DESIGN AND ANALYSIS OF ALGORITHMS

Practical File

COCSC06

Name : Ekankaar Khera

Branch : Mathematics And Computing

Roll No. : 2022UCM2343

INDEX

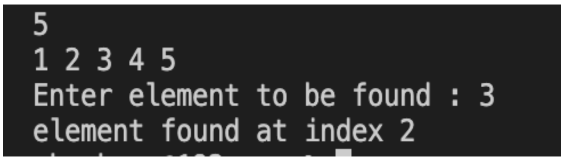
Q1 : Write a code to find a number in an array using binary search

CODE :

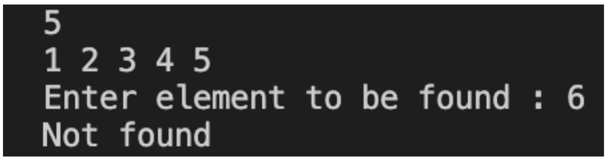
```
#include <iostream>
#include <climits>
#include <math.h>
#include <string>
#include <cstring> //using strlen in char arrays
#include <set>
#include <algorithm>
#include <vector>
#include <fstream>
#include <list>
#include <stack>
#include <queue>
#include <unordered_map>
#include <map>
#include <set>
#include <cstdlib> // for absolute function

using namespace std;
#define mod 1000000007
#define int  long long int


int32_t main()
{
   int n;
   cin >> n;
   int arr[n];
   for (int i = 0; i < n; i++)
   {
      cin >> arr[i];
   }

   cout << "Enter element to be found : ";
   int element;
   cin >> element;

   int s = 0, e = n - 1;

   int index = -1;
   while (s <= e)
   {
      int mid = (s + e) / 2;

      if (arr[mid] == element)
```

```
        {
            index = mid;
            break;
        }

        else if (element < arr[mid])
        {
            e = mid - 1;
        }

        else
        {
            s = mid + 1;
        }


    }

    if (index == -1)
    {

        cout << "Not found" << endl;
    }

    else
    {
        cout << "element found at index " << index << endl;

    }

}
```

OUTPUT :



```
5
1 2 3 4 5
Enter element to be found : 3
element found at index 2
```



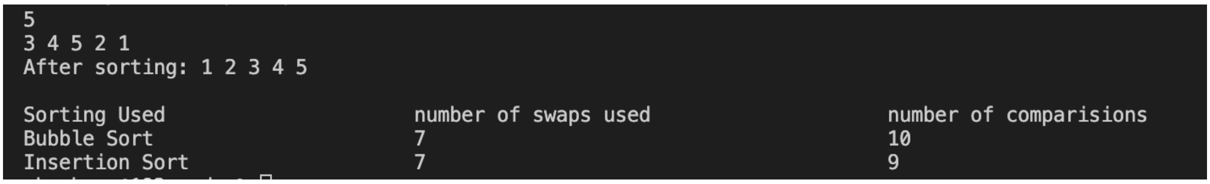
```
5
1 2 3 4 5
Enter element to be found : 6
Not found
```

Q2 : Implement Bubble sort and Insertion sort

CODE :

```
#include<stdio.h>
#include<stdlib.h>
#include <string.h>
#include <limits.h>


int swaps_b = 0, comparions_b = 0;

void bubble_sort(int arr[], int n)
{
   for (int i = 0; i < n - 1; i++)
   {
      for (int j = 0; j < n - 1 - i; j++)
      {
         comparions_b ++;
         if (arr[j] > arr[j + 1])
         {
            swaps_b ++;
            int temp = arr[j];
            arr[j] = arr[j + 1];
            arr[j + 1] = temp;

         }
      }
   }

}

int swaps_i = 0, comparions_i = 0;

void insertion_sort(int arr[], int n)
{
   for (int i = 1; i < n; i++)
   {
      int element = arr[i];
      for (int j = i - 1; j >= 0; j--)
      {
         comparions_i ++;
         if (element < arr[j])
         {
            swaps_i ++;
            arr[j + 1] = arr[j];

            if (j == 0)
            {
                    arr[j] = element;
                }
            }

            else
            {
                arr[j + 1] = element;
                break;
            }
        }


    }
}

int main()
{
    int n;
    scanf("%d", &n);

    int arr[n];
    for (int i = 0; i < n; i++)
    {
        scanf("%d", &arr[i]);
    }

    int temp[n];
    for (int i = 0; i < n; i++)
    {
        temp[i] = arr[i];
    }


    bubble_sort(temp, n);


    for (int i = 0; i < n; i++)
    {
        temp[i] = arr[i];
    }


    insertion_sort(temp, n);
    printf("After sorting: ");
    for (int i = 0; i < n; i++)
    {
        printf("%d ", temp[i]);
    }

    printf("\n\n");
    printf("Sorting Used\t\t\t number of swaps used\t\t\t number of comparisions\n");

    printf("Bubble Sort\t\t\t %d\t\t\t\t\t %d\n", swaps_b, comparions_b);
    printf("Insertion Sort\t\t\t %d\t\t\t\t\t %d\n", swaps_i, comparions_i);

}
```

OUTPUT :



```
5
3 4 5 2 1
After sorting: 1 2 3 4 5

Sorting Used                      number of swaps used                    number of comparisions
Bubble Sort                       7                                       10
Insertion Sort                    7                                       9
```

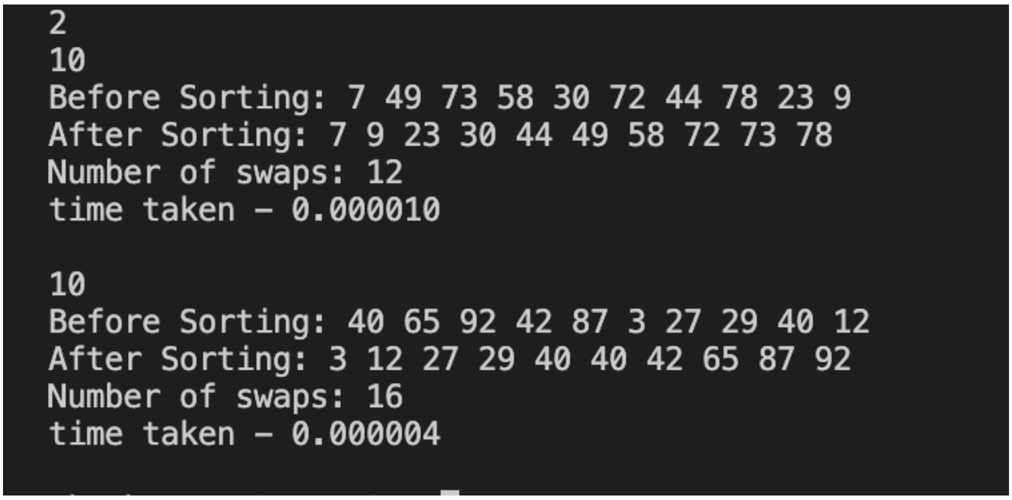
Q3 : Sort a given set of elements using the Quick sort method and determine the time required to sort the elements.

CODE :

```
#include<stdio.h>
#include<stdlib.h>
#include <string.h>
#include <limits.h>
#include <time.h>

int swaps;
int make_partion(int arr[], int s, int e)
{

    // pivoting last element

    // j represents the first segment
    int j = s - 1;
    for (int i = s; i < e; i++)
    {
        if (arr[i] <= arr[e])
        {
            swaps ++;

            int temp = arr[j + 1];
            arr[j + 1] = arr[i];
            arr[i] = temp;
            j++;
        }
    }
    int index = j + 1;

    // swapping the pivot element into the right place
    swaps ++;
    int temp = arr[index];
    arr[index] = arr[e];
    arr[e] = temp;

    return index;
}

void quicksort(int arr[], int s, int e)
{
    if (s >= e)
    {
        return;
    }

    int index = make_partion(arr, s, e);

    quicksort(arr, s, index - 1);
    quicksort(arr, index + 1, e);

}

int main()
{
    int t;
    scanf("%d", &t);

    for (int i = 0; i < t; i++)
    {
        swaps = 0;
        int n;
        scanf("%d", &n);

        int arr[n];


        for (int i = 0; i < n; i++)
        {   // Random Generator.
            arr[i] = (rand()) % 100;
        }

        printf("Before Sorting: ");
        for (int i = 0; i < n; i++)
        {
```

```
      printf("%d ", arr[i]);
    }

    printf("\n");

    clock_t start, end;


    start = clock();
    quicksort(arr, 0, n - 1);

    end = clock();
    // time time_used = difftime (start, end);
    double diff_t;
    diff_t = difftime(end, start) / (CLOCKS_PER_SEC);

     printf("After Sorting: ");
    for (int i = 0; i < n; i++)
    {
      printf("%d ", arr[i]);
    }

    printf("\n");

    printf("Number of swaps: %d\n", swaps);
    printf("time taken - %f\n\n", diff_t);



  }
}
```

OUTPUT :



```
2
10
Before Sorting: 7 49 73 58 30 72 44 78 23 9
After Sorting: 7 9 23 30 44 49 58 72 73 78
Number of swaps: 12
time taken - 0.000010

10
Before Sorting: 40 65 92 42 87 3 27 29 40 12
After Sorting: 3 12 27 29 40 40 42 65 87 92
Number of swaps: 16
time taken - 0.000004
```

Ques 4 : Write a program to implement Red Black Tree

CODE :

```
#include <iostream>
#include <climits>
#include <math.h>
#include <string>
#include <cstring> //using strlen in char arrays
#include <set>
#include <algorithm>
#include <vector>
#include <fstream>
#include <list>
#include <stack>
#include <queue>
#include <unordered_map>
#include <map>
#include <set>
#include <cstdlib> // for absolute function

using namespace std;

#define ll long long

// things left to do-
// change parents after rotation  / tried to do, but something is wrong  / done

class node
{
public:
    int data;
    node * left;
    node * right;
    string color;
    node * parent;

    node(int d)
    {
        data = d;
        left = NULL;
        right = NULL;

    }

};

void print_in(node * root);
```

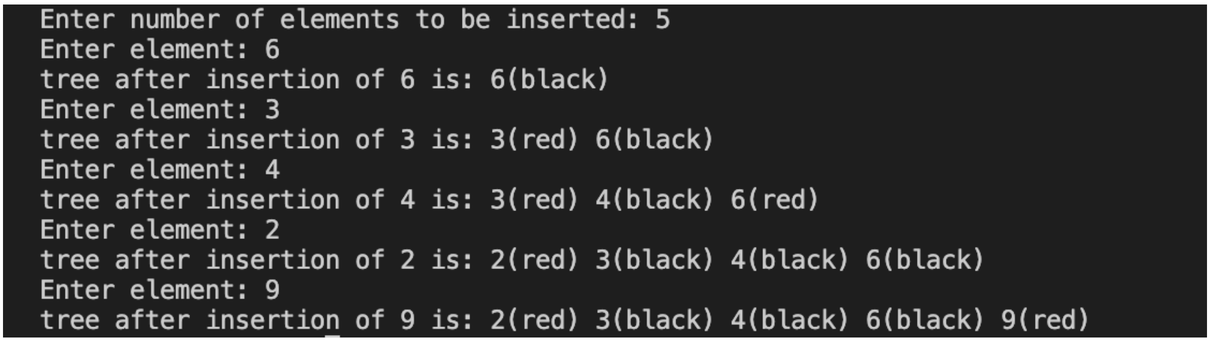
```
node * og_root = NULL;


// we rotate towards the right
node * ll_rotation(node * root)
{
    node * temp = root->left;

    node * temp1 = temp->right;

    node * temp2 = root->parent;

    root->parent = temp;
    temp->parent = temp2;

    temp->right = root;

    root->left = NULL;

    if (temp1 != NULL)
    {
        root->left = temp1;
        temp1->parent = root;
    }
    return temp;
}

// we rotate towards the left
node * rr_rotation(node * root)
{
    node * temp = root->right;

    node * temp1 = temp->left;

    node * temp2 = root->parent;

    root->parent = temp;
    temp->parent = temp2;

    temp->left = root;
    root->right = NULL;

    if (temp1 != NULL)
    {
        root->right = temp1;
        temp1->parent = root;
    }

    return temp;
}
```

```
void recolor(node * root)
{
  if (root->color == "black")
  {
    root->color = "red";
  }

  else
  {
    root->color = "black";
  }
}

bool is_returning_node_the_root_node = false;
node * returning = NULL;
bool did_rotation_happen = false;
void check_and_resolve_RR(node * root, int d)
{
  if (root->parent->color == "black")
    {
      // do nothing
    }

    else
    {
      node * parent = root->parent;
      node * grandparent = parent->parent;

      node * sibling = NULL;

      // that is we got the sibling
      if (grandparent->left != parent)
      {
        sibling = grandparent->left;
      }

      else
      {
        sibling = grandparent->right;
      }

      // if sibling is null color is black
      if (sibling == NULL || sibling->color == "black")
      {
        did_rotation_happen = true;
        node * temp = grandparent;
        char rotation[2];
        for (int i = 0; i < 2; i++)
        {
```

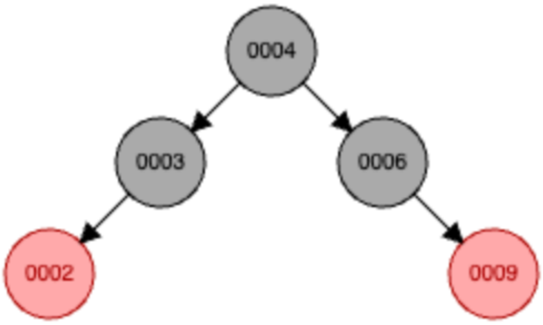
```
    if (d <= temp->data)
    {
        rotation[i] = 'L';
        temp = temp->left;
    }

    else
    {
        rotation[i] = 'R';
        temp = temp->right;
    }

}

returning = NULL;
if (rotation[0] == 'L' && rotation[1] == 'L')
{
    recolor(grandparent);
    recolor(parent);
    returning = ll_rotation(grandparent);
}

else if (rotation[0] == 'R' && rotation[1] == 'R')
{

    recolor(grandparent);
    recolor(parent);
    returning = rr_rotation(grandparent);

}

else if (rotation[0] == 'L' && rotation[1] == 'R')
{
    recolor(grandparent);
    recolor(root);
    grandparent->left = rr_rotation(grandparent->left);
    returning = ll_rotation(grandparent);
}

else if (rotation[0] == 'R' && rotation[1] == 'L')
{
    recolor(grandparent);
    recolor(root);
    grandparent->right = ll_rotation(grandparent->right);
    returning = rr_rotation(grandparent);
}

// problem might be here
if (returning->parent != NULL)
{
```

```
                    node * bigp = returning->parent;
                    if (bigp->left == grandparent)
                    {
                        bigp->left = returning;
                    }

                    else
                    {
                        bigp->right = returning;
                    }
                }

                else
                {
                    is_returning_node_the_root_node = true;
                }
            }

            else
            {
                recolor(sibling);
                recolor(parent);

                if (grandparent == og_root)
                {
                    // do nothing
                }

                else
                {
                    recolor(grandparent);

                    check_and_resolve_RR(grandparent, grandparent->data);
                }
            }

        }
}

bool is_root_node = true;
node * insert_in_red_black_tree(node * root, int d, node * parent)
{
    if (root == NULL)
    {
        root = new node(d);

        if (is_root_node)
        {
            root->color = "black";
            root->parent = NULL;
            is_root_node = false;
        }

        else
        {
            root->color = "red";
            root->parent = parent;

            if (root->data <= parent->data)
            {
                parent->left = root;
            }

            else
            {
                parent->right = root;
            }
            check_and_resolve_RR(root, d);


        }

        return root;

    }

    if (d <= root->data)
    {
        // root->left->parent = root;
        // root->left = insert_in_red_black_tree(root->left, d, root);

        // node * temp = insert_in_red_black_tree(root->left, d, root);

        // if (!did_rotation_happen)
        // {
        //     root->left = temp;
        // }
        insert_in_red_black_tree(root->left, d, root);
    }

    else
    {
        // root->right->parent = root;
        // root->right = insert_in_red_black_tree(root->right, d, root);

        // node * temp = insert_in_red_black_tree(root->right, d, root);

        // if (!did_rotation_happen)
        // {
        //     root->right = temp;
        // }

        insert_in_red_black_tree(root->right, d, root);
    }


    return root;


}

void print_in(node * root)
{
    if (root == NULL)
    {
        return;
    }

    print_in(root->left);
    cout << root->data << "(" << root->color << ")" << " ";
    print_in(root->right);

}
int main()
{
    cout << "Enter number of elements to be inserted: ";
    int n;
    cin >> n;

    node * root = NULL;
    for (int i = 0; i < n; i++)
    {
        int d;
        cout << "Enter element: ";
        cin >> d;

        root = insert_in_red_black_tree(root, d, NULL);

        if (is_returning_node_the_root_node)
        {
            root = returning;
        }
        og_root = root;

        cout << "tree after insertion of " << d << " is: ";


        print_in(root);
        cout << endl;
        is_returning_node_the_root_node = false;
        did_rotation_happen = false;

    }

}
```

OUTPUT :



```
Enter number of elements to be inserted: 5
Enter element: 6
tree after insertion of 6 is: 6(black)
Enter element: 3
tree after insertion of 3 is: 3(red) 6(black)
Enter element: 4
tree after insertion of 4 is: 3(red) 4(black) 6(red)
Enter element: 2
tree after insertion of 2 is: 2(red) 3(black) 4(black) 6(black)
Enter element: 9
tree after insertion of 9 is: 2(red) 3(black) 4(black) 6(black) 9(red)
```



Ques 5 : Implement Radix Sort and Bucket Sort algorithms and compare their performance on a set of randomly generated integers.

CODE :

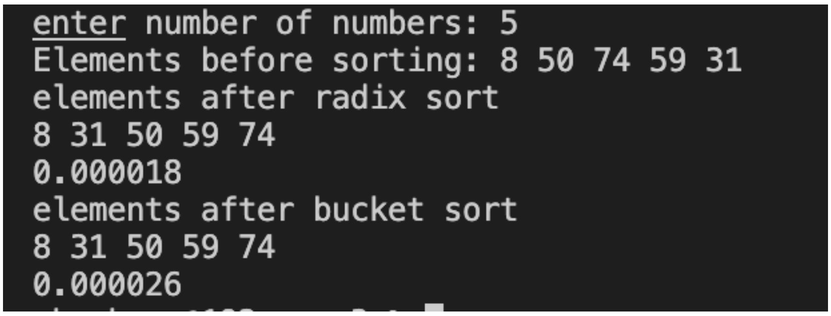
```
#include <iostream>
#include <climits>
#include <math.h>
#include <string>
#include <cstring> //using strlen in char arrays
#include <set>
#include <algorithm>
#include <vector>
#include <fstream>
#include <list>
#include <stack>
#include <queue>
#include <unordered_map>
#include <map>
#include <set>
#include <cstdlib> // for absolute function

using namespace std;

#define ll long long

int get_digit(int n, int d)
{
   int ans = 0;
   while (d != 0)
   {
      ans = n % 10;
      n /= 10;
      d--;
   }

   return ans;
}


int sorted_arr[10000];
// first will be digit, second will be the number
void count_sort(pair< int, int >  arr[], int n)
{
   int counting_array[10] = {0};

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

```
    {
        int digit = arr[i].first;
        counting_array[digit] ++;
    }

    // cout << 5434545 << endl;
    // make counting_arr as csum arr
    int csum = counting_array[0];
    for (int i = 1; i < 10; i++)
    {
        counting_array[i] += csum;
        csum = counting_array[i];
    }

    // cout << 5434545 << endl;
    for (int i = n - 1; i >= 0; i--)
    {
        int num = arr[i].first;
        sorted_arr[counting_array[num] - 1] = arr[i].second;
        counting_array[num] --;
    }


}


void bucketSort(int arr[], int n, int max_el, int min_el)
{
    int bucket_size = 10;


    int range = max_el - min_el + 1;
    int bucket_count = (range / bucket_size) + 1;

    vector <int> buckets[bucket_count];

    for (int i = 0; i < n; i++)
    {
        int index = (arr[i] - min_el) / bucket_size;
        buckets[index].push_back(arr[i]);
    }

    int index = 0;
    for (int i = 0; i < bucket_count; i++)
    {
        sort(buckets[i].begin(), buckets[i].end());

        for (int j = 0; j < buckets[i].size(); j++)
        {
            arr[index] = buckets[i][j];
            index ++;
        }
    }

    for (int i = 0; i < n; i++)
    {
        cout << arr[i] << " ";
    }
    cout << endl;
}

int main()
{

    cout << "enter number of numbers: ";
    int n;
    cin >> n;


    int arr[n];

    int largest = INT_MIN;
    int smallest = INT_MAX;

    for (int i = 0; i < n; i++)
    {
        arr[i] = rand() % 100 + 1;

        sorted_arr[i] = arr[i];

        largest = max(largest, arr[i]);
        smallest = min(smallest, arr[i]);

    }

    cout << "Elements before sorting: ";
    for (int i = 0; i < n; i++)
    {
        cout << arr[i] <<  " ";
    }

    cout << endl;
    int Largest = largest;
    clock_t start, end;
    start = clock();
    // count number of digits of largest number

    int count  = 0;
    while (Largest != 0)
    {
```

```
   count ++;
   Largest /= 10;

}

pair < int, int > to_sort[n];

int d = 1;
while (d != count + 1)
{


for (int i = 0; i < n; i++)
{
   to_sort[i].second = sorted_arr[i];
   to_sort[i].first = get_digit(sorted_arr[i], d);

}

count_sort(to_sort, n);
d++;
}


cout << "elements after radix sort" << endl;

for (int i = 0; i < n; i++)
{
   cout << sorted_arr[i] << " ";
}


cout << endl;
 cout << fixed << setprecision(6);
end = clock();
double diff_t;
   diff_t = difftime(end, start) / (CLOCKS_PER_SEC);
cout << diff_t << endl;

// gcl123

// bucket sort

cout << "elements after bucket sort" << endl;

start = clock();

bucketSort(arr, n, largest, smallest);

 end = clock();
```

```
    diff_t = difftime(end, start) / (CLOCKS_PER_SEC);

   cout << diff_t << endl;
}
```

OUTPUT :



```
enter number of numbers: 5
Elements before sorting: 8 50 74 59 31
elements after radix sort
8 31 50 59 74
0.000018
elements after bucket sort
8 31 50 59 74
0.000026
```

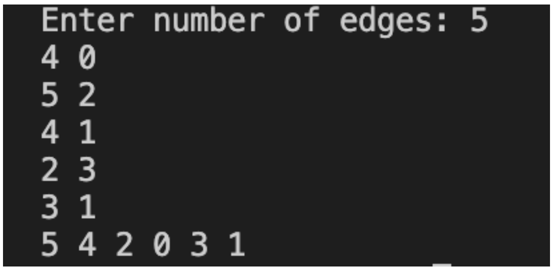
Ques 6 : (A) Obtain the topological sorting of vertices in a digraph
(B) Compute the Transitive closure of a given directed graph using Warshall algorithm

(A)
CODE :

```
#include <iostream>
#include <climits>
#include <math.h>
#include <string>
#include <cstring> //using strlen in char arrays
#include <set>
#include <algorithm>
#include <vector>
#include <fstream>
#include <list>
#include <stack>
#include <queue>
#include <unordered_map>
#include <map>
#include <set>
#include <cstdlib> // for absolute function

using namespace std;

#define ll long long
```

```
class graph
{
   map<int, list<int> > l;

public:
   void insert_edge(int x, int y)
   {
      l[x].push_back(y);

   }

   void topological_sort()
   {
      unordered_map<int, int> indegree;
      // find indegrees

      for (auto p: l)
      {
         indegree[p.first] = 0;
      }

      for (auto p: l)
      {
         for (auto x: p.second)
         {
            indegree[x] ++;
         }
      }

      // find vertices with zero indegree
      queue<int> q;

      for (auto p: indegree)
      {
         if (p.second == 0)
         {
            q.push(p.first);
         }
      }

      while(!q.empty())
      {
         int front = q.front();
         cout << front << " ";
         for (auto nbr: l[front])
         {
            indegree[nbr] --;
            if (indegree[nbr] == 0)
            {
                    q.push(nbr);
                }
            }
            q.pop();
        }
        cout << endl;
    }


};

int main()
{

    cout << "Enter number of edges: ";
    int e;
    cin >> e;

    graph g;

    for (int i = 0; i < e; i++)
    {
        int x, y;
        cin >> x >> y;
        g.insert_edge(x, y);
    }

    g.topological_sort();
}
```

OUTPUT :



```
Enter number of edges: 5
4 0
5 2
4 1
2 3
3 1
5 4 2 0 3 1
```

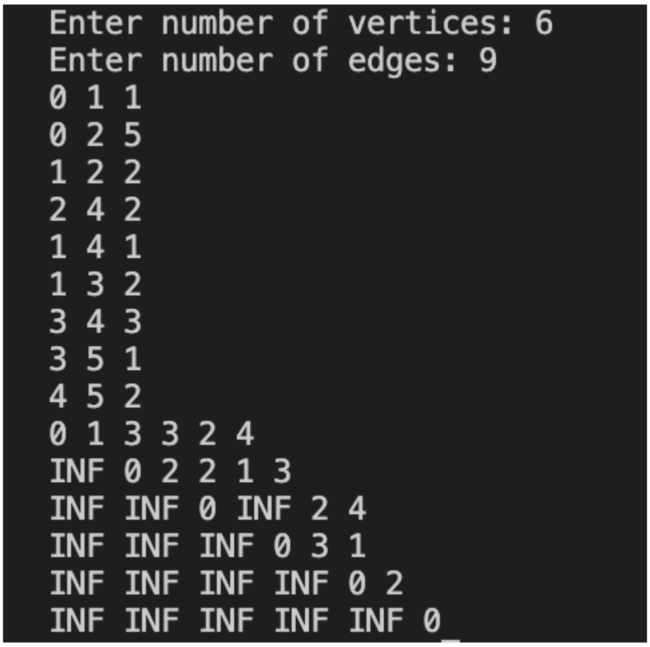
(B)

CODE –

```
#include <iostream>
#include <climits>
#include <math.h>
#include <string>
#include <cstring> //using strlen in char arrays
#include <set>
#include <algorithm>
#include <vector>
#include <fstream>
#include <list>
#include <stack>
#include <queue>
#include <unordered_map>
#include <map>
#include <set>
#include <cstdlib> // for absolute function

using namespace std;

#define ll long long

void floyd_warshall(vector < vector <int > > 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++)
         {
            // we can either skip it, or include it, it wont make a difference
            // if (i == k || j == k)
            // {
            //     continue;
            // }

            if (graph[i][k] != INT_MAX && graph[k][j] != INT_MAX && graph[i][j] >
graph[i][k] + graph[k][j])
            {
               graph[i][j] = graph[i][k] + graph[k][j];
            }

         }
```

```
        }
    }

    for (int i = 0; i < v; i++)
        {
            for (int j = 0; j < v; j++)
            {
                if (graph[i][j] == INT_MAX)
                {
                    cout << "INF" << " ";
                    continue;
                }
                cout << graph[i][j] << " ";

            }

            cout << endl;
        }

}

int main()
{

    int v;
    cout << "Enter number of vertices: ";
    cin >> v;

    int e;
    cout << "Enter number of edges: ";
    cin >> e;

    vector <vector <int> > graph(v, vector<int>(v, INT_MAX));
    for (int i = 0; i < e; i++)
    {
        int x, y, d;
        cin >> x >> y >> d;

        graph[x][y] = d;

    }

    for (int i = 0; i < v; i++)
    {
        graph[i][i] = 0;
    }

    floyd_warshall(graph);
}
```

OUTPUT :



```
Enter number of vertices: 6
Enter number of edges: 9
0 1 1
0 2 5
1 2 2
2 4 2
1 4 1
1 3 2
3 4 3
3 5 1
4 5 2
0 1 3 3 2 4
INF 0 2 2 1 3
INF INF 0 INF 2 4
INF INF INF 0 3 1
INF INF INF INF 0 2
INF INF INF INF INF 0_
```

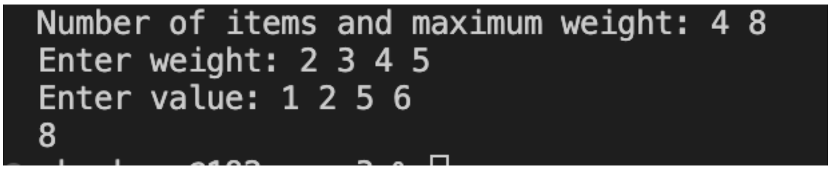
Ques 7 : Implement 0/1 Knapsack problem using Dynamic Programming.

CODE :

```
#include <iostream>
#include <climits>
#include <math.h>
#include <string>
#include <cstring> //using strlen in char arrays
#include <set>
#include <algorithm>
#include <vector>
#include <fstream>
#include <list>
#include <stack>
#include <queue>
#include <unordered_map>
#include <map>
#include <set>
#include <cstdlib> // for absolute function
#define mod 1000000007

using namespace std;
```

```
#define ll long long
#define MAX 50000000000

int dp[1005][10005];
ll int knapsack(int wt[], int val[], int n, int w)
{
    if (n == -1)
    {
        return 0;
    }

    if (dp[w][n] != -1)
    {
        return dp[w][n];
    }
    ll int ans = 0;

    // include
    ll int op1 = 0;


    if (wt[n] <= w)
    {
        op1 = val[n] + knapsack(wt, val, n - 1, w - wt[n]);
    }

    // exclude
    ll int op2 = knapsack(wt, val, n - 1, w);


    ans = max(op1, op2);

    return dp[w][n] = ans;

}

int main()
{
    cout << "Number of items and maximum weight: ";
    int n, max_weight;
    cin >> n >> max_weight;
    int wt[n];
    int val[n];
    memset(dp, -1, sizeof(dp));

    cout << "Enter weight: ";
    for (int i = 0; i < n; i++)
    {
        cin >> wt[i];
    }

    cout << "Enter value: ";

    for (int i = 0; i < n; i++)
    {
        cin >> val[i];
    }

    cout << knapsack(wt, val, n - 1, max_weight) << endl;


}
```

OUTPUT :



```
Number of items and maximum weight: 4 8
Enter weight: 2 3 4 5
Enter value: 1 2 5 6
8
```

Ques 8 : From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijikstra’s algorithm

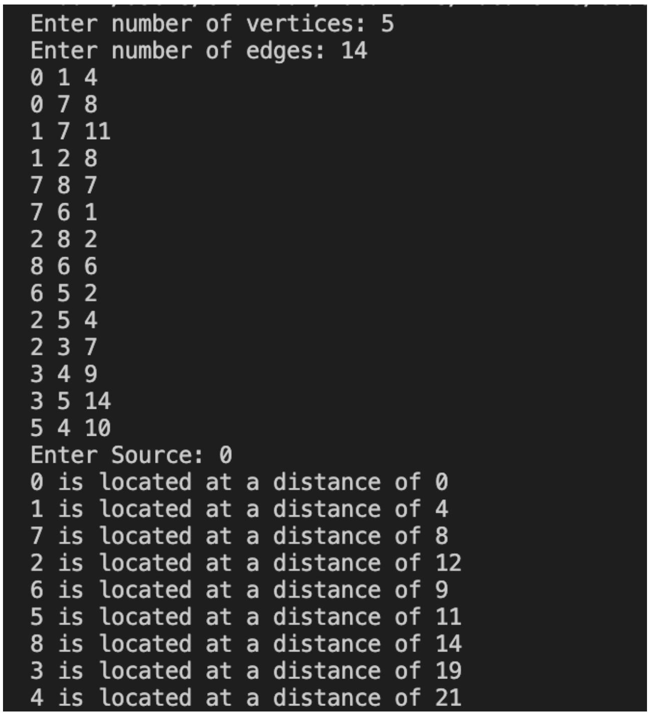
CODE :

```
#include <iostream>
#include <climits>
#include <math.h>
#include <string>
#include <cstring> //using strlen in char arrays
#include <set>
#include <algorithm>
#include <vector>
#include <fstream>
#include <list>
#include <stack>
#include <queue>
#include <unordered_map>
#include <map>
#include <set>
#include <cstdlib> // for absolute function

using namespace std;

#define ll long long

class graph
{
   int v;
   unordered_map < int, list < pair < int, int > > > l;

public:

   graph (int V)
   {
      v = V;
   }

   void insert_edge(int x, int y, int w)
   {
      l[x].push_back({y, w});
      l[y].push_back({x, w});
   }

   void dijisktraSSSP(int src)
   {
      unordered_map <int, int> dist;
```

```
    for (auto p: l)
    {
        dist[p.first] = INT_MAX;
    }

    dist[src] = 0;


    // let an element be weight, source
    set < pair < int, int > > s;

    s.insert({0, src});

    while (s.size() != 0)
    {
        auto top = *s.begin();
        int weight = top.first;
        int node = top.second;

        s.erase(s.begin());

        for (auto nbr: l[node])
        {
            int distance = dist[node] + nbr.second;

            if (distance < dist[nbr.first])
            {

                // if the element is already in the set we must remove it

                auto f = s.find({dist[nbr.first], nbr.first});

                if (f != s.end())
                {
                    s.erase(f);
                }

                s.insert({distance, nbr.first});
                dist[nbr.first] = distance;
            }
        }
    }

    // lets print distance to all other nodes from source
    for (auto p: dist)
    {
        cout << p.first << " is located at a distance of " << p.second << endl;
    }

}
```

```
};

int main()
{
   int v;
   cout << "Enter number of vertices: ";
   cin >> v;
   graph g(v);

   int e;
   cout << "Enter number of edges: ";
   cin >> e;

   for (int i = 0; i < e; i++)
   {
      int x, y, w;
      cin >> x >> y >> w;
      g.insert_edge(x, y, w);
   }

   cout << "Enter Source: ";
   int src;
   cin >> src;

   g.dijisktraSSSP(src);



}
```

OUTPUT :



```
Enter number of vertices: 5
Enter number of edges: 14
0 1 4
0 7 8
1 7 11
1 2 8
7 8 7
7 6 1
2 8 2
8 6 6
6 5 2
2 5 4
2 3 7
3 4 9
3 5 14
5 4 10
Enter Source: 0
0 is located at a distance of 0
1 is located at a distance of 4
7 is located at a distance of 8
2 is located at a distance of 12
6 is located at a distance of 9
5 is located at a distance of 11
8 is located at a distance of 14
3 is located at a distance of 19
4 is located at a distance of 21
```

Ques 9 :Find Minimum Cost Spanning Tree of a given undirected graph using Kruskal's algorithm

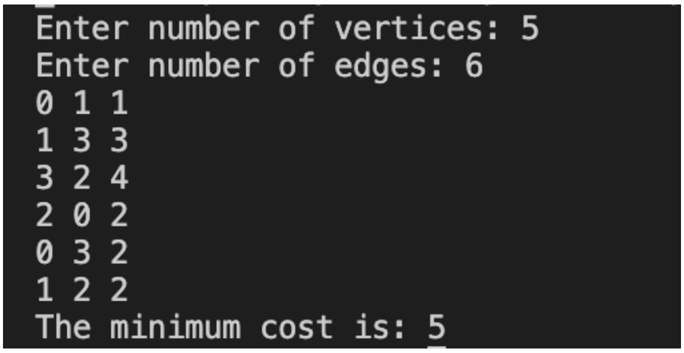
CODE :

```
#include <iostream>
#include <climits>
#include <math.h>
#include <string>
#include <cstring> //using strlen in char arrays
#include <set>
#include <algorithm>
#include <vector>
#include <fstream>
#include <list>
#include <stack>
#include <queue>
#include <unordered_map>
#include <map>
#include <set>
#include <cstdlib> // for absolute function

using namespace std;

#define ll long long


// input -
// Enter number of vertices: 5
// Enter number of edges: 6
// 0 1 1
// 1 3 3
// 3 2 4
// 2 0 2
// 0 3 2
// 1 2 2

class graph
{
   // we are making a list of edge pair

   int v;
   vector <pair < int, pair <int, int > > > l;

public:

   graph(int V)
   {
      v = V;
   }
```

```
void add_edge(int x, int y, int w)
{
   l.push_back({w, {x, y}});
}

int find(int i, int parent[])
{
   if (parent[i] == -1)
   {
      return i;
   }

   parent[i] = find(parent[i], parent);
   return parent[i];
}

int krushal_mst()
{
   int parent[v];
   int rank[v];

   for (int i = 0; i < v; i++)
   {
      parent[i] = -1;
      rank[i] = 1;
   }

   sort(l.begin(), l.end());

   int ans = 0;

   for (auto p: l)
   {
      int x = p.second.first;
      int y = p.second.second;
      int w = p.first;

      // union part
      int s1 = find(x, parent);
      int s2 = find(y, parent);

      if (s1 != s2)
      {

         if (rank[s1] >= rank[s2])
         {
            parent[s2] = s1;
            rank[s1] += rank[s2];
         }
```

```
                else
                {
                    parent[s1] = s2;
                    rank[s2] += rank[s1];
                }

                ans += w;


            }

            else
            {

                // return true;
            }

        }



        return ans;
        // return false;
    }


};



int main()
{
    cout << "Enter number of vertices: ";
    int v;
    cin >> v;
    graph g(v);

    cout << "Enter number of edges: ";
    int e;
    cin >> e;

    for (int i = 0; i < e; i++)
    {
        int x, y, w;
        cin >> x >> y >> w;
        g.add_edge(x, y, w);
    }
    cout << "The minimum cost is: " << g.krushal_mst() << endl;


}
```

OUTPUT :



```
Enter number of vertices: 5
Enter number of edges: 6
0 1 1
1 3 3
3 2 4
2 0 2
0 3 2
1 2 2
The minimum cost is: 5
```

Ques 10 : A) Print all the nodes reachable from a given starting node in a digraph using BFS method.
B) Check whether a given graph is connected or not using DFS method.

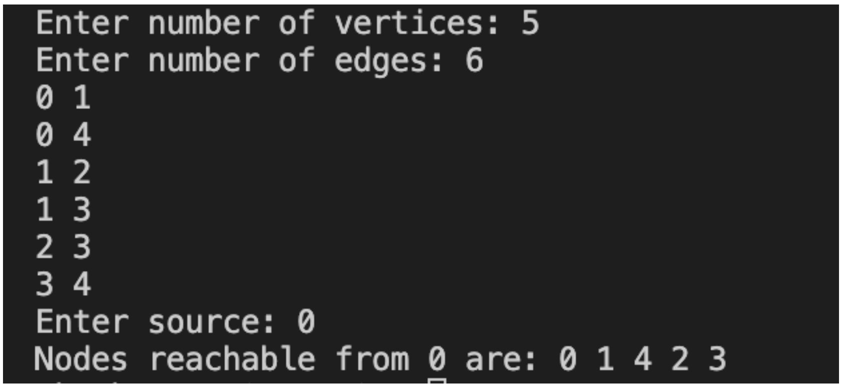
CODE :
(A)

```
#include <iostream>
#include <climits>
#include <math.h>
#include <string>
#include <cstring> //using strlen in char arrays
#include <set>
#include <algorithm>
#include <vector>
#include <fstream>
#include <list>
#include <stack>
#include <queue>
#include <unordered_map>
#include <map>
```

```
#include <set>
#include <cstdlib> // for absolute function

using namespace std;

#define ll long long



class graph
{
   int v;

   unordered_map<int, list<int> > l;
public:
   graph(int V)
   {
      v = V;
   }

   void insert_edge(int x, int y)
   {
      l[x].push_back(y);
      l[y].push_back(x);
   }

   void bfs(int source)
   {
      unordered_map<int, int> visited;
      queue <int> q;
      q.push(source);
      visited[source] = 1;

      while(!q.empty())
      {

         int front = q.front();
         cout << front << " ";
         // go to its neighbours
         for (auto x: l[front])
         {
            if (visited[x] == 0)
            {
               q.push(x);
               visited[x] = 1;
            }
         }

         q.pop();
      }
```

```
        cout << endl;
    }


};

int main()
{

    cout << "Enter number of vertices: ";
    int v;
    cin >> v;

    graph g(v);

    cout << "Enter number of edges: ";
    int e;
    cin >> e;

    for (int i = 0; i < e; i++)
    {
        int x, y;
        cin >> x >> y;
        g.insert_edge(x, y);
    }

    int source;
    cout << "Enter source: ";
    cin >> source;

    cout << "Nodes reachable from " << source << " are: ";
    g.bfs(source);

}
```

OUTPUT :



```
Enter number of vertices: 5
Enter number of edges: 6
0 1
0 4
1 2
1 3
2 3
3 4
Enter source: 0
Nodes reachable from 0 are: 0 1 4 2 3
```

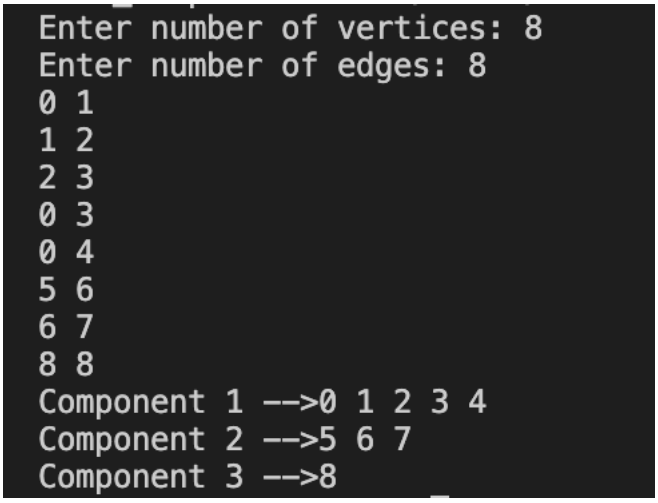
(B)

Code –

```
#include <iostream>
#include <climits>
#include <math.h>
#include <string>
#include <cstring> //using strlen in char arrays
#include <set>
#include <algorithm>
#include <vector>
#include <fstream>
#include <list>
#include <stack>
#include <queue>
#include <unordered_map>
#include <map>
#include <set>
#include <cstdlib> // for absolute function

using namespace std;

#define ll long long

// sample input
// Enter number of vertices: 8
// Enter number of edges: 8
// 0 1
// 1 2
// 2 3
// 0 3
// 0 4
// 5 6
// 6 7
// 8 8


class graph
{
   int v;
   map<int, list<int> > l;

public:

   graph(int V)
   {
      v = V;
   }
```

```
    void insert_edge(int x, int y)
    {
        l[x].push_back(y);
        l[y].push_back(x);

    }

    void dfs_helper(int source, unordered_map<int, int> & visited)
    {
        cout << source << " ";
        visited[source] = 1;

        for (auto nbr: l[source])
        {
            if (visited[nbr] == 0)
            {
                dfs_helper(nbr, visited);
            }
        }
    }

    void dfs()
    {
        unordered_map<int, int> visited;
        int count = 0;

        for (auto p: l)
        {
            if (visited[p.first] == 0)
            {
                count ++;
                cout << "Component " << count << " -->";
                dfs_helper(p.first, visited);
                cout << endl;
            }
        }
    }

};

int main()
{


    cout << "Enter number of vertices: ";
    int v;
    cin >> v;

    graph g(v);
    cout << "Enter number of edges: ";
    int e;
    cin >> e;

    for (int i = 0; i < e; i++)
    {
        int x, y;
        cin >> x >> y;
        g.insert_edge(x, y);
    }


    g.dfs();

}
```

OUTPUT :



```
Enter number of vertices: 8
Enter number of edges: 8
0 1
1 2
2 3
0 3
0 4
5 6
6 7
8 8
Component 1 -->0 1 2 3 4
Component 2 -->5 6 7
Component 3 -->8
```