else

{

cout << f->data << " ";

if (f->left)

{

q->push(q, f->left);

}

if (f->right)

{

q->push(q, f->right);

}

}

}

}

int min\_value(node \*root)

{

node \*r = root->right;

while (r->left != NULL)

{

r = r->left;

}

return r->data;

}

node \*deletion(node \* root, int d)

{

if (root == NULL)

return NULL;

if (root->data < d)

root->right = deletion(root->right, d);

else if (root->data > d)

root->left = deletion(root->left, d);

else if(root->data==d)

{

if (root->left == NULL)

{

node \*a = root->right;

delete root;

return a;

}

else if (root->right == NULL)

{

node \*a = root->left;

delete root;

return a;

}

else if (root->right != NULL && root->left != NULL)

{

int x = min\_value(root);

root->data = x;

root->right = deletion(root->right, x);

//return root;

}

}

if(root==NULL) return root;

int bf=get\_balance\_factor(root);

if(bf>1 && get\_balance\_factor(root->left)>=0)

{

return left\_r(root);

}

if(bf>1 && get\_balance\_factor(root->left)<0)

{

root->left=right\_r(root->left);

return left\_r(root);

}

if(bf<-1 && get\_balance\_factor(root->right)<=0)

{

return right\_r(root);

}

if(bf<-1 && get\_balance\_factor(root->right)>0)

{

root->right=left\_r(root->right);

return right\_r(root);

}

return root;

}

};

int main()

{

// queue<int> \*q = NULL;

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

// {

// q->push(q, i + 1);

// }

// while (!q->isempty(q))

// {

// cout << q->top(q);

// q->pop(q);

// }

node \*root = NULL;

root->insert(root);

cout << "\nlevel order \n";

root->level\_order(root);

int data;

cout << "\nenter the data for deletion : (-1 to stop) ";

cin >> data;

while (data != -1)

{

root = root->deletion(root, data);

cout << "\nlevel order after deletion :\n";

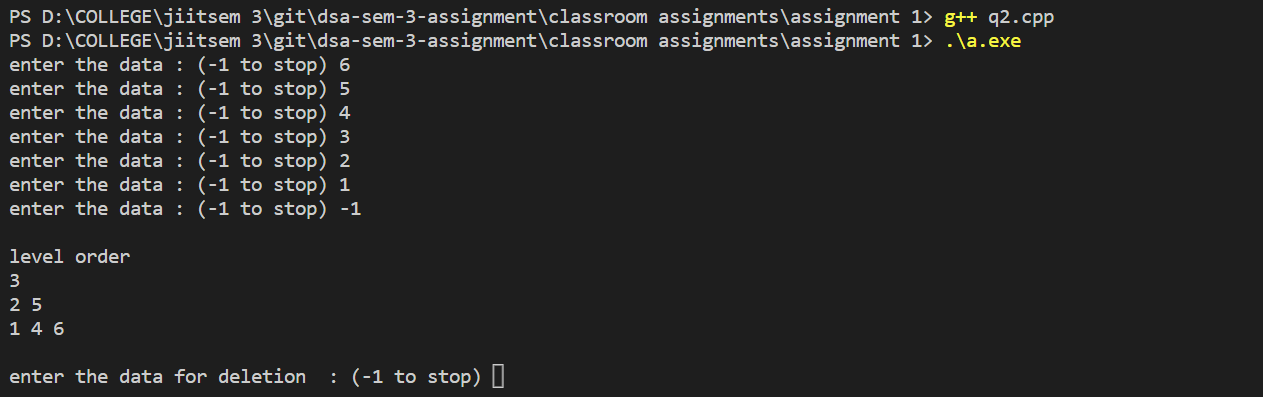
root->level\_order(root);

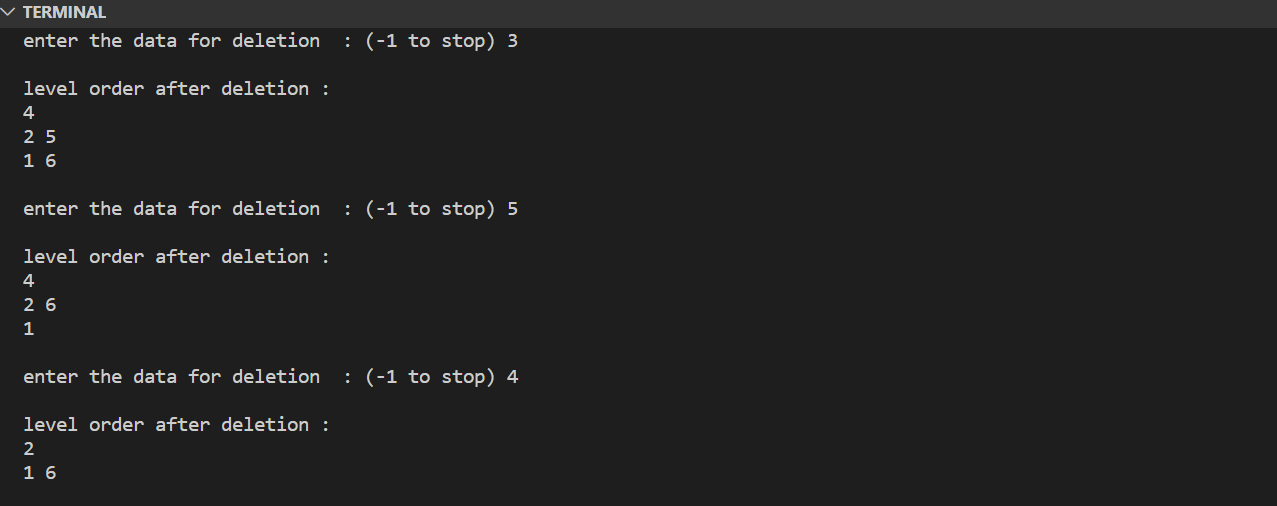
cout << "\nenter the data for deletion : (-1 to stop) ";

cin >> data;

}

}





Q3

#include <iostream>

using namespace std;

#include <queue>

class max\_heap

{

public:

int \*arr;

int capacity;

int size;

max\_heap(int c)

{

this->arr = new int[c];

this->capacity = c;

this->size = 0;

}

int parent(int i) { return (i - 1) / 2; }

int left\_child(int i) { return (2 \* i) + 1; }

int right\_child(int i) { return (2 \* i) + 2; }

void insert(int data)

{

if (size == capacity)

{

cout << "overflow \n";

return;

}

arr[size] = data;

int i = size;

size++;

while (i >= 1)

{

if (arr[parent(i)] < arr[i])

{

int temp = arr[parent(i)];

arr[parent(i)] = arr[i];

arr[i] = temp;

i = parent(i);

}

else

break;

}

}

void deletion()

{

if (size < 0)

{

cout << "NOTHING TO DELETE\n";

return;

}

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

size--;

int i = 0;

while (i < size)

{

int largest = i;

int l = left\_child(i);

int r = right\_child(i);

if (l < size && arr[i] < arr[l])

{

largest = l;

}

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

{

largest = r;

}

if (largest != i)

{

int temp = arr[i];

arr[i] = arr[largest];

arr[largest] = temp;

i = largest;

}

else

return;

}

}

void print()

{

queue<int> q;

if (size == 0)

{

cout << "NULL";

return;

}

q.push(0);

q.push(-1);

while (!q.empty())

{

int i = q.front();

q.pop();

if (i == -1)

{

if (!q.empty())

q.push(-1);

cout << "\n";

}

else

{

if (i < size)

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

if (left\_child(i) < size)

q.push(left\_child(i));

if (right\_child(i) < size)

q.push(right\_child(i));

}

}

}

int max\_element()

{

if (size >= 0)

return arr[0];

return -1;

}

void increase\_key(int key, int value)

{

if (size < 0)

{

cout << "\n no element : \n ";

return;

}

int i;

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

{

if (arr[i] == key)

{

break;

}

}

if (i == size)

{

cout << "\n key is not found : \n ";

return;

}

arr[i] = key + value;

while (i >= 1)

{

if (arr[parent(i)] < arr[i])

{

int temp = arr[i];

arr[i] = arr[parent(i)];

arr[parent(i)] = temp;

i = parent(i);

}

else

break;

}

}

void decrease\_key(int key, int value)

{

if (size < 0)

{

cout << "\n no element : \n ";

return;

}

int i;

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

{

if (arr[i] == key)

{

break;

}

}

if (i == size)

{

cout << "\n key is not found : \n ";

return;

}

arr[i] = key - value;

while (i < size)

{

int largest = i;

int l = left\_child(i);

int r = right\_child(i);

if (l < size && arr[i] < arr[l])

{

largest = l;

}

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

{

largest = r;

}

if (largest != i)

{

int temp = arr[i];

arr[i] = arr[largest];

arr[largest] = temp;

i = largest;

}

else

break;

}

}

};

int main()

{

int n;

cout << "enter capacity : ";

cin >> n;

max\_heap \*h = new max\_heap(n);

int s;

cout << "enter data ( -1 to stop entering data) ";

cin >> s;

while (s != -1)

{

h->insert(s);

cout << endl;

h->print();

cout << endl;

cout << "enter data ( -1 to stop entering data) ";

cin >> s;

}

h->print();

cout << "enter any number for deletion ( -1 to stop entering data) ";

cin >> s;

while (s != -1)

{

h->deletion();

cout << endl;

h->print();

cout << endl;

cout << "enter any number for deletion ( -1 to stop entering data) ";

cin >> s;

}

cout << "enter key for increasing key ( -1 to stop entering data) ";

cin >> s;

while (s != -1)

{

int v;

cout << "enter value to be add on : ";

cin >> v;

h->increase\_key(s, v);

cout << endl;

h->print();

cout << endl;

cout << "enter key for increasing key ( -1 to stop entering data) ";

cin >> s;

}

cout << "enter key for decreasing key ( -1 to stop entering data) ";

cin >> s;

while (s != -1)

{

int v;

cout << "enter value to be subracted off : ";

cin >> v;

h->decrease\_key(s, v);

cout << endl;

h->print();

cout << endl;

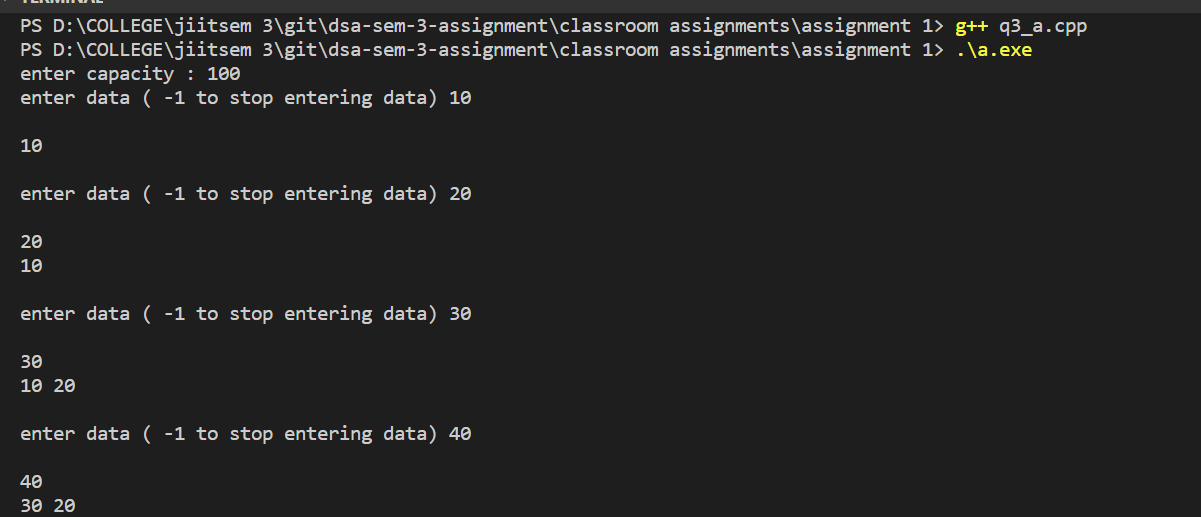
cout << "enter key for decreasing key ( -1 to stop entering data) ";

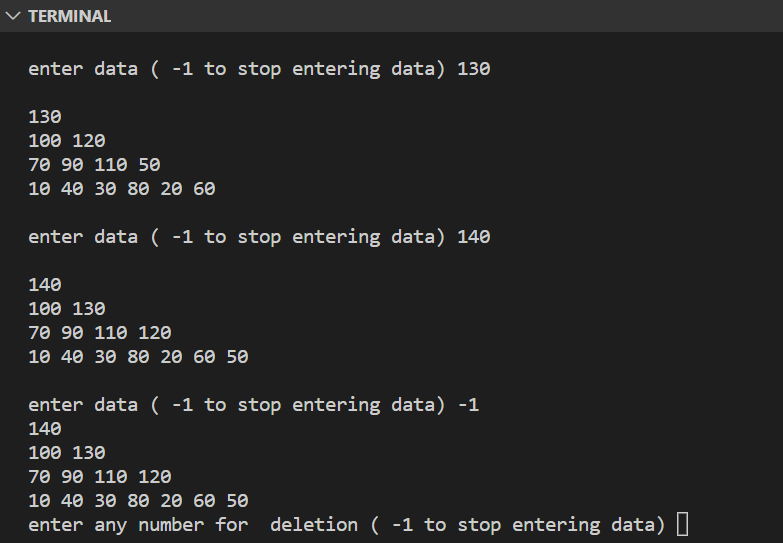
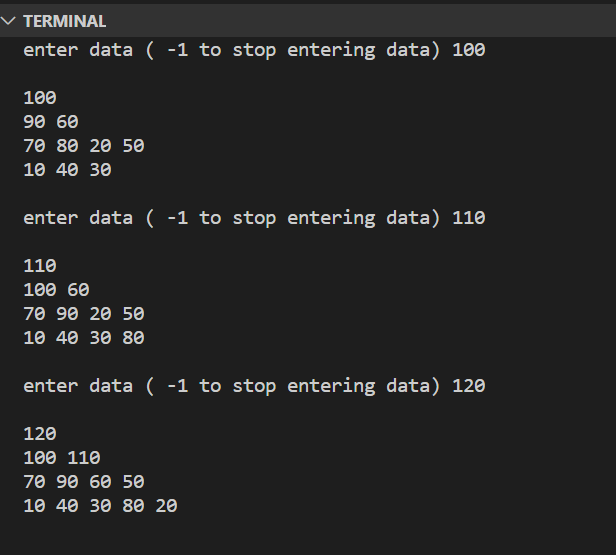
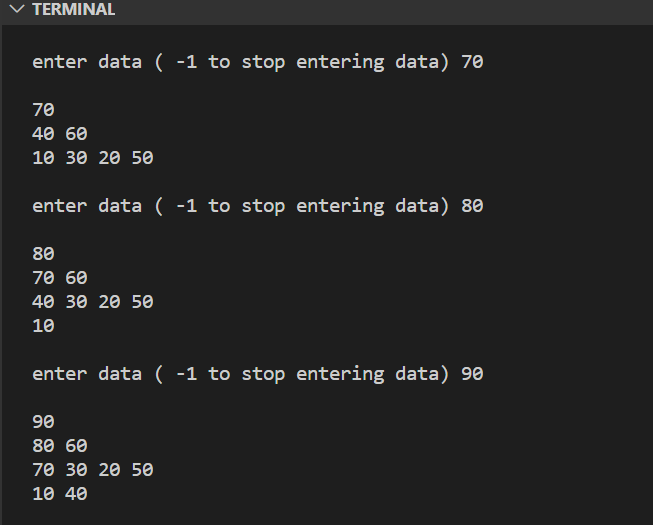
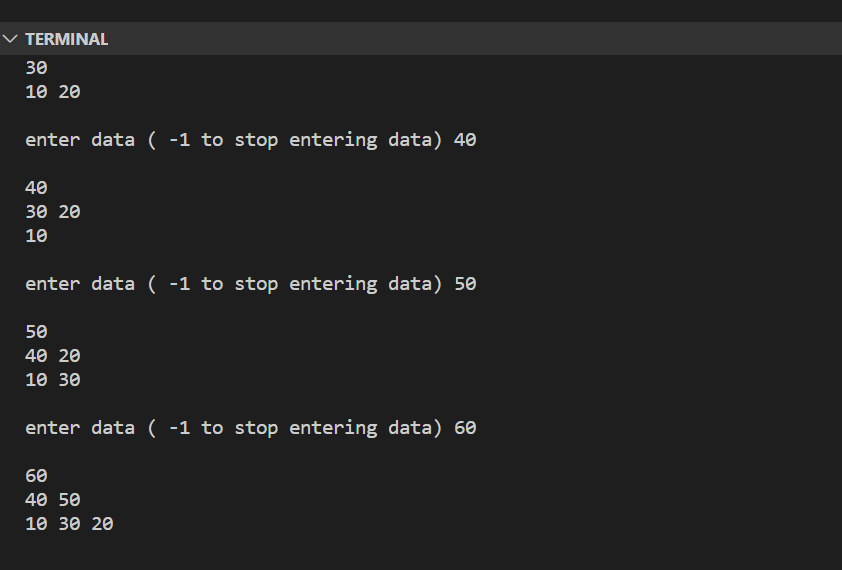
cin >> s;

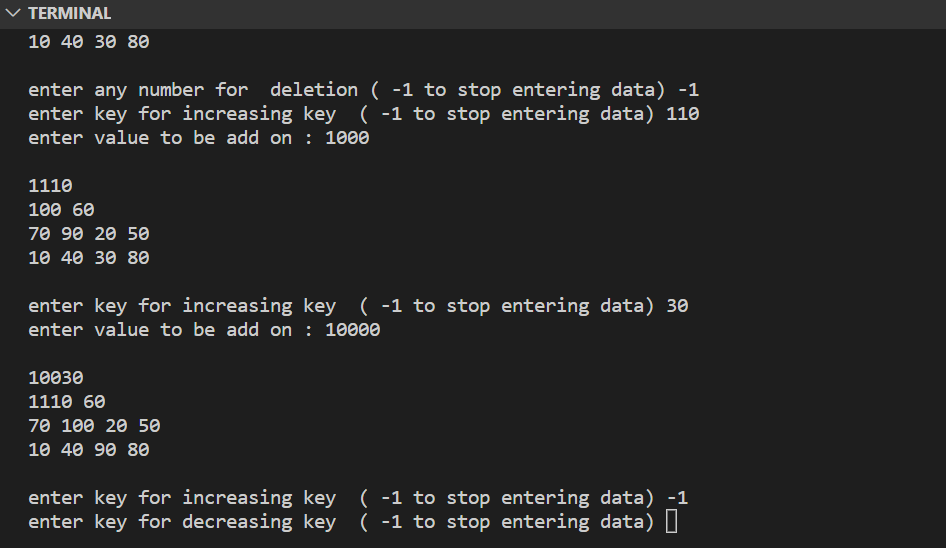
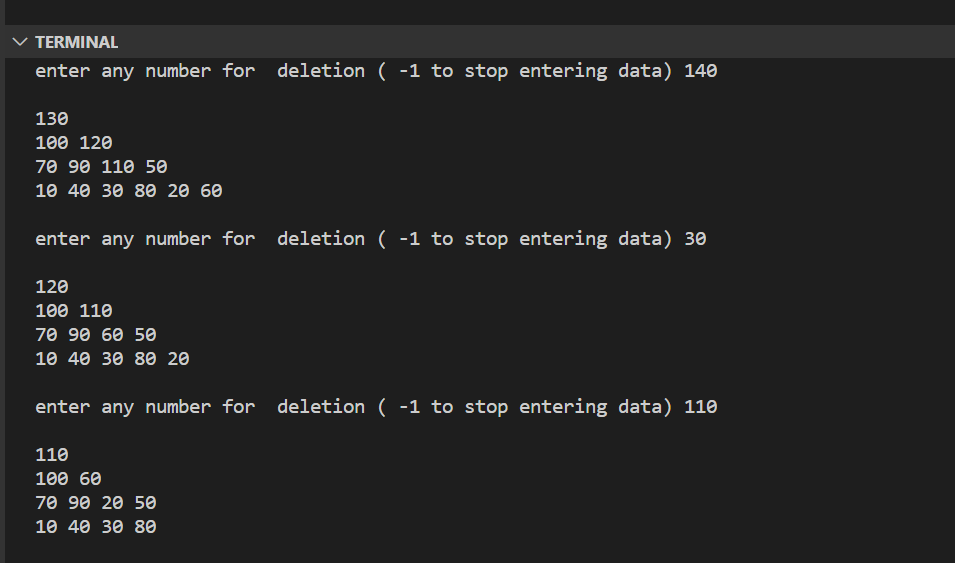
}

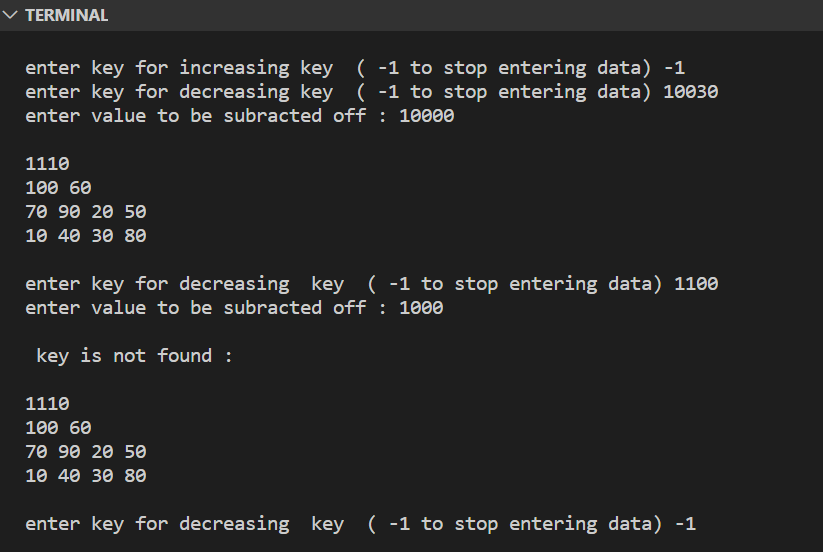
}

// 100 10 20 30 40 50 60 70 80 90 100 110 120 130 140 150 160 -1









Q4

#include <iostream>

#include <queue>

using namespace std;

class min\_heap

{

public:

int \*arr;

int size;

int capacity;

min\_heap(int c)

{

this->size = 0;

this->capacity = c;

this->arr = new int[c];

}

int parent(int i){ return (i-1)/2; }

int left\_child(int i){return (2\*i)+1;}

int right\_child(int i){return (2\*i)+2; }

void insert(int d)

{

if (size == capacity)

{

cout << "over flow : ";

return;

}

arr[size]=d;

int i=size;

size++;

while(i>=1)

{

if(arr[parent(i)]>arr[i])

{

int temp=arr[parent(i)];

arr[parent(i)]=arr[i];

arr[i]=temp;

i=parent(i);

}

else break;

}

}

void print()

{

if(size==0){

cout<<"NULL";

return ;

}

queue<int> q;

q.push(0);

q.push(-1);

while(!q.empty())

{

int i=q.front();

q.pop();

if(i==-1)

{

if(!q.empty())

{

q.push(-1);

}

cout<<endl;

}

else

{

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

if(left\_child(i)<size)

{

q.push(left\_child(i));

}

if(right\_child(i)<size)

{

q.push(right\_child(i));

}

}

}

}

void deletion()

{

if (size < 0)

{

cout << "NOTHING TO DELETE\n";

return;

}

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

size--;

int i = 0;

while (i < size)

{

int smallest = i;

int l = left\_child(i);

int r = right\_child(i);

if (l < size && arr[i] > arr[l])

{

smallest = l;

}

if (r < size && arr[smallest] > arr[r])

{

smallest = r;

}

if (smallest != i)

{

int temp = arr[i];

arr[i] = arr[smallest];

arr[smallest] = temp;

i = smallest;

}

else

return;

}

}

int min\_element()

{

if (size >= 0)

return arr[0];

return -1;

}

void increase\_key(int key, int value)

{

if (size < 0)

{

cout << "\n no element : \n ";

return;

}

int i;

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

{

if (arr[i] == key)

{

break;

}

}

if (i == size)

{

cout << "\n key is not found : \n ";

return;

}

arr[i] = key + value;

while (i < size)

{

int smallest = i;

int l = left\_child(i);

int r = right\_child(i);

if (l < size && arr[i] > arr[l])

{

smallest = l;

}

if (r < size && arr[smallest] > arr[r])

{

smallest = r;

}

if (smallest != i)

{

int temp = arr[i];

arr[i] = arr[smallest];

arr[smallest] = temp;

i = smallest;

}

else

return;

}

}

void decrease\_key(int key, int value)

{

if (size < 0)

{

cout << "\n no element : \n ";

return;

}

int i;

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

{

if (arr[i] == key)

{

break;

}

}

if (i == size)

{

cout << "\n key is not found : \n ";

return;

}

arr[i] = key - value;

while (i >= 1)

{

if (arr[parent(i)] > arr[i])

{

int temp = arr[i];

arr[i] = arr[parent(i)];

arr[parent(i)] = temp;

i = parent(i);

}

else

break;

}

}

};

int main()

{

int n;

cout << "enter capacity : ";

cin >> n;

min\_heap \*h = new min\_heap(n);

int s;

cout << "enter data ( -1 to stop entering data) ";

cin >> s;

while (s != -1)

{

h->insert(s);

cout << endl;

h->print();

cout << endl;

cout << "enter data ( -1 to stop entering data) ";

cin >> s;

}

h->print();

cout << "enter any number for deletion ( -1 to stop entering data) ";

cin >> s;

while (s != -1)

{

h->deletion();

cout << endl;

h->print();

cout << endl;

cout << "enter any number for deletion ( -1 to stop entering data) ";

cin >> s;

}

cout << "enter key for increasing key ( -1 to stop entering data) ";

cin >> s;

while (s != -1)

{

int v;

cout << "enter value to be add on : ";

cin >> v;

h->increase\_key(s, v);

cout << endl;

h->print();

cout << endl;

cout << "enter key for increasing key ( -1 to stop entering data) ";

cin >> s;

}

cout << "enter key for decreasing key ( -1 to stop entering data) ";

cin >> s;

while (s != -1)

{

int v;

cout << "enter value to be subracted off : ";

cin >> v;

h->decrease\_key(s, v);

cout << endl;

h->print();

cout << endl;

cout << "enter key for decreasing key ( -1 to stop entering data) ";

cin >> s;

}

}

// 100 10 20 30 40 50 60 70 80 90 100 110 120 130 140 150 160 -1