Dr. B.R Ambedkar National Institute of Technology, Jalandhar

Introduction to Design and analysis of algorithms Lab
(ITPC - 222)

Submitted to

Dr. Mohit Kumar

Department of IT

Submitted By

Aditya Anand

20124009
IT (G1)

Table of contents

	8.3) Find Min 8.4) Extract Min 8.5) Union 8.6) Decrease a key 8.7) Delete			
9.	Red Black Tree 9.1) Inserting an element 9.2) Deleting an element	April 14, 2022		
10.	Greedy Algorithms 10.1) Deadline based job scheduling 10.2) Activity Selection Problem 10.3) Huffman Code 10.4) Kruskals Algorithm(MST) 10.5) Prims Algorithm(MST) 10.6) Fractional Knapsack 10.7) Dijkstras Algorithm(SSSP) 10.8) Bellman Ford	April 21, 2022		
11.	Dynamic Programming Algorithms 11.1) 0/1 Knapsack 11.2) Assembly Scheduling 11.3) LCS 11.4) MCM 11.5) Bellman Ford 11.6) Floyd Warshall 11.7) Optimal BST	April 28, 2022		

Lab – 1

S No	Program Title	Date of Implementation	Remarks
i.	Linear Search	Jan 6, 2022	
ii.	Binary Search	Jan 6, 2022	
iii.	Selection Sort	Jan 6, 2022	

a) Linear Search

Input: Array of integer

Output: Index of element to be search in input array

Time Complexity: O(n)

Code:

```
#include<bits/stdc++.h>
using namespace :: std;

// O(n)
bool seq_search(vector<int> v, int target){
    for(int i=0; i<v.size(); i++){
        if(target==v[i]){
            return true;
        }
    }
    return false;
}

int main(){
    int n = 0;
    cin>>n;
    vector<int> v(n, 0);
    for(int i=0; i<n; i++){
        cin>>v[i];
    }

    int target = 0;
```

```
    cin>>target;
    cout<<"SEQUENTIAL SEARCH:\n";
    if(seq_search(v, target)){
        cout<<"Present\n";
    }
    else{
        cout<<"Not Present\n";
    }

    return 0;
}
```

Output:

```
PS C:\Users\beadi\Desktop\DAA LAB>
IEngine-In-dcoczwbc.ygg' '--stdout=
z' '--dbgExe=C:\msys64\mingw64\bin'
4
1 7 2 3
7
SEQUENTIAL SEARCH:
Present
```

b) Binary Search

Input: Array of integer

Output: Index of element to be search in sorted input array

Time Complexity: O(log(n))

Code:

```
#include<bits/stdc++.h>
using namespace :: std;

// O(log(n))
bool binarySearch(vector<int> v, int target){
    int i=0, j=v.size()-1;
    while(j>=i){
        int mid = i + (j-i)/2;
        if(v[mid]==target){
            return true;
        }
        else if(v[mid]>target){
            j=mid-1;
        }
        else{
            i=mid+1;
```

```
        }
    }
    return false;
}

int main(){
    int n = 0;
    cin>>n;
    vector<int> v(n, 0);
    for(int i=0; i<n; i++){
        cin>>v[i];
    }

    int target = 0;
    cin>>target;

    cout<<"BINARY SEARCH:\n";
    if(binarySearch(v, target)){
        cout<<"Present\n";
    }
    else{
        cout<<"Not Present\n";
    }

    return 0;
}
```

Output:

```
PS C:\Users\beadi\Desktop\DAA LAB>
IEngine-In-5cpvh3w1.jnt' '--stdout
r' '--dbgExe=C:\msys64\mingw64\bin
4
1 4 7 9
4
SEQUENTIAL SEARCH:
Present
```

c) Selection Sort

Input: Array of integer

Output: Index of element to be search in input array

Time Complexity: O(n^2)

Code:

```
#include<bits/stdc++.h>
using namespace :: std;
```

```cpp
// O(n^2)
void selection_sort(vector<int> &v){
    // Array from 0 to i-1 is sorted, i to n is unsorted
    int i=0;
    while(i<v.size()){
        int mn = INT_MAX;
        int id = -1;
        for(int j=i; j<v.size(); j++){
            if(mn>v[j]){
                mn = v[j];
                id = j;
            }
        }
        swap(v[i], v[id]);
        i++;
        for(auto ele:v){
            cout<<ele<<" ";
        }
        cout<<"\n";
    }
}

int main(){
    int n=0;
    cin>>n;
    vector<int> v(n, 0);
    for(int i=0; i<n; i++){
        cin>>v[i];
    }
    selection_sort(v);
    cout<<"Sorted array: ";
    for(auto i:v){
        cout<<i<<" ";
    }
    return 0;
}
```

Output:

```
PS C:\Users\beadi\Desktop\DAA LAB\Assignment 1>
 .\selectionSort }
4
1 7 2 4
1 7 2 4
1 2 7 4
1 2 4 7
1 2 4 7
Sorted array: 1 2 4 7
```

Lab – 2

S No	Program Title	Date of Implementation	Remarks
i.	Insertion Sort	Jan 31, 2022	
ii.	Quick sort	Jan 31, 2022	

a) Insertion Sort

Input: Array of integer

Output: Sorted array of integer

Time Complexity: O(n^2)

Code:

```
#include <bits/stdc++.h>
using namespace :: std;

void insertionSort(vector<int> &arr, int n){
    int i, key, j;
    for (i = 1; i < n; i++){
        key = arr[i];
        cout<<"key = "<<key<<"\n";
        j = i - 1;

        while (j >= 0 && arr[j] > key){
            arr[j + 1] = arr[j];
            j--;
        }
        arr[j + 1] = key;

        for(auto i: arr){
            cout<<i<<" ";
        }
        cout<<"\n";
    }
    return;
}
```

```
int main(){
    vector<int> arr = {7,8,2,5,6};
    int n = arr.size();

    insertionSort(arr, n);
    cout<<"Sorted Array: ";
    for(auto i:arr){
        cout<<i<<" ";
    }

    return 0;
}
```

Output:

```
PS C:\Users\beadi\Desktop\DAA LAB\Assignment 1>
 { .\insertion_sort }
key = 8
7 8 2 5 6
key = 2
2 7 8 5 6
key = 5
2 5 7 8 6
key = 6
2 5 6 7 8
Sorted Array: 2 5 6 7 8
```


b) Quick Sort

Input: Array of integer

Output: Sorted array of integer

Time Complexity:

Best case: Ω(n*log(n)) Average case: θ(n*log(n)) Worst case: O(n^2)

Code:

```
#include <bits/stdc++.h>
using namespace std;

int N;

void printArray(int arr[], int size)
{
    int i;
    for (i = 0; i < size; i++)
        cout << arr[i] << " ";
    cout << endl;
}

int partition1(int a[], int start, int end)
{

    int pivot = a[start], p1 = start + 1, i, temp;

    for (i = start + 1; i <= end; i++)
    {

        if (a[i] < pivot)
        {
            if (i != p1)
            {
                temp = a[p1];
                a[p1] = a[i];
                a[i] = temp;
            }
            p1++;
        }
    }

    a[start] = a[p1 - 1];
    a[p1 - 1] = pivot;

    return p1 - 1;
}

void quicksort(int *a, int start, int end)
{
    int p1;
    if (start < end)
    {
        p1 = partition1(a, start, end);
        quicksort(a, start, p1 - 1);
        quicksort(a, p1 + 1, end);
    }
```

```
}

int main()
{
    int arr[] = {7, 8, 2, 5, 6};
    int n = sizeof(arr) / sizeof(arr[0]);
    N = n;
    quicksort(arr, 0, n - 1);
    cout << "Sorted array: \n";
    printArray(arr, n);
    return 0;
}
```

Output:

```
PS C:\Users\beadi\Desktop\DAA LAB>
pivot } ; if ($?) { .\quick_sort_f
Sorted array:
2 5 6 7 8
```

Lab – 3

S No	Program Title	Date of Implementation	Remarks
i.	Merge Sort	Feb 07, 2022	

Merge Sort

Input: Array of integer

Output: Sorted array of integer

Time Complexity: O(n*log(n))

Code:

```
#include <iostream>
using namespace std;
void merge(int *arr, int low, int mid, int high)
{
    int n1 = mid - low + 1, n2 = high - mid;
    int left[n1], right[n2];
    for (int i = 0; i < n1; i++) // making auxiliary arrays left and right
        left[i] = arr[i + low];
    for (int i = 0; i < n2; i++)
        right[i] = arr[mid + 1 + i];

    int i = 0, j = 0, k = low;
    for (; i < n1 && j < n2;)
    {
        if (left[i] <= right[j])
        {
            arr[k] = left[i];
            i++;
            k++;
        }
        else
        {
            arr[k] = right[j];
            k++;
            j++;
        }
    }

    while (i < n1)
```

```
    { // if i<n1 this loop run
        arr[k] = left[i];
        i++, k++;
    }
    while (j < n2)
    { // else if j<n2 this loop run
        arr[k] = right[j];
        k++, j++;
    }

    cout << "Work done by Merge function: ";
    for (int i = low; i <= high; i++)
    {
        cout << arr[i] << " ";
    }
    cout << endl
         << endl;
}

void mergeSort(int *arr, int left, int right)
{
    static int size = right + 1;
    if (left < right)
    {
        int mid = (left + right) / 2;
        mergeSort(arr, left, mid);
        mergeSort(arr, mid + 1, right);
        merge(arr, left, mid, right);
        cout << "Array after internal merge sort working:  ";
        for (int i = 0; i < size; i++)
        {
            cout << arr[i] << " ";
        }
        cout << endl;
    }
}

int main()
{

    cout << "------MERGE SORT------" << endl
         << endl;
    int n = 0;
    cout << "Enter the size of array: ";
    cin >> n;
    int arr[n];
    cout << "Enter the array elements: ";
    for (int i = 0; i < n; i++)
```

```
        cin >> arr[i];

    cout << "\nArray before sorting: ";
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";
    cout << endl
         << endl;
    mergeSort(arr, 0, n - 1);
    cout << "\nArray after sorting: ";
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";

    return 0;
}
```

Output:

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL

PS C:\IDAA Lab> g++ .\mergeSort.cpp
PS C:\IDAA Lab> .\a.exe
------MERGE SORT------

Enter the size of array: 6
Enter the array elements: 45 7 11 25 36 4

Array before sorting: 45 7 11 25 36 4

Work done by Merge function: 7 45

Array after internal merge sort working:  7 45 11 25 36 4
Work done by Merge function: 7 11 45

Array after internal merge sort working:  7 11 45 25 36 4
Work done by Merge function: 25 36

Array after internal merge sort working:  7 11 45 25 36 4
Work done by Merge function: 4 25 36

Array after internal merge sort working:  7 11 45 4 25 36
Work done by Merge function: 4 7 11 25 36 45

Array after internal merge sort working:  4 7 11 25 36 45

Array after sorting: 4 7 11 25 36 45
PS C:\IDAA Lab>
```

Lab – 4

S No	Program Title	Date of Implementation	Remarks
i.	Binary Search using Recusion	Feb 14, 2022	
ii.	Factorial using Recusion	Feb 14, 2022	
iii.	Fibonacci series using Recursion	Feb 14, 2022	
iv.	Sum of n Natural numbers using Recusion	Feb 14, 2022	

a) Binary Search

Input: Array of integer

Output: Index of element to be search in sorted input array

Time Complexity: O(log(n))

Code:

```
#include<bits/stdc++.h>
using namespace :: std;

int binarySearch(vector<int> v, int s, int e, int key){
    if(s>e){
        return -1;
    }

    int mid = s+(e-s)/2;
    if(v[mid]==key){
        return mid;
    }
    else if(v[mid]>key){
        return binarySearch(v, s, mid-1, key);
    }
    else{
        return binarySearch(v, mid+1, e, key);
    }
}
```

```
int main(){
    vector<int> v = {1,2,3,6,8,9};
    int n = v.size();

    cout<<"Enter key: ";
    int key=0;
    cin>>key;
    int idx = binarySearch(v, 0, n-1, key);
    if(idx!=-1){
        cout<<"Present at index "<<idx;
    }
    else{
        cout<<"Element is not present";
    }
    return 0;
}
```

Output:

```
PS C:\Users\beadi\Desktop\IDAA\DAA LAB\Assignment 8>
rySearch }
Enter key: 6
Present at index 3
```

b) Factorial

Input: Integer

Output: Factorial of input integer

Time Complexity: O(n)

Code:

```
#include<bits/stdc++.h>
using namespace :: std;

int factorial(int n){
    if(n==0){
        return 1;
    }

    return n*factorial(n-1);
}

int main(){
    int n=0;
    cin>>n;
    cout<<n<<" factorial ="<<factorial(n)<<"\n";
    return 0;
}
```

Output:

```
PS C:\Users\beadi\Desktop\IDAA\DAA LAB\Assignment 4>
}
5
5 factorial =120
```

c) Fibonacci

Input: Integer

Output: Fibonacci series having length equal to Input integer

Time Complexity: O(n)

Code:

```
#include<bits/stdc++.h>
using namespace :: std;

int fibonacci(int n){
    if(n==0 || n==1){
        return n;
    }

    return fibonacci(n-1)+fibonacci(n-2);
}

int main(){
    int n = 0;
    cin>>n;
    cout<<n<<"th fibonacci = "<<fibonacci(n);
    return 0;
}
```

Output:

```
PS C:\Users\beadi\Desktop\IDAA\DAA LAB\Assignment 4>
}
6
6th fibonacci = 8
```

d) Sum of n Natural numbers

Input: Integer

Output: Sum from 1 to input integer value

Time Complexity: O(n)

Code:

```
#include<bits/stdc++.h>
using namespace :: std;

int sum(int n){
    if(n==1){
        return 1;
    }

    return n+sum(n-1);
}

int main(){
    int n=0;
    cin>>n;
    cout<<"sum of first "<<n<< " numbers = "<<sum(n);
    return 0;
}
```

Output:

```
PS C:\Users\beadi\Desktop\IDAA\DAA LAB\Assignment 4>
if ($?) { .\sumOfNnaturalNumbers }
12
sum of first 12 numbers = 78
```

Lab – 5

S No	Program Title	Date of Implementation	Remarks
i.	Matrix Multiplication using recursion	March 24, 2022	

Matrix Multiplication

Input: 2 Matrix of integers

Output: Product of matrices in matrix form

Time Complexity: O(n^2.81)

Code:

```
#include <bits/stdc++.h>
using namespace std;
int **setAllZero(int n)
{
    int **res = new int *[n];
    for (int i = 0; i < n; i++)
    {
        res[i] = new int[n];
        for (int j = 0; j < n; j++)
        {
            res[i][j] = 0;
        }
    }
    return res;
}
int **matrixAddition(int **arr, int **arr1, int n)
{
    int **res = setAllZero(n);
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            res[i][j] = arr[i][j] + arr1[i][j];
        }
    }
    return res;
}
int **matrixSubtraction(int **arr, int **arr1, int n)
{
```

```
    int **res = setAllZero(n);
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            res[i][j] = arr[i][j] - arr1[i][j];
        }
    }
    return res;
}
int **matrixMultiplication(int **arr, int **arr1, int n)
{
    int **res = setAllZero(n);
    if (n == 1)
    {
        res[0][0] = arr[0][0] * arr1[0][0];
        return res;
    }
    int **a11 = setAllZero(n / 2);
    int **a12 = setAllZero(n / 2);
    int **a21 = setAllZero(n / 2);
    int **a22 = setAllZero(n / 2);
    int **b11 = setAllZero(n / 2);
    int **b12 = setAllZero(n / 2);
    int **b21 = setAllZero(n / 2);
    int **b22 = setAllZero(n / 2);
    for (int i = 0; i < n / 2; i++)
    {
        for (int j = 0; j < n / 2; j++)
        {
            a11[i][j] = arr[i][j];
            a12[i][j] = arr[i][n / 2 + j];
            a21[i][j] = arr[i + n / 2][j];
            a22[i][j] = arr[i + n / 2][j + n / 2];
            b11[i][j] = arr1[i][j];
            b12[i][j] = arr1[i][n / 2 + j];
            b21[i][j] = arr1[i + n / 2][j];
            b22[i][j] = arr1[i + n / 2][j + n / 2];
        }
    }
    int **p = matrixMultiplication(matrixAddition(a11, a22, n / 2),
matrixAddition(b11, b22, n / 2), n / 2);
    int **q = matrixMultiplication(matrixAddition(a21, a22, n / 2), b11, n /
2);
    int **r = matrixMultiplication(a11, matrixSubtraction(b12, b22, n / 2), n
/ 2);
    int **s = matrixMultiplication(a22, matrixSubtraction(b21, b11, n / 2), n
/ 2);
```

```
    int **t = matrixMultiplication(matrixAddition(a11, a12, n / 2), b22, n /
2);
    int **u = matrixMultiplication(matrixSubtraction(a21, a11, n / 2),
matrixAddition(b11, b12, n / 2), n / 2);
    int **v = matrixMultiplication(matrixSubtraction(a12, a22, n / 2),
matrixAddition(b21, b22, n / 2), n / 2);
    int **c11 = matrixAddition(p, matrixAddition(v, matrixSubtraction(s, t, n
/ 2), n / 2), n / 2);
    int **c12 = matrixAddition(r, t, n / 2);
    int **c21 = matrixAddition(q, s, n / 2);
    int **c22 = matrixAddition(p, matrixAddition(u, matrixSubtraction(r, q, n
/ 2), n / 2), n / 2);
    for (int i = 0; i < n / 2; i++)
    {
        for (int j = 0; j < n / 2; j++)
        {
            res[i][j] = c11[i][j];
            res[i][j + n / 2] = c12[i][j];
            res[i + n / 2][j] = c21[i][j];
            res[i + n / 2][j + n / 2] = c22[i][j];
        }
    }
    return res;
}
int main()
{
    int n;
    cout << "Enter the dimension : ";
    cin >> n;
    int **arr = setAllZero(n);
    int **arr1 = setAllZero(n);
    cout << "Enter the elements for first matrix : " << endl;
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            cin >> arr[i][j];
        }
    }
    cout << "Enter the elements for second matrix : " << endl;
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            cin >> arr1[i][j];
        }
    }
    int **res;
```

```
    res = matrixMultiplication(arr, arr1, n);
    cout << "The multiplication of 2 matrices is : " << endl;
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            cout << res[i][j] << "\t";
        }
        cout << endl;
    }
}
```

Output:

```
PS C:\Users\beadi\Desktop\IDAA\DAA LAB\Assignment 4>
if ($?) { .\matMultiplicationRec }
Enter the dimension : 4
Enter the elements for first matrix :
1 2 3 4
1 2 3 4
5 2 3 1
6 3 4 1
Enter the elements for second matrix :
4 1 2 3
8 3 6 2
6 2 5 1
8 3 1 5
The multiplication of 2 matrices is :
70      25      33      30
70      25      33      30
62      20      38      27
80      26      51      33
```

Lab – 6

S No	Program Title	Date of Implementation	Remarks
i.	Heap Sort	Mar 23, 2022	

Heap Sort

Input: Array of integer

Output: Sorted array of integer

Time Complexity: O(n*log(n))

Code:

```
#include<bits/stdc++.h>
using namespace :: std;

void heapSort(vector<int> &v){
    int n=v.size();

    // Create the heap
    for(int i=1; i<n; i++){
        int t=i;
        while((t/2)>0 && v[t/2]>v[t]){
            swap(v[t/2], v[t]);
            t/=2;
        }
    }

    // Remove root from the heap and store it at the end
    for(int i=n-1; i>1; i--){
        swap(v[i], v[1]);
        int t=1;
        while((2*t+1)<i){
            int l=2*t;
            int r=2*t+1;

            if(l<i && r<i){
                if(v[l]<v[r]){
                   if(v[2*t]<v[t]){
                        swap(v[2*t], v[t]);
                        t=2*t;
                        continue;
```

```
                    }
                }
                else{
                    if(v[2*t+1]<v[t]){
                        swap(v[2*t+1], v[t]);
                        t=2*t+1;
                        continue;
                    }
                }
            }
            else{
                break;
            }
        }
        if((2*t+1)==i){
            if(v[2*t]<v[t]){
                swap(v[2*t], v[t]);
                t=2*t;
            }
        }
    }
}

int main(){
    int n=0;
    cin>>n;
    vector<int> v(n+1, 0);   // because 1 based indexing
    for(int i=1; i<n+1; i++){
        cin>>v[i];
    }
    heapSort(v);

    for(int i=1; i<n+1; i++){
        cout<<v[i]<<" ";
    }
    return 0;
}
```

Output:

```
PS C:\Users\beadi\Desktop\DAA LAB\Assignment 7:
t }
4
1 5 2 3
5 3 2 1
```

Lab – 7

S No	Program Title	Date of Implementation	Remarks
i.	Binomial Heap	Mar 30, 2022	

Binomial Heap

Input: Integers

Output: Minimum element in heap and full heap traversal print

Time Complexity:

a) Make Heap = O(1)
b) Find Min = O(nlog(n))
c) Union = O(nlog(n))
d) Insert a node = O(nlog(n))
e) Extract min = O(nlog(n))
f) Decrease a key = O(nlog(n))
g) Delete a node = O(nlog(n))

Code:

```
#include <bits/stdc++.h>
using namespace std;

// A binomial heap node structure
struct node{
    int degree, data;
    struct node *parent, *sibling, *child;
};

// This function creates a new node with given key
struct node *newNode(int key){
    struct node *temp = new node;
    temp->data = key;
    temp->degree = 0;
    temp->child = temp->parent = temp->sibling = NULL;
    return temp;
    }
```

```
// Merging two binomial trees
struct node *mergeBinomialTrees(struct node *b1, struct node *b2){

if (b1->data > b2->data)
    swap(b1, b2);

b2->parent = b1;
b2->sibling = b1->child;
b1->child = b2;
b1->degree++;

return b1;
}

// This function performs union of two Binomial heaps
list<node *> unionBinomialHeap(list<node *> l1, list<node *> l2){
list<node *> res;

auto i = l1.begin();
auto j = l2.begin();

while (i != l1.end() && j != l2.end())
{
    if ((*i)->degree <= (*j)->degree)
    {
    res.push_back((*i));
    i++;
    }
    else
    {
    res.push_back((*j));
    j++;
    }
}

while (i != l1.end())
{
    res.push_back((*i));
    i++;
}

while (j != l2.end())
{
    res.push_back((*j));
    j++;
}

return res;
```

```
}

// Adjust function ensures that the root nodes in list are in increasing order
and no two binomial trees
// have the same degree.
list<node *> adjust(list<node *> heap){

    if (heap.size() <= 1)
        return heap;

    list<node *> new_heap;
    list<node *>::iterator it1, it2, it3;
    it1 = it2 = it3 = heap.begin();

    if (heap.size() == 2)
    {
        it2 = it1;
        it2++;
        it3 = heap.end();
    }
    else
    {
        it2++;
        it3 = it2;
        it3++;
    }

    while (it1 != heap.end())
    {
        if (it2 == heap.end())
        it1++;
        else if ((*it1)->degree < (*it2)->degree)
        {
        it1++;
        it2++;
        if (it3 != heap.end())
            it3++;
        }
        else if (it3 != heap.end() && (*it1)->degree == (*it2)->degree &&
(*it1)->degree == (*it3)->degree)
        {
        it1++;
        it2++;
        it3++;
        }
        else if ((*it1)->degree == (*it2)->degree)
        {
        struct node *temp;
```

```
            *it1 = mergeBinomialTrees(*it1, *it2);
            it2 = heap.erase(it2);
            if (it3 != heap.end())
                it3++;
            }
        }

        return heap;
    }

    // This function adds a Binomial tree in heap and then performs union on it
    list<node *> insertTreeInHeap(list<node *> heap, struct node *tree){
        list<node *> temp;
        temp.push_back(tree);

        temp = unionBinomialHeap(temp, heap);

        return adjust(temp);
    }

    // This function inserts a new node in Binomial heap
    list<node *> insert(int key, list<node *> heap){
        struct node *temp;
        temp = newNode(key);
        return insertTreeInHeap(heap, temp);
    }

    // This function returns the pointer to the minimum element of entire heap
    struct node *getMin(list<node *> heap){
        struct node *minimum = NULL;
        int mini = INT_MAX;
        auto it = heap.begin();
        while (it != heap.end())
        {
            if ((*it)->data < mini)
            {
            mini = (*it)->data;
            minimum = (*it);
            }

            it++;
        }

        return minimum;
    }
```

```
// This function is a helper function and it includes all the children of
minimum node in the main root list and then performs union and adjust to
ensure binomial trees of unique degree
list<node *> removeMinimum(struct node *tree){
    list<node *> heap;
    struct node *temp = tree->child;
    struct node *helper;

    while (temp)
    {
        helper = temp;
        temp = temp->sibling;
        helper->sibling = NULL;
        helper->parent = NULL;
        heap.push_front(helper);
    }
    return heap;
}

// This function extracts the minimum node from the Binomial heap and returns
the modified heap
list<node *> extractMin(list<node *> heap){
    list<node *> new_heap, helper;
    struct node *temp;

    temp = getMin(heap);
    auto it = heap.begin();

    while (it != heap.end())
    {
        if ((*it) != temp)
        {
        new_heap.push_back((*it));
        }
        it++;
    }

    helper = removeMinimum(temp);
    helper = unionBinomialHeap(new_heap, helper);
    helper = adjust(helper);
    return helper;
}

// This function searches a given Binomial tree for a node with a given value
and returns the pointer to that node
struct node *findNode(struct node *h, int val){
  if (h == NULL)
    return NULL;
```

```
  if (h->data == val)
    return h;

  struct node *res = findNode(h->child, val);
  if (res != NULL)
    return res;

  return findNode(h->sibling, val);
}

// This function takes input an old and a new key and replaces old key with
new key and performs necessary swapping to ensure min-heap property
list<node *> decreaseKey(list<node *> heap, int old_val, int new_val){
  struct node *temp = NULL;
  auto it = heap.begin();

  while (it != heap.end())
  {
    temp = findNode(*it, old_val);

    if (temp != NULL)
      break;
    // (*it) = (*it)->sibling;
    it++;
  }

  if (temp == NULL)
    return heap;

  temp->data = new_val;

  struct node *parent = temp->parent;

  while (parent != NULL && temp->data < parent->data)
  {
    swap(temp->data, parent->data);
    temp = parent;
    parent = parent->parent;
  }

  return heap;
}

// This function takes input a value and deletes the node with corresponding
value from the Binary heap
list<node *> deleteNode(list<node *> heap, int val){
  struct node *temp = NULL;
```

```
  auto it = heap.begin();

  while (it != heap.end())
  {
    temp = findNode(*it, val);

    if (temp != NULL)
      break;

    it++;
  }

  if (temp == NULL)
  {
    cout << "Value to be deleted not found in heap " << endl;
    return heap;
  }

  temp->data = INT_MIN;
  struct node *parent = temp->parent;

  while (parent != NULL && temp->data < parent->data)
  {
    swap(temp->data, parent->data);
    temp = parent;
    parent = parent->parent;
  }
  heap = extractMin(heap);

  return heap;
}

// This function take input a root of a Binomial tree and prints all the
values in that tree using DFS approach
void printTree(struct node *root){
  while (root)
  {
    cout << root->data << " ";
    printTree(root->child);
    root = root->sibling;
  }
}

// This function takes input of a Binomial heap and prints all its key values
void printHeap(list<node *> heap){
  auto it = heap.begin();

  while (it != heap.end())
```

```
  {
    printTree(*it);
    it++;
  }

  cout << endl;
}

// Main function
int main(){
    // 1. Creating a Binomial heap
    list<node *> heap;

    // 2. Inserting values in Binomial heap
    heap = insert(1, heap);
    heap = insert(2, heap);
    heap = insert(3, heap);
    heap = insert(4, heap);
    heap = insert(5, heap);
    heap = insert(6, heap);
    cout << "The heap formed is as follows\n";
    printHeap(heap);

    // 3. Getting minimum element from heap
    cout << "The minimum element in heap is " << getMin(heap)->data << endl;

    // 4. Removing minimum element from heap
    heap = extractMin(heap);
    cout << "Heap after extracing minimum value is as follows \n";
    printHeap(heap);

    // 5. Decreasing a key
    heap = decreaseKey(heap, 2, -1);
    cout << "Heap after decreasing a key is as follows\n";
    printHeap(heap);

    // 6. Deleting a node
    heap = deleteNode(heap, 4);
    cout << "Heap after deleting node is as follows\n";
    printHeap(heap);

    return 0;
}
```

Output:

```
PS C:\Users\beadi\Desktop\IDAA\DAA LAB\Assignment 6>
$?) { .\tempCodeRunnerFile }
The heap formed is as follows
5 6 1 3 4 2
The minimum element in heap is 1
Heap after extracing minimum value is as follows
2 3 5 6 4
Heap after decreasing a key is as follows
-1 3 5 6 4
Heap after deleting node is as follows
-1 5 6 3
```

Lab – 8

S No	Program Title	Date of Implementation	Remarks
i.	Fibonacci Heap	April 7, 2022	

Binomial Heap

Input: Integers

Output: Minimum element in heap and full heap traversal print

Time Complexity:

h) Make Heap = O(1)
i) Find Min = O(1)
j) Union = O(1)
k) Insert a node = O(1)
l) Extract min = O(log(n))
m) Decrease a key = O(1)
n) Delete a node = O(nlog(n))

Code:

```
// C++ program to demonstrate various operations of fibonacci heap
#include<bits/stdc++.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 = found->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;
}
```

Output:

```
PS C:\Users\beadi\Desktop\IDAA\DAA LAB\Assignment 8> cd
{ .\fibonacciHeap }
Creating an initial heap
The root nodes of Heap are:
2-->5-->8
The heap has 3 nodes

Extracting min
The root nodes of Heap are:
5
The heap has 2 nodes

Decrease value of 8 to 7
The root nodes of Heap are:
5
The heap has 2 nodes

Delete the node 7
Key Deleted
The root nodes of Heap are:
5
The heap has 1 nodes
```

Lab – 9

S No	Program Title	Date of Implementation	Remarks
i.	Red Black Trees	April 14, 2022	

Heap Sort

Input: Array of integer

Output: Sorted array of integer

Time Complexity:

Insert an element

O(log(n))

Delete an element

O(log(n))

Code:

```
// Implementing Red-Black Tree in C++

#include <iostream>
using namespace std;

struct Node{
    int data;
    Node *parent;
    Node *left;
    Node *right;
    int color;
};

typedef Node *NodePtr;

class RedBlackTree{
private:
    NodePtr root;
    NodePtr TNULL;

    void initializeNULLNode(NodePtr node, NodePtr parent){
```

```
    node->data = 0;
    node->parent = parent;
    node->left = nullptr;
    node->right = nullptr;
    node->color = 0;
}

// Preorder
void preOrderHelper(NodePtr node){
    if (node != TNULL){
        cout << node->data << " ";
        preOrderHelper(node->left);
        preOrderHelper(node->right);
    }
}

// Inorder
void inOrderHelper(NodePtr node){
    if (node != TNULL){
        inOrderHelper(node->left);
        cout << node->data << " ";
        inOrderHelper(node->right);
    }
}

// Post order
void postOrderHelper(NodePtr node){
    if (node != TNULL){
        postOrderHelper(node->left);
        postOrderHelper(node->right);
        cout << node->data << " ";
    }
}

NodePtr searchTreeHelper(NodePtr node, int key){
    if (node == TNULL || key == node->data){
        return node;
    }

    if (key < node->data){
        return searchTreeHelper(node->left, key);
    }
    return searchTreeHelper(node->right, key);
}

// For balancing the tree after deletion
void deleteFix(NodePtr x){
    NodePtr s;
```

```
    while (x != root && x->color == 0){
        if (x == x->parent->left){
            s = x->parent->right;
            if (s->color == 1){
                s->color = 0;
                x->parent->color = 1;
                leftRotate(x->parent);
                s = x->parent->right;
            }

            if (s->left->color == 0 && s->right->color == 0){
                s->color = 1;
                x = x->parent;
            }
            else{
                if (s->right->color == 0){
                    s->left->color = 0;
                    s->color = 1;
                    rightRotate(s);
                    s = x->parent->right;
                }

                s->color = x->parent->color;
                x->parent->color = 0;
                s->right->color = 0;
                leftRotate(x->parent);
                x = root;
            }
        }
        else
        {
            s = x->parent->left;
            if (s->color == 1){
                s->color = 0;
                x->parent->color = 1;
                rightRotate(x->parent);
                s = x->parent->left;
            }

            if (s->right->color == 0 && s->right->color == 0){
                s->color = 1;
                x = x->parent;
            }
            else{
                if (s->left->color == 0){
                    s->right->color = 0;
                    s->color = 1;
                    leftRotate(s);
```

```
                    s = x->parent->left;
                }

                s->color = x->parent->color;
                x->parent->color = 0;
                s->left->color = 0;
                rightRotate(x->parent);
                x = root;
            }
        }
    }
    x->color = 0;
}

void rbTransplant(NodePtr u, NodePtr v){
    if (u->parent == nullptr){
        root = v;
    }
    else if (u == u->parent->left){
        u->parent->left = v;
    }
    else{
        u->parent->right = v;
    }
    v->parent = u->parent;
}

void deleteNodeHelper(NodePtr node, int key){
    NodePtr z = TNULL;
    NodePtr x, y;
    while (node != TNULL){
        if (node->data == key){
            z = node;
        }

        if (node->data <= key){
            node = node->right;
        }
        else{
            node = node->left;
        }
    }

    if (z == TNULL){
        cout << "Key not found in the tree" << endl;
        return;
    }
```

```
    y = z;
    int y_original_color = y->color;
    if (z->left == TNULL){
        x = z->right;
        rbTransplant(z, z->right);
    }
    else if (z->right == TNULL){
        x = z->left;
        rbTransplant(z, z->left);
    }
    else{
        y = minimum(z->right);
        y_original_color = y->color;
        x = y->right;
        if (y->parent == z){
            x->parent = y;
        }
        else{
            rbTransplant(y, y->right);
            y->right = z->right;
            y->right->parent = y;
        }

        rbTransplant(z, y);
        y->left = z->left;
        y->left->parent = y;
        y->color = z->color;
    }
    delete z;
    if (y_original_color == 0){
        deleteFix(x);
    }
}

// For balancing the tree after insertion
void insertFix(NodePtr k){
    NodePtr u;
    while (k->parent->color == 1){
        if (k->parent == k->parent->parent->right){
            u = k->parent->parent->left;
            if (u->color == 1){
                u->color = 0;
                k->parent->color = 0;
                k->parent->parent->color = 1;
                k = k->parent->parent;
            }
            else{
                if (k == k->parent->left){
```

```
                    k = k->parent;
                    rightRotate(k);
                }
                k->parent->color = 0;
                k->parent->parent->color = 1;
                leftRotate(k->parent->parent);
            }
        }
        else{
            u = k->parent->parent->right;

            if (u->color == 1){
                u->color = 0;
                k->parent->color = 0;
                k->parent->parent->color = 1;
                k = k->parent->parent;
            }
            else{
                if (k == k->parent->right)
                {
                    k = k->parent;
                    leftRotate(k);
                }
                k->parent->color = 0;
                k->parent->parent->color = 1;
                rightRotate(k->parent->parent);
            }
        }
        if (k == root){
            break;
        }
    }
    root->color = 0;
}

void printHelper(NodePtr root, string indent, bool last){
    if (root != TNULL){
        cout << indent;
        if (last){
            cout << "R----";
            indent += "   ";
        }
        else{
            cout << "L----";
            indent += "|  ";
        }

        string sColor = root->color ? "RED" : "BLACK";
```

```
            cout << root->data << "(" << sColor << ")" << endl;
            printHelper(root->left, indent, false);
            printHelper(root->right, indent, true);
        }
    }

public:
    RedBlackTree(){
        TNULL = new Node;
        TNULL->color = 0;
        TNULL->left = nullptr;
        TNULL->right = nullptr;
        root = TNULL;
    }

    void preorder(){
        preOrderHelper(this->root);
    }

    void inorder(){
        inOrderHelper(this->root);
    }

    void postorder(){
        postOrderHelper(this->root);
    }

    NodePtr searchTree(int k){
        return searchTreeHelper(this->root, k);
    }

    NodePtr minimum(NodePtr node){
        while (node->left != TNULL)
        {
            node = node->left;
        }
        return node;
    }

    NodePtr maximum(NodePtr node){
        while (node->right != TNULL)
        {
            node = node->right;
        }
        return node;
    }

    NodePtr successor(NodePtr x){
```

```
    if (x->right != TNULL){
        return minimum(x->right);
    }

    NodePtr y = x->parent;
    while (y != TNULL && x == y->right){
        x = y;
        y = y->parent;
    }
    return y;
}

NodePtr predecessor(NodePtr x){
    if (x->left != TNULL){
        return maximum(x->left);
    }

    NodePtr y = x->parent;
    while (y != TNULL && x == y->left){
        x = y;
        y = y->parent;
    }

    return y;
}

void leftRotate(NodePtr x){
    NodePtr y = x->right;
    x->right = y->left;
    if (y->left != TNULL){
        y->left->parent = x;
    }
    y->parent = x->parent;
    if (x->parent == nullptr){
        this->root = y;
    }
    else if (x == x->parent->left){
        x->parent->left = y;
    }
    else{
        x->parent->right = y;
    }
    y->left = x;
    x->parent = y;
}

void rightRotate(NodePtr x){
    NodePtr y = x->left;
```

```
    x->left = y->right;
    if (y->right != TNULL){
        y->right->parent = x;
    }
    y->parent = x->parent;
    if (x->parent == nullptr){
        this->root = y;
    }
    else if (x == x->parent->right){
        x->parent->right = y;
    }
    else{
        x->parent->left = y;
    }
    y->right = x;
    x->parent = y;
}

// Inserting a node
void insert(int key){
    NodePtr node = new Node;
    node->parent = nullptr;
    node->data = key;
    node->left = TNULL;
    node->right = TNULL;
    node->color = 1;

    NodePtr y = nullptr;
    NodePtr x = this->root;

    while (x != TNULL){
        y = x;
        if (node->data < x->data){
            x = x->left;
        }
        else{
            x = x->right;
        }
    }

    node->parent = y;
    if (y == nullptr){
        root = node;
    }
    else if (node->data < y->data){
        y->left = node;
    }
    else{
```

```
            y->right = node;
        }

        if (node->parent == nullptr){
            node->color = 0;
            return;
        }

        if (node->parent->parent == nullptr){
            return;
        }

        insertFix(node);
    }

    NodePtr getRoot(){
        return this->root;
    }

    void deleteNode(int data){
        deleteNodeHelper(this->root, data);
    }

    void printTree(){
        if (root){
            printHelper(this->root, "", true);
        }
    }
};

int main(){
    RedBlackTree bst;
    bst.insert(55);
    bst.insert(40);
    bst.insert(65);
    bst.insert(60);
    bst.insert(75);
    bst.insert(57);

    bst.printTree();
    cout << endl
         << "After deleting" << endl;
    bst.deleteNode(40);
    bst.printTree();
}
```

Output:

```
PS C:\Users\beadi\Desktop\IDAA\DAA LAB\Assignment 8>
.\redBlackTree }
R----55(BLACK)
   L----40(BLACK)
   R----65(RED)
      L----60(BLACK)
      |  L----57(RED)
      R----75(BLACK)

After deleting
R----65(BLACK)
   L----57(RED)
   |  L----55(BLACK)
   |  R----60(BLACK)
   R----75(BLACK)
```

Lab – 10

S No	Program Title	Date of Implementation	Remarks
i.	Greedy Algorithms	April 21, 2022	

a) Deadline based job scheduling

Input:

The deadlines and profits of the jobs

Output:

The count of jobs and the total profit

Time Complexity:

Time complexity is O(MxN) where M is the maximum value of deadline of a job

Code:

```
#include<bits/stdc++.h>
using namespace :: std;

// we assume that each job takes 1 unit time to complete
class Job{
    public:
        int dead, profit;
};

bool comp(Job a, Job b){
    return a.profit>b.profit;
}

vector<int> JobScheduling(Job arr[], int n)
{
    int md=0;
    for(int i=0; i<n; i++){
        md=max(md, arr[i].dead);
    }
    vector<int> avail(md+1, -1);
    sort(arr, arr+n, comp);

    int pro=0, count=0;
```

```
    for(int i=0; i<n; i++){
        int j=arr[i].dead;
        while(j>0 && avail[j]!=-1){
            j--;
        }
        if(j>0 && avail[j]==-1){
            avail[j]=i;
            pro+=arr[i].profit;
            count++;
        }
    }
    return {count, pro};
}

int main(){
    int n = 0;
    cout<<"Enter the number of jobs: ";
    cin>>n;
    Job* arr = new Job[n];
    for(int i=0; i<n; i++){
        cout<<"Enter the deadline and profit for job"<<i+1<<":";
        cin>>arr[i].dead>>arr[i].profit;
    }
    vector<int> ans = JobScheduling(arr, n);

    cout<<"count of jobs = "<<ans[0]<<"\nprofit = "<<ans[1];
    return 0;
}
```

Output:

```
PS C:\Users\beadi\Desktop\IDAA\DAA LAB\Assignment 9> cd
 { .\jobSequencing }
Enter the number of jobs: 3
Enter the deadline and profit for job1:2 200
Enter the deadline and profit for job2:2 100
Enter the deadline and profit for job3:1 50
count of jobs = 2
profit = 300
```

b) Activity Selection Problem

Input:

Time of start and end of each job

Output:

The number of jobs that can be completed

Time Complexity:

O(NlogN)

Code:

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

#define int long long int
#define ff first
#define ss second
#define pb push_back
#define pii pair<int, int>

using namespace :: std;

bool comp(pii a, pii b){
    return a.second<b.second;
}

int32_t main(){

    int n=0;
    cout<<"Enter the number of activities: ";
    cin>>n;
    vector<pii> v;

    for(int i=0; i<n; i++){
        pii p;
        cout<<"Enter the start time and end time of job"<<i+1<<":";
        cin>>p.first>>p.second;
        v.push_back(p);
    }

    sort(v.begin(), v.end(), comp);

    int ans=0;
    int end=0;
    int i=0;
    while(i<n){
        if(v[i].first<end){
            i++;
        }
        else{
            end = v[i].second;
```

```
            i++;
            ans++;
        }
    }

    cout<<"The number of jobs that can be done are: "<<ans<<"\n";

    return 0;
}
```

Output:

```
Enter the number of activities: 4
Enter the start time and end time of job1:1 5
Enter the start time and end time of job2:2 4
Enter the start time and end time of job3:4 6
Enter the start time and end time of job4:6 7
The number of jobs that can be done are: 3
```

c) Huffman Code

Input:

-

Output:

Huffman code attached to each letter

Time Complexity:

O(NlogN)

Code:

```
#include<bits/stdc++.h>
#define MAX_TREE_HT 100

// A Huffman tree node
struct MinHeapNode {

    // One of the input characters
    char data;

    // Frequency of the character
    unsigned freq;

    // Left and right child of this node
```

```
    struct MinHeapNode *left, *right;
};

// A Min Heap: Collection of
// min-heap (or Huffman tree) nodes
struct MinHeap {

    // Current size of min heap
    unsigned size;

    // capacity of min heap
    unsigned capacity;

    // Array of minheap node pointers
    struct MinHeapNode** array;
};

// A utility function allocate a new
// min heap node with given character
// and frequency of the character
struct MinHeapNode* newNode(char data, unsigned freq){
    struct MinHeapNode* temp = (struct MinHeapNode*)malloc(
        sizeof(struct MinHeapNode));

    temp->left = temp->right = NULL;
    temp->data = data;
    temp->freq = freq;

    return temp;
}

// A utility function to create
// a min heap of given capacity
struct MinHeap* createMinHeap(unsigned capacity){

    struct MinHeap* minHeap
        = (struct MinHeap*)malloc(sizeof(struct MinHeap));

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

    minHeap->capacity = capacity;

    minHeap->array = (struct MinHeapNode**)malloc(
        minHeap->capacity * sizeof(struct MinHeapNode*));
    return minHeap;
}
```

```
// A utility function to
// swap two min heap nodes
void swapMinHeapNode(struct MinHeapNode** a, struct MinHeapNode** b)

{

    struct MinHeapNode* t = *a;
    *a = *b;
    *b = t;
}

// 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->array[left]->freq
            < minHeap->array[smallest]->freq)
        smallest = left;

    if (right < minHeap->size
        && minHeap->array[right]->freq
            < minHeap->array[smallest]->freq)
        smallest = right;

    if (smallest != idx) {
        swapMinHeapNode(&minHeap->array[smallest],
                        &minHeap->array[idx]);
        minHeapify(minHeap, 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
struct MinHeapNode* extractMin(struct MinHeap* minHeap){
    struct MinHeapNode* temp = minHeap->array[0];
    minHeap->array[0] = minHeap->array[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, struct MinHeapNode* minHeapNode){

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

    while (i
        && minHeapNode->freq
                < minHeap->array[(i - 1) / 2]->freq) {

        minHeap->array[i] = minHeap->array[(i - 1) / 2];
        i = (i - 1) / 2;
    }

    minHeap->array[i] = minHeapNode;
}

// 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);
}

// A utility function to print an array of size n
void printArr(int arr[], int n){
    int i;
    for (i = 0; i < n; ++i)
        printf("%d", arr[i]);

    printf("\n");
}

// Utility function to check if this node is leaf
int isLeaf(struct MinHeapNode* root){
    return !(root->left) && !(root->right);
}

// Creates a min heap of capacity
```

```
// equal to size and inserts all character of
// data[] in min heap. Initially size of
// min heap is equal to capacity
struct MinHeap* createAndBuildMinHeap(char data[], int freq[], int size){
    struct MinHeap* minHeap = createMinHeap(size);
    for (int i = 0; i < size; ++i)
        minHeap->array[i] = newNode(data[i], freq[i]);

    minHeap->size = size;
    buildMinHeap(minHeap);

    return minHeap;
}

// The main function that builds Huffman tree
struct MinHeapNode* buildHuffmanTree(char data[], int freq[], int size){
    struct MinHeapNode *left, *right, *top;

    // Step 1: Create a min heap of capacity
    // equal to size. Initially, there are
    // modes equal to size.
    struct MinHeap* minHeap
        = createAndBuildMinHeap(data, freq, size);

    // Iterate while size of heap doesn't become 1
    while (!isSizeOne(minHeap)) {

        // Step 2: Extract the two minimum
        // freq items from min heap
        left = extractMin(minHeap);
        right = extractMin(minHeap);

        // Step 3: Create a new internal
        // node with frequency equal to the
        // sum of the two nodes frequencies.
        // Make the two extracted node as
        // left and right children of this new node.
        // Add this node to the min heap
        // '$' is a special value for internal nodes, not
        // used
        top = newNode('$', left->freq + right->freq);

        top->left = left;
        top->right = right;

        insertMinHeap(minHeap, top);
    }
```

```
    // Step 4: The remaining node is the
    // root node and the tree is complete.
    return extractMin(minHeap);
}

// Prints huffman codes from the root of Huffman Tree.
// It uses arr[] to store codes
void printCodes(struct MinHeapNode* root, int arr[], int top){

    // Assign 0 to left edge and recur
    if (root->left) {

        arr[top] = 0;
        printCodes(root->left, arr, top + 1);
    }

    // Assign 1 to right edge and recur
    if (root->right) {

        arr[top] = 1;
        printCodes(root->right, arr, top + 1);
    }

    // If this is a leaf node, then
    // it contains one of the input
    // characters, print the character
    // and its code from arr[]
    if (isLeaf(root)) {
        printf("%c: ", root->data);
        printArr(arr, top);
    }
}

// The main function that builds a
// Huffman Tree and print codes by traversing
// the built Huffman Tree
void HuffmanCodes(char data[], int freq[], int size){
    // Construct Huffman Tree
    struct MinHeapNode* root
        = buildHuffmanTree(data, freq, size);

    // Print Huffman codes using
    // the Huffman tree built above
    int arr[MAX_TREE_HT], top = 0;

    printCodes(root, arr, top);
}
```

```
int main(){
    char arr[] = { 'a', 'b', 'c', 'd', 'e', 'f' };
    int freq[] = { 5, 9, 12, 13, 16, 45 };

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

    HuffmanCodes(arr, freq, size);

    return 0;
}
```

Output:

```
PS C:\Users\beadi\Desktop\IDAA\DAA LAB\Assignment 10>
.\huffmanCode }
f: 0
c: 100
d: 101
a: 1100
b: 1101
e: 111
```

d) Kruskals Algorithm for MST

Input:

Edges of the graph and their weights

Output:

The weight of minimum spanning tree

Time Complexity:

O(ElogV) where E is number of edges and V is number of vertices

Code:

```
#include<bits/stdc++.h>
using namespace :: std;

bool comp(pair<pair<int, int>, int> a, pair<pair<int, int>, int> b){
    return a.second<b.second;
}

class Graph{
    public:
        int v;
        vector<pair<pair<int, int>, int> > l;

        Graph(int n){
            this->v=n;
```

```
}

void addEdge(int a, int b, int w){
    l.push_back({{a, b}, w});
}

int findSet(int i, int par[]){
    if(par[i]==-1) return i;

    return par[i] = findSet(par[i], par);
}

void unionSet(int i, int j, int par[], int rank[]){
    int p1 = findSet(i, par);
    int p2 = findSet(j, par);

    if(p1!=p2){
        if(rank[p1]>rank[p2]){
            par[p2]=p1;
            rank[p1]+=rank[p2];
        }
        else{
            par[p1]=p2;
            rank[p2]+=rank[p1];
        }
    }
}

int kruskalsMST(){
    sort(l.begin(), l.end(), comp);
    int par[v];
    int rank[v];
    for(int i=0; i<v; i++){
        par[i]=-1;
        rank[i]=1;
    }

    int ans=0;
    for(auto i:l){
        int a=i.first.first;
        int b=i.first.second;
        int w=i.second;

        if(findSet(a, par)!=findSet(b, par)){
            unionSet(a, b, par, rank);
            ans+=w;
        }
    }
```

```
            return ans;
        }
};

int main(){

    Graph g(4);
    g.addEdge(0, 1, 1);
    g.addEdge(0, 2, 2);
    g.addEdge(0, 3, 2);
    g.addEdge(1, 2, 2);
    g.addEdge(2, 3, 3);
    g.addEdge(1, 3, 3);

    cout<<"MST = "<<g.kruskalsMST();
    return 0;
}
```

Output:

```
PS C:\Users\beadi\Desktop\IDAA\DAA LAB\Assignment 10>
; if ($?) { .\KruskalsAlgorithm }
MST = 5
```

e) Prims Algorithm for MST

Input:

Edges of the graph and their weights

Output:

The weight of the minimum spanning tree

Time Complexity:

O((V + E)logV) where E is number of edges and V is number of vertices

Code:

```
#include<bits/stdc++.h>
using namespace :: std;

class Graph{
    public:
        int v;
```

```
        vector<pair<int, int> > * l;

        Graph(int n){
            this->v=n;
            l=new vector<pair<int, int> > [n];
        }

        void addEdge(int a, int b, int w){
            l[a].push_back({b, w});
            l[b].push_back({a, w});
        }

        int primsMST(){
            priority_queue<pair<int, int>, vector<pair<int, int> > ,
    greater<pair<int, int> > > q;

            bool*vis= new bool [v]{false};

            int ans=0;
            q.push({0, 0});
            while(!q.empty()){
                int to=q.top().second;
                int w=q.top().first;
                q.pop();

                if(vis[to]){
                    continue;
                }

                ans+=w;
                vis[to]=true;

                for(auto i:l[to]){
                    if(!vis[i.first]){
                        q.push({i.second, i.first});
                    }
                }
            }
            return ans;
        }
    };

    int main(){
        Graph g(4);
        g.addEdge(0, 1, 1);
        g.addEdge(0, 2, 2);
        g.addEdge(0, 3, 2);
        g.addEdge(1, 2, 2);
```

```
    g.addEdge(2, 3, 3);
    g.addEdge(1, 3, 3);

    cout<<"MST = "<<g.primsMST();
    return 0;
}
```

Output:

```
PS C:\Users\beadi\Desktop\IDAA\DAA LAB\Assignment 10> cd
sMST }
MST = 5
```

f) Fractional Knapsack

Input:

Integer weights and values

Output:

Integer – Maximum value that can be accommodated

Time Complexity:

O(NlogN)

Code:

```
#include<bits/stdc++.h>
using namespace :: std;

class Item{
    public:
        int weight, value;
};

bool comp(Item a, Item b){
    return (1.0*a.value)/a.weight > (1.0*b.value)/b.weight;
}

double fractionalKnapsack(int W, Item arr[], int n)
{
    double ans=0;
    sort(arr, arr+n, comp);
    int id=0;
    while(W && id<n){
        if(W>=arr[id].weight){
```

```
            W-=arr[id].weight;
            ans+=arr[id].value;
        }
        else{
            ans+=((1.0*arr[id].value)/(1.0*arr[id].weight))*W;
            W=0;
        }
        id++;
    }
    return ans;
}

int main(){
    int n = 0, W = 0;
    cout<<"Enter the number of items: ";
    cin>>n;
    cout<<"Enter the size of knapsack: ";
    cin>>W;
    Item *arr = new Item[n];
    for(int i=0; i<n; i++){
        cout<<"Enter the weight and value for item "<<i+1<<":";
        cin>>arr[i].weight>>arr[i].value;
    }

    cout<<"The maximum value that can be accomodated =
"<<fractionalKnapsack(W, arr, n);
    return 0;
}
```

Output:

```
PS C:\Users\beadi\Desktop\IDAA\DAA LAB\Assignment 10> cd
} ; if ($?) { .\fractionalKnapsack }
Enter the number of items: 4
Enter the size of knapsack: 10
Enter the weight and value for item 1:4 12
Enter the weight and value for item 2:5 6
Enter the weight and value for item 3:3 7
Enter the weight and value for item 4:2 5
The maximum value that can be accomodated = 25.2
```

g) Dijkstras Algortihm for Single Source Shortest Path

Input:

Edges of the graph and their weights

Output:

Array containing the shortest paths of each node from the source node

Time Complexity:

O(V+Elog(V)), where E is number of edges and V is number of vertices

Code:

```
#include<bits/stdc++.h>
using namespace std;

vector<int> shortestPath(vector<vector<int> > roads,int n)
{
    vector<pair<int, int> > l[n+1];
    for(auto i:roads){
        l[i[0]].push_back({i[1], i[2]});
        l[i[1]].push_back({i[0], i[2]});
    }

    vector<int> dist(n+1, INT_MAX);
    vector<int> par(n+1, -1);
    set<pair<int, int> > s;

    dist[1]=0;
    s.insert({0, 1});
    while(!s.empty()){
        auto it=s.begin();
        s.erase(it);
        int node = it->second;
        int nodeDist = it->first;

        for(auto i:l[node]){
            int nbr=i.first;
            int wt = i.second;

            if(dist[nbr]>(nodeDist+wt)){
                par[nbr]=node;

                auto f=s.find({dist[nbr], nbr});
                if(f!=s.end()){
                    s.erase(f);
                }
                dist[nbr]=nodeDist+wt;
                s.insert({dist[nbr], nbr});
            }
        }
    }
```

```
    // for(auto i:par){
    //     cout<<i<<" ";
    // }
    // cout<<"\n";
    stack<int> u;
    int d=n;
    while(par[d]!=-1){
        u.push(d);
        d=par[d];
    }
    u.push(1);

    vector<int> ans;
    while(!u.empty()){
        ans.push_back(u.top());
        u.pop();
    }

    return ans;
}

int main(){
    int n=5;
    vector<vector<int> > edges =
{{1,2,2},{2,5,5},{2,3,4},{1,4,1},{4,3,3},{3,5,1}};
    vector<int> ans=shortestPath(edges, n);
    for(auto i:ans){
        cout<<i<<" ";
    }
    return 0;
}
```

Output:

```
PS C:\Users\beadi\Desktop\IDAA\DAA LAB> cd
sSSSP }
1 4 3 5
```

h) Bellman Ford Algortihm for Single Source Shortest Path

Input:

Edges of the graph and their weights

Output:

Array containing the shortest paths of each node from the source node

Time Complexity:

O(V*E), where E is number of edges and V is number of vertices

Code:

```
#include<bits/stdc++.h>
using namespace :: std;

// SSSP for graphs with negative edges
vector<int> bellmanFord(int n, vector<vector<int> > edges, int src){
    vector<int> dist(n+1, INT_MAX);
    dist[src]=0;

    for(int i=0; i<n-1; i++){
        for(auto i: edges){
            int u=i[0];
            int v=i[1];
            int wt=i[2];

            if(dist[u]!=INT_MAX && dist[v]>(dist[u]+wt)){
                dist[v]=dist[u]+wt;
            }
        }
    }

    // negative edge cycle detection
    for(auto i:edges){
        int u=i[0];
        int v=i[1];
        int wt=i[2];

        if(dist[u]!=INT_MAX && dist[v]>(dist[u]+wt)){
            cout<<"Negative Edge Cycle Present";
            exit(0);
        }
    }

    return dist;
}

int main(){
    int n=0, m=0;
    cin>>n>>m;
    vector<vector<int> > edges;
```

```
    for(int i=0; i<m; i++){
        int u=0, v=0, wt=0;
        cin>>u>>v>>wt;
        edges.push_back({u, v, wt});
    }
    vector<int> dist = bellmanFord(n, edges, 1);

    for(int i=1; i<n+1; i++){
        cout<<"node "<<i<<" is at a distance: "<<dist[i]<<"\n";
    }

    return 0;
}
```

Output:

```
PS C:\Users\beadi\Desktop\IDAA\DAA LAB> cd
}
5
6
1 2 2
2 5 5
2 3 4
1 4 1
4 3 3
3 5 1
node 1 is at a distance: 0
node 2 is at a distance: 2
node 3 is at a distance: 4
node 4 is at a distance: 1
node 5 is at a distance: 5
```

Lab – 11

S No	Program Title	Date of Implementation	Remarks
i.	Dynamic Programming Algorithms	April 28, 2022	

a) 0/1 Knapsack Problem

Input:

Integer weights and values

Output:

Integer – Maximum value that can be accommodated

Time Complexity:

O(n*m) , n = no. of items

m = size of knapsack

Code:

```
#include<bits/stdc++.h>
using namespace :: std;

int knapsack(int wt[], int val[], int n, int W){
    int t[102][1002];
    for(int i=0; i<n+1; i++){
        for(int j=0; j<W+1; j++){
            if(i==0 || j==0){
                t[i][j]=0;
            }
        }
    }

    for(int i=1; i<n+1; i++){
        for(int j=1; j<W+1; j++){
            if(wt[i-1]<=j){
                t[i][j]= max(val[i-1]+t[i-1][j-wt[i-1]], t[i-1][j]);
            }
```

```
            else{
                t[i][j]= t[i-1][j];
            }
        }
    }

    return t[n][W];
}

int main(){
    int n=0;
    cin>>n;
    int*wt=new int [n];
    int*val=new int [n];
    for(int i=0; i<n; i++){
        cin>>wt[i]>>val[i];
    }
    int W=0;
    cin>>W;
    cout<<knapsack(wt, val, n, W);
    return 0;
}
```

Output:

```
PS C:\Users\beadi\Desktop\IDAA\DAA LAB\Assignment 11>
.\01Knapsack }
5
2 5
3 6
6 2
1 9
4 5
10
The maximum value that can be accomodated = 25
```

b) Assembly Line Scheduling

Input:

Time taken at each station for each assembly line along with time for switching

Output:

Minimum time taken to assemble the object

Time Complexity:

O(2*n) , n = no. of stations

Code:

```
#include<bits/stdc++.h>
using namespace :: std;

int t[2][10000];

int solveTab(int**A, int**P, int e[], int s[], int n){
    int t[2][n];

    for(int i=0; i<2; i++){
        t[i][0] = s[i]+A[i][0];
    }

    for(int j=1; j<n; j++){
        for(int i=0; i<2; i++){
            t[i][j] = A[i][j]+min(t[i][j-1], P[!i][j]+t[!i][j-1]);
        }
    }

    return min(e[0]+t[0][n-1], e[1]+t[1][n-1]);
}

int solve(int**A, int**P, int e[], bool path, int i, int n){
    if(i==n-1){
        return e[path]+A[path][i];
    }

    if(t[path][i]!=-1){
        return t[path][i];
    }

    return t[path][i] = A[path][i]+min(solve(A, P, e, path, i+1, n), solve(A,
P, e, !path, i+1, n)+ P[path][i+1]);
}

int main(){
    memset(t, -1, sizeof(t));
    int n=0;
    cin>>n;
    int**A=new int*[2];
```

```
    int**P=new int*[2];
    for(int i=0; i<2; i++){
        A[i]=new int [n];
        P[i]=new int [n];
    }

    int start[2];
    for(int i=0; i<2; i++){
        cin>>start[i];
    }
    for(int i=0; i<2; i++){
        for(int j=0; j<n; j++){
            cin>>A[i][j];
        }
    }
    for(int i=0; i<2; i++){
        for(int j=0; j<n; j++){
            cin>>P[i][j];
        }
    }
    int end[2];
    for(int i=0; i<2; i++){
        cin>>end[i];
    }

    // cout<<min(start[0]+solve(A, P, end, false, 0, n), start[1]+solve(A, P,
end, true, 0, n))<<"\n";
    cout<<"The time taken = "<<solveTab(A, P, end, start, n)<<"\n";
    return 0;
}
```

Output:

```
PS C:\Users\beadi\Desktop\IDAA\DAA LAB\Assignment 11>
cheduling } ; if ($?) { .\AssemblyLineScheduling }
4
10 12
4 5 3 2
2 10 1 4
0 7 4 5
0 9 2 8
18 7
The time taken = 35
```

c) Longest Common Subsequence

Input:

Two strings

Output:

Length of LCS of the given two strings

Time Complexity:

O(n*m) , n and m are lengths of strings

Code:

```
#include<bits/stdc++.h>
using namespace :: std;

int t[102][1002];

int LCS(string x, string y, int n, int m){
    if(n==0 || m==0){
        return t[n][m]=0;
    }
    if(t[n][m]!=-1){
        return t[n][m];
    }
    else{
        if(x[n-1]==y[m-1]){
            return t[n][m]=(1+LCS(x, y, n-1, m-1));
        }
        else{
            return t[n][m]=max(LCS(x, y, n-1, m),LCS(x, y, n, m-1));
        }
    }
}

int main(){
    for(int i=0; i<102; i++){
        for(int j=0; j<1002; j++){
            t[i][j]=-1;
        }
    }
    string x, y;
    cin>>x>>y;
    int n=x.length();
    int m=y.length();
```

```
    cout<<"Length of LCS = "<<LCS(x, y, n, m)<<endl;
    return 0;
}
```

Output:

```
PS C:\Users\beadi\Desktop\IDAA\DAA LAB\Assignment 11>
abbcdab
cbbadb
Length of LCS = 4
```

d) Matrix Chain Multiplication

Input:

Array representing n-1 matrices

Output:

Minimum number of calculations to get the product matrix

Time Complexity:

O(n^3) , n is the size of array

Code:

```
#include<bits/stdc++.h>
using namespace :: std;

int t[1002][1002];

int MCMmemoized(int A[], int i, int j){
    if(t[i][j]!=-1){
        return t[i][j];
    }
    else{
        int temp=INT_MAX;
        for(int k=i; k<j; k++){
            t[i][k]=MCMmemoized(A, i, k);
            t[k+1][j]=MCMmemoized(A, k+1, j);
            int tempAns=t[i][k]+t[k+1][j]+(A[i-1]*A[k]*A[j]);
            if(tempAns<temp){
                temp=tempAns;
            }
        }
        return t[i][j]=temp;
```

```
    }
}

int main(){
    int n=0;
    cin>>n;
    for(int i=0; i<n+1; i++){
        for(int j=0; j<n+1; j++){
            t[i][j]=-1;
            if(i>=j){
                t[i][j]=0;
            }
        }
    }
    int*A=new int [n];
    for(int i=0; i<n; i++){
        cin>>A[i];
    }
    cout<<"Minimum number of calculations required = "<<MCMmemoized(A, 1, n-
1);
    return 0;
}
```

Output:

```
PS C:\Users\beadi\Desktop\IDAA\DAA LAB\Assignment 11>
5
12 24 45 23 41
Minimum number of calculations required = 36696
```

e) Bellman Ford

Input:

Edge list of the graph

Output:

Distance of each node from the source node (1)

Time Complexity:

O(n*m) , n = number of nodes in graph

M = number of edges in the graph

Code:

```
#include<bits/stdc++.h>
using namespace :: std;

vector<int> bellmanFord(int n, vector<vector<int> > edges, int src){
    vector<int> dist(n+1, INT_MAX);
    dist[src]=0;

    for(int i=0; i<n-1; i++){
        for(auto i: edges){
            int u=i[0];
            int v=i[1];
            int wt=i[2];

            if(dist[u]!=INT_MAX && dist[v]>(dist[u]+wt)){
                dist[v]=dist[u]+wt;
            }
        }
    }

    // negative edge cycle detection
    for(auto i:edges){
        int u=i[0];
        int v=i[1];
        int wt=i[2];

        if(dist[u]!=INT_MAX && dist[v]>(dist[u]+wt)){
            cout<<"Negative Edge Cycle Present";
            exit(0);
        }
    }

    return dist;
}

int main(){
    int n=0, m=0;
    cin>>n>>m;
    vector<vector<int> > edges;
    for(int i=0; i<m; i++){
        int u=0, v=0, wt=0;
        cin>>u>>v>>wt;
        edges.push_back({u, v, wt});
    }
    vector<int> dist = bellmanFord(n, edges, 1);

    for(int i=1; i<n+1; i++){
        cout<<"node "<<i<<" is at a distance: "<<dist[i]<<"\n";
```

```
    }

    return 0;
}
```

Output:

```
PS C:\Users\beadi\Desktop\IDAA\DAA LAB\Assignment 11>
{ .\BellmanFord }
4 4
1 4 5
1 3 4
2 4 7
1 2 4
node 1 is at a distance: 0
node 2 is at a distance: 4
node 3 is at a distance: 4
node 4 is at a distance: 5
```

f) Floyd Warshall

Input:

Adjacency Matrix of the graph

Output:

Distance between each pair of nodes (represented as a distance matrix)

Time Complexity:

O(n^3) , n = number of nodes in graph

Code:

```
#include<bits/stdc++.h>
using namespace :: std;

vector<vector<int> > floyd_warshall(vector<vector<int> > graph){
    vector<vector<int> > dist(graph);
    int v=graph.size();

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

```
            for(int j=0; j<v; j++){
                if(dist[i][k]!=INT_MAX && dist[k][j]!=INT_MAX &&
dist[i][j]>(dist[i][k]+dist[k][j])){
                    dist[i][j]=dist[i][k]+dist[k][j];
                }
            }
        }
    }

    return dist;
}

int main(){
    vector<vector<int> > graph={{0, INT_MAX, -2, INT_MAX},{4, 0, 3,
INT_MAX},{INT_MAX, INT_MAX, 0, 2},{INT_MAX, -1, INT_MAX, 0}};
    vector<vector<int> > ans = floyd_warshall(graph);

    cout<<"Shortest Distance Matrix for the given graph: \n";

    for(auto i: ans){
        for(int j=0; j<graph.size(); j++){
            cout<<i[j]<<" ";
        }
        cout<<"\n";
    }

    return 0;
}
```

Output:

```
PS C:\Users\beadi\Desktop\IDAA\DAA LAB\Assignment 11>
$?) { .\FloydWarshall }
Shortest Distance Matrix for the given graph:
0 -1 -2 0
4 0 2 4
5 1 0 2
3 -1 1 0
```

g) Optimal Binary Search Tree

Input:

Key and Frequency array

Output:

The minimum cost of the searches in the optimal BST

Time Complexity:

O(n^4) , n = number of nodes

Code:

```
#include <bits/stdc++.h>
using namespace std;

int sum(int freq[], int i, int j);

int optCost(int freq[], int i, int j){
    if (j < i)
        return 0;
    if (j == i)
        return freq[i];

    int fsum = sum(freq, i, j);

    int min = INT_MAX;

    for (int r = i; r <= j; ++r){
        int cost = optCost(freq, i, r - 1) + optCost(freq, r + 1, j);
        if (cost < min)
            min = cost;
    }

    return min + fsum;
}


int optimalSearchTree(int keys[], int freq[], int n){
    return optCost(freq, 0, n - 1);
}

int sum(int freq[], int i, int j)
```

```
{
    int s = 0;
    for (int k = i; k <= j; k++)
    s += freq[k];
    return s;
}

int main()
{
    int keys[] = {10, 12, 20};
    int freq[] = {34, 8, 50};
    int n = sizeof(keys) / sizeof(keys[0]);
    cout << "Cost of Optimal BST is "<< optimalSearchTree(keys, freq, n);
    return 0;
}
```

Output:

```
PS C:\Users\beadi\Desktop\IDAA\DAA LAB\Assignment 11>
.\OptimalBST }
Cost of Optimal BST is 142
```

---------- THANK YOU----------
