***NAME : Himanshu Dixit***

***ENROLL NO. : 21103262***

***BATCH : B11***

***Data Structure Lab – II [15B17CI271]***

***Lab A Week 13***

***Q1)***

*#include <iostream>*

*using namespace std;*

*void heapify(int arr[], int n, int i)*

*{*

*// Find largest among root, left child and right child*

*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;*

*// Swap and continue heapifying if root is not largest*

*if (largest != i)*

*{*

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

*heapify(arr, n, largest);*

*}*

*}*

*// main function to do heap sort*

*void heapSort(int arr[], int n)*

*{*

*// Build max heap*

*for (int i = n / 2 - 1; i >= 0; i--)*

*heapify(arr, n, i);*

*// Heap sort*

*for (int i = n - 1; i >= 0; i--)*

*{*

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

*// Heapify root element to get highest element at root again*

*heapify(arr, i, 0);*

*}*

*}*

*// Print an array*

*void printArray(int arr[], int n)*

*{*

*for (int i = 0; i < n; ++i)*

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

*cout << "\n";*

*}*

*// Driver code*

*int main()*

*{*

*int arr[] = {1, 12, 9, 5, 6, 10};*

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

*heapSort(arr, n);*

*cout << "Sorted array is \n";*

*printArray(arr, n);*

*}*

***Q2)***

*// C++ program to merge two max heaps.*

*#include <bits/stdc++.h>*

*using namespace std;*

*// Standard heapify function to heapify a*

*// subtree rooted under idx. It assumes*

*// that subtrees of nodes are already heapified.*

*void maxHeapify(int arr[], int N, int idx)*

*{*

*// Find largest of node and its children*

*if (idx >= N)*

*return;*

*int l = 2 \* idx + 1;*

*int r = 2 \* idx + 2;*

*int max = idx;*

*if (l < N && arr[l] > arr[idx])*

*max = l;*

*if (r < N && arr[r] > arr[max])*

*max = r;*

*// Put maximum value at root and*

*// recur for the child with the*

*// maximum value*

*if (max != idx)*

*{*

*swap(arr[max], arr[idx]);*

*maxHeapify(arr, N, max);*

*}*

*}*

*// Builds a max heap of given arr[0..n-1]*

*void buildMaxHeap(int arr[], int N)*

*{*

*// building the heap from first non-leaf*

*// node by calling max heapify function*

*for (int i = N / 2 - 1; i >= 0; i--)*

*maxHeapify(arr, N, i);*

*}*

*// Merges max heaps a[] and b[] into merged[]*

*void mergeHeaps(int merged[], int a[], int b[], int N,*

*int M)*

*{*

*// Copy elements of a[] and b[] one by one*

*// to merged[]*

*for (int i = 0; i < N; i++)*

*merged[i] = a[i];*

*for (int i = 0; i < M; i++)*

*merged[N + i] = b[i];*

*// build heap for the modified array of*

*// size n+m*

*buildMaxHeap(merged, N + M);*

*}*

*// Driver's code*

*int main()*

*{*

*int a[] = {10, 5, 6, 2};*

*int b[] = {12, 7, 9};*

*int N = sizeof(a) / sizeof(a[0]);*

*int M = sizeof(b) / sizeof(b[0]);*

*int merged[N + M];*

*// Function call*

*mergeHeaps(merged, a, b, N, M);*

*for (int i = 0; i < N + M; i++)*

*cout << merged[i] << " ";*

*return 0;*

*}*

***Q3)***

*// C++ implementation to merge the*

*// elements of the array until all*

*// the array element of the array*

*// greater than or equal to K*

*#include <bits/stdc++.h>*

*using namespace std;*

*// Function to find the minimum*

*// operation required to merge*

*// elements of the array*

*int minOperations(int arr[], int K,*

*int size)*

*{*

*int least, second\_least,*

*min\_operations = 0,*

*new\_ele = 0, flag = 0;*

*// Heap to store the elements*

*// of the array and to extract*

*// minimum elements of O(logN)*

*priority\_queue<int, vector<int>,*

*greater<int>>*

*heap;*

*// Loop to push all the elements*

*// of the array into heap*

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

*{*

*heap.push(arr[i]);*

*}*

*// Loop to merge the minimum*

*// elements until there is only*

*// all the elements greater than K*

*while (heap.size() != 1)*

*{*

*// Condition to check minimum*

*// element of the array is*

*// greater than the K*

*if (heap.top() >= K)*

*{*

*flag = 1;*

*break;*

*}*

*// Merge the two minimum*

*// elements of the heap*

*least = heap.top();*

*heap.pop();*

*second\_least = heap.top();*

*heap.pop();*

*new\_ele = (1 \* least) +*

*(2 \* second\_least);*

*min\_operations++;*

*heap.push(new\_ele);*

*}*

*if (*

*heap.top() >= K)*

*{*

*flag = 1;*

*}*

*if (*

*flag == 1)*

*{*

*return min\_operations;*

*}*

*return -1;*

*}*

*// Driver Code*

*int main()*

*{*

*int N = 6, K = 7;*

*int arr[] = {1, 2, 3, 9, 10, 12};*

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

*cout << minOperations(arr, K, size);*

*return 0;*

*}*

***Q4)***

*// C++ program for connecting*

*// n ropes with minimum cost*

*#include <bits/stdc++.h>*

*using namespace std;*

*// A Min Heap: Collection of min heap nodes*

*struct MinHeap*

*{*

*unsigned size; // Current size of min heap*

*unsigned capacity; // capacity of min heap*

*int \*harr; // Array of min heap nodes*

*};*

*// A utility function to create*

*// a min-heap of a given capacity*

*struct MinHeap \*createMinHeap(unsigned capacity)*

*{*

*struct MinHeap \*minHeap = new MinHeap;*

*minHeap->size = 0; // current size is 0*

*minHeap->capacity = capacity;*

*minHeap->harr = new int[capacity];*

*return minHeap;*

*}*

*// A utility function to swap two min heap nodes*

*void swapMinHeapNode(int \*a, int \*b)*

*{*

*int temp = \*a;*

*\*a = \*b;*

*\*b = temp;*

*}*

*// The standard minHeapify function.*

*void minHeapify(struct MinHeap \*minHeap, int idx)*

*{*

*int smallest = idx;*

*int left = 2 \* idx + 1;*

*int right = 2 \* idx + 2;*

*if (left < minHeap->size && minHeap->harr[left] < minHeap->harr[smallest])*

*smallest = left;*

*if (right < minHeap->size && minHeap->harr[right] < minHeap->harr[smallest])*

*smallest = right;*

*if (smallest != idx)*

*{*

*swapMinHeapNode(&minHeap->harr[smallest],*

*&minHeap->harr[idx]);*

*minHeapify(min Heap, smallest);*

*}*

*}*

*// A utility function to check*

*// if size of heap is 1 or not*

*int isSizeOne(struct MinHeap \*minHeap)*

*{*

*return (minHeap->size == 1);*

*}*

*// A standard function to extract*

*// minimum value node from heap*

*int extractMin(struct MinHeap \*minHeap)*

*{*

*int temp = min Heap->harr[0];*

*minHeap->harr[0] = minHeap->harr[minHeap->size - 1];*

*--minHeap->size;*

*minHeapify(minHeap, 0);*

*return temp;*

*}*

*// A utility function to insert*

*// a new node to Min Heap*

*void insertMinHeap(struct MinHeap \*minHeap, int val)*

*{*

*++minHeap->size;*

*int i = minHeap->size - 1;*

*while (i && (val < minHeap->harr[(i - 1) / 2]))*

*{*

*minHeap->harr[i] = minHeap->harr[(i - 1) / 2];*

*i = (i - 1) / 2;*

*}*

*minHeap->harr[i] = val;*

*}*

*// A standard function to build min-heap*

*void buildMinHeap(struct MinHeap \*minHeap)*

*{*

*int n = minHeap->size - 1;*

*int i;*

*for (i = (n - 1) / 2; i >= 0; --i)*

*minHeapify(minHeap, i);*

*}*

*// Creates a min-heap of capacity*

*// equal to size and inserts all values*

*// from len[] in it. Initially, size*

*// of min heap is equal to capacity*

*struct MinHeap \*createAndBuildMinHeap(int len[], int size)*

*{*

*struct MinHeap \*minHeap = createMinHeap(size);*

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

*minHeap->harr[i] = len[i];*

*minHeap->size = size;*

*buildMinHeap(minHeap);*

*return minHeap;*

*}*

*// The main function that returns*

*// the minimum cost to connect n*

*// ropes of lengths stored in len[0..n-1]*

*int minCost(int len[], int n)*

*{*

*int cost = 0; // Initialize result*

*// Create a min heap of capacity*

*// equal to n and put all ropes in it*

*struct MinHeap \*minHeap = createAndBuildMinHeap(len, n);*

*// Iterate while size of heap doesn't become 1*

*while (!isSizeOne(minHeap))*

*{*

*// Extract two minimum length*

*// ropes from min heap*

*int min = extractMin(minHeap);*

*int sec\_min = extractMin(minHeap);*

*cost += (min + sec\_min); // Update total cost*

*// Insert a new rope in min heap*

*// with length equal to sum*

*// of two extracted minimum lengths*

*insertMinHeap(minHeap, min + sec\_min);*

*}*

*// Finally return total minimum*

*// cost for connecting all ropes*

*return cost;*

*}*

*// Driver program to test above functions*

*int main()*

*{*

*int len[] = {4, 3, 2, 6};*

*int size = sizeof(len) / sizeof(len[0]);*

*cout << "Total cost for connecting ropes is "*

*<< minCost(len, size);*

*return 0;*

*}*

***Q6)***

*// C++ program to demonstrate Extract min, Deletion()*

*// and Decrease key() operations in a fibonacci heap*

*#include <cmath>*

*#include <cstdlib>*

*#include <iostream>*

*#include <malloc.h>*

*using namespace std;*

*// Creating a structure to represent a node in the heap*

*struct node*

*{*

*node \*parent; // Parent pointer*

*node \*child; // Child pointer*

*node \*left; // Pointer to the node on the left*

*node \*right; // Pointer to the node on the right*

*int key; // Value of the node*

*int degree; // Degree of the node*

*char mark; // Black or white mark of the node*

*char c; // Flag for assisting in the Find node function*

*};*

*// Creating min pointer as "mini"*

*struct node \*mini = NULL;*

*// Declare an integer for number of nodes in the heap*

*int no\_of\_nodes = 0;*

*// Function to insert a node in heap*

*void insertion(int val)*

*{*

*struct node \*new\_node = new node();*

*new\_node->key = val;*

*new\_node->degree = 0;*

*new\_node->mark = 'W';*

*new\_node->c = 'N';*

*new\_node->parent = NULL;*

*new\_node->child = NULL;*

*new\_node->left = new\_node;*

*new\_node->right = new\_node;*

*if (mini != NULL)*

*{*

*(mini->left)->right = new\_node;*

*new\_node->right = mini;*

*new\_node->left = mini->left;*

*mini->left = new\_node;*

*if (new\_node->key < mini->key)*

*mini = new\_node;*

*}*

*else*

*{*

*mini = new\_node;*

*}*

*no\_of\_nodes++;*

*} // Linking the heap nodes in parent child relationship*

*void Fibonnaci\_link(struct node \*ptr2, struct node \*ptr1)*

*{*

*(ptr2->left)->right = ptr2->right;*

*(ptr2->right)->left = ptr2->left;*

*if (ptr1->right == ptr1)*

*mini = ptr1;*

*ptr2->left = ptr2;*

*ptr2->right = ptr2;*

*ptr2->parent = ptr1;*

*if (ptr1->child == NULL)*

*ptr1->child = ptr2;*

*ptr2->right = ptr1->child;*

*ptr2->left = (ptr1->child)->left;*

*((ptr1->child)->left)->right = ptr2;*

*(ptr1->child)->left = ptr2;*

*if (ptr2->key < (ptr1->child)->key)*

*ptr1->child = ptr2;*

*ptr1->degree;*

*} // Consolidating the heap*

*void Consolidate()*

*{*

*int temp1;*

*float temp2 = (log(no\_of\_nodes)) / (log(2));*

*int temp3 = temp2;*

*struct node \*arr[temp3 + 1];*

*for (int i = 0; i <= temp3; i++)*

*arr[i] = NULL;*

*node \*ptr1 = mini;*

*node \*ptr2;*

*node \*ptr3;*

*node \*ptr4 = ptr1;*

*do*

*{*

*ptr4 = ptr4->right;*

*temp1 = ptr1->degree;*

*while (arr[temp1] != NULL)*

*{*

*ptr2 = arr[temp1];*

*if (ptr1->key > ptr2->key)*

*{*

*ptr3 = ptr1;*

*ptr1 = ptr2;*

*ptr2 = ptr3;*

*}*

*if (*

*ptr2 == mini)*

*mini = ptr1;*

*Fibonnaci\_link(ptr2, ptr1);*

*if (ptr1->right == ptr1)*

*mini = ptr1;*

*arr[temp1] = NULL;*

*temp1++;*

*}*

*arr[temp1] = ptr1;*

*ptr1 = ptr1->right;*

*} while (ptr1 != mini);*

*mini = NULL;*

*for (int j = 0; j <= temp3; j++)*

*{*

*if (arr[j] != NULL)*

*{*

*arr[j]->left = arr[j];*

*arr[j]->right = arr[j];*

*if (mini != NULL)*

*{*

*(mini->left)->right = arr[j];*

*arr[j]->right = mini;*

*arr[j]->left = mini->left;*

*mini->left = arr[j];*

*if (arr[j]->key < mini->key)*

*mini = arr[j];*

*}*

*else*

*{*

*mini = arr[j];*

*}*

*if (*

*mini == NULL)*

*mini = arr[j];*

*else if (arr[j]->key < mini->key)*

*mini = arr[j];*

*}*

*}*

*}*

*// Function to extract minimum node in the heap*

*void Extract\_min()*

*{*

*if (mini == NULL)*

*cout << "The heap is empty" << endl;*

*else*

*{*

*node \*temp = mini;*

*node \*pntr;*

*pntr = temp;*

*node \*x = NULL;*

*if (temp->child != NULL)*

*{*

*x = temp->child;*

*do*

*{*

*pntr = x->right;*

*(mini->left)->right = x;*

*x->right = mini;*

*x->left = mini->left;*

*mini->left = x;*

*if (x->key < mini->key)*

*mini = x;*

*x->parent = NULL;*

*x = pntr;*

*} while (pntr != temp->child);*

*}*

*(*

*temp->left)*

*->right =*

*temp->right;*

*(temp->right)->left = temp->left;*

*mini = temp->right;*

*if (temp == temp->right && temp->child == NULL)*

*mini = NULL;*

*else*

*{*

*mini = temp->right;*

*Consolidate();*

*}*

*no\_of\_nodes--;*

*}*

*}*

*// Cutting a node in the heap to be placed in the root list*

*void Cut(struct node \*found, struct node \*temp)*

*{*

*if (found == found->right)*

*temp->child = NULL;*

*(found->left)->right = round->right;*

*(found->right)->left = found->left;*

*if (found == temp->child)*

*temp->child = found->right;*

*temp->degree = temp->degree - 1;*

*found->right = found;*

*found->left = found;*

*(mini->left)->right = found;*

*found->right = mini;*

*found->left = mini->left;*

*mini->left = found;*

*found->parent = NULL;*

*found->mark = 'B';*

*}*

*// Recursive cascade cutting function*

*void Cascase\_cut(struct node \*temp)*

*{*

*node \*ptr5 = temp->parent;*

*if (ptr5 != NULL)*

*{*

*if (temp->mark == 'W')*

*{*

*temp->mark = 'B';*

*}*

*else*

*{*

*Cut(temp, ptr5);*

*Cascase\_cut(ptr5);*

*}*

*}*

*}*

*// Function to decrease the value of a node in the heap*

*void Decrease\_key(struct node \*found, int val)*

*{*

*if (mini == NULL)*

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

*if (found == NULL)*

*cout << "Node not found in the Heap" << endl;*

*found->key = val;*

*struct node \*temp = found->parent;*

*if (temp != NULL && found->key < temp->key)*

*{*

*Cut(found, temp);*

*Cascase\_cut(temp);*

*}*

*if (*

*found->key <*

*mini->key)*

*mini = found;*

*}*

*// Function to find the given node*

*void Find(struct node \*mini, int old\_val, int val)*

*{*

*struct node \*found = NULL;*

*node \*temp5 = mini;*

*temp5->c = 'Y';*

*node \*found\_ptr = NULL;*

*if (temp5->key == old\_val)*

*{*

*found\_ptr = temp5;*

*temp5->c = 'N';*

*found = found\_ptr;*

*Decrease\_key(found, val);*

*}*

*if (*

*found\_ptr == NULL)*

*{*

*if (temp5->child != NULL)*

*Find(temp5->child, old\_val, val);*

*if ((temp5->right)->c != 'Y')*

*Find(temp5->right, old\_val, val);*

*}*

*temp5->c =*

*' N';*

*found = found\_ptr;*

*}*

*// Deleting a node from the heap*

*void Deletion(int val)*

*{*

*if (mini == NULL)*

*cout << "The heap is empty" << endl;*

*else*

*{*

*// Decreasing the value of the node to 0*

*Find(mini, val, 0);*

*// Calling Extract\_min function to*

*// delete minimum value node, which is 0*

*Extract\_min();*

*cout << "Key Deleted" << endl;*

*}*

*}*

*// Function to display the heap*

*void display()*

*{*

*node \*ptr = mini;*

*if (ptr == NULL)*

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

*else*

*{*

*cout << "The root nodes of Heap are: " << endl;*

*do*

*{*

*cout << ptr->key;*

*ptr = ptr->right;*

*if (ptr != mini)*

*{*

*cout << "-->";*

*}*

*} while (ptr != mini && ptr->right != NULL);*

*cout << endl*

*<< "The heap has " << no\_of\_nodes << " nodes" << endl*

*<< endl;*

*}*

*}*

*// Driver code*

*int main()*

*{*

*// We will create a heap and insert 3 nodes into it*

*cout << "Creating an initial heap" << endl;*

*insertion(5);*

*insertion(2);*

*insertion(8);*

*// Now we will display the root list of the heap*

*display();*

*// Now we will extract the minimum value node from the heap*

*cout << "Extracting min" << endl;*

*Extract\_min();*

*display();*

*// Now we will decrease the value of node '8' to '7'*

*cout << "Decrease value of 8 to 7" << endl;*

*Find(mini, 8, 7);*

*display();*

*// Now we will delete the node '7'*

*cout << "Delete the node 7" << endl;*

*Deletion(7);*

*display();*

*return 0;*

*}*

***Q7)***

*// C++ program for implementation of Heap Sort*

*#include <bits/stdc++.h>*

*using namespace std;*

*// To heapify a subtree rooted with node i which is*

*// an index in arr[]. n is size of heap*

*void heapify(int arr[], int n, int i)*

*{*

*int smallest = i; // Initialize smallest as root*

*int l = 2 \* i + 1; // left = 2\*i + 1*

*int r = 2 \* i + 2; // right = 2\*i + 2*

*// If left child is smaller than root*

*if (l < n && arr[l] < arr[smallest])*

*smallest = l;*

*// If right child is smaller than smallest so far*

*if (r < n && arr[r] < arr[smallest])*

*smallest = r;*

*// If smallest is not root*

*if (smallest != i)*

*{*

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

*// Recursively heapify the affected sub-tree*

*heapify(arr, n, smallest);*

*}*

*}*

*// main function to do heap sort*

*void heapSort(int arr[], int n)*

*{*

*// Build heap (rearrange array)*

*for (int i = n / 2 - 1; i >= 0; i--)*

*heapify(arr, n, i);*

*// One by one extract an element from heap*

*for (int i = n - 1; i >= 0; i--)*

*{*

*// Move current root to end*

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

*// call min heapify on the reduced heap*

*heapify(arr, i, 0);*

*}*

*}*

*/\* A utility function to print array of size n \*/*

*void printArray(int arr[], int n)*

*{*

*for (int i = 0; i < n; ++i)*

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

*cout << "\n";*

*}*

*// Driver program*

*int main()*

*{*

*int arr[] = {4, 6, 3, 2, 9};*

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

*heapSort(arr, n);*

*cout << "Sorted array is \n";*

*printArray(arr, n);*

*}*