ADSA – FINAL SEMESTER

1.MERGE AND INSERTION SORT

```
#include<iostream>
using namespace std;
void insertion(int *a,int n)
{
    for(int i = 1;i<n;i++)
    {
        int key = a[i];
        int j = i-1;
        while(j>-1 && key<a[j])
        {
            a[j+1] = a[j];
            j--;
        }
        a[j+1]=key;
    }
}
void merge(int *a,int l,int m,int r)
{
    int n1 = m-l+1;
    int n2 = r-m;
    int a1[n1],b1[n2];
    for(int i = 0;i<n1;i++)
      a1[i] = a[l+i];
    for(int i = 0;i<n2;i++)
      b1[i] = a[m+1+i];
    int i = 0,j = 0,k = l;
    while(i<n1 && j<n2)
    {
        if(a1[i]<=b1[j])
```

```
            {
                a[k] = a1[i];
                i++;
            }
            else
            {
                a[k] = b1[j];
                j++;
            }
            k++;
        }
        while(i<n1){
            a[k] = a1[i];
            i++;
            k++;
        }
        while(j<n2){
            a[k] = b1[j];
            j++;
            k++;
        }

    }
    void mergesort(int *a,int l,int r)
    {
        if(l>=r)
          return;
        int m = (l+r)/2;
        mergesort(a,l,m);
        mergesort(a,m+1,r);
        merge(a,l,m,r);
```

```
}
void print(int *a)
{
  for(int i = 0;i<5;i++)
   {
     cout<<a[i];
   }
}
int main()
{
   cout<<"Insertion sort: ";
   int a[5] = {5,4,3,2,1};
   insertion(a,5);
   print(a);
   cout<<endl;
   cout<<"Merge sort: ";
   int b[5] = {5,4,3,2,1};
   mergesort(b,0,5);
   print(a);


}
```

2.TRAVELLING SALES MAN PROBLEM

```
#include<iostream>
#include<vector>
using namespace std;
void tsp(vector<vector<int>> a,int start,int n)
{
   vector<int> other;
   for(int i = 0;i<n;i++)
   {
      if(i!=start)
      {
         other.push_back(i);
      }
   }
   int minpath = 99999;
```

```
  vector<int> mintour;
  do
  {
    int k = start;
    int current_path = 0;
    vector<int> tour;
    tour.push_back(start);
    for(int i = 0;i<other.size();i++)
    {
      current_path = current_path + a[k][other[i]];
      k = other[i];
      tour.push_back(k);
    }
    tour.push_back(start);
    current_path = current_path + a[k][start];
    if(current_path<minpath)
    {
      minpath = current_path;
      mintour = tour;
    }

  }while(next_permutation(other.begin(),other.end()));
  cout<<"min path is "<<minpath<<endl;
  for(int i = 0;i<n;i++)
  {
    cout<<mintour[i]<<"-->";
  }
}
int main()
{
  int n;
  cout<<"Enter the no of cities :";
  cin>>n;
  vector<vector<int>> a(n,vector<int>(n));
  for(int i = 0;i<n;i++)
  {
    for(int j = 0;j<n;j++)
    {
      if(i == j)
       a[i][j] = 0;
      else
       cin>>a[i][j];
    }
  }
  int start = 0;
  tsp(a,start,n);
}
```

3.FRACTIONAL KNAPSACK(GREEDY APPROACH)

```
#include<iostream>
using namespace std;
struct process
{
    int id;//item
    float w,pr,r;//weight,profit,pw ratio
};
void fractional_knapsack(struct process *p,int c,int n)
{
    float profit = 0;
    for(int i = 0;i<n;i++)
    {
        if(p[i].w<=c)
        {
            cout<<"\t"<<p[i].id<<"\t"<<p[i].pr<<"\t"<<p[i].w<<"\t"<<p[i].r<<endl;
            profit = profit + p[i].pr;
            c = c - p[i].w;

        }
        else
        {
             cout<<"\t"<<p[i].id<<"\t"<<p[i].pr*(c/p[i].w)<<"\t"<<p[i].w<<"\t"<<p[i].r<<endl;
             profit = profit + p[i].pr*(c/p[i].w);
             c = 0;
        }
    }
    cout<<"Total profit "<<profit;
}
int main()
{
    int n,capacity;
    cout<<"Enter no of items :";
    cin>>n;
    cout<<"enter capacity";
    cin>>capacity;
    struct process p[n];
    cout<<"Enter weight: ";
    for(int i= 0;i<n;i++) { p[i].id = i+1;cin>>p[i].w; }
    cout<<"Enter profit ";
    for(int i= 0;i<n;i++) { cin>>p[i].pr; }
    for(int i= 0;i<n;i++) { p[i].r = p[i].pr/p[i].w; }
    //sorting profit based on descending order
    for(int i = 0;i<n;i++)
    {
        for(int j = i+1;j<n;j++)
        {
            if(p[i].r<p[j].r)
            {
                struct process t = p[i];
                p[i] = p[j];
                p[j] = t;
            }
        }
    }
    fractional_knapsack(p,capacity,n);
}
```

4.JOB SEQUENCING PROBLEM

```
#include<iostream>
#include<vector>
using namespace std;
struct process
{
    int id,d;//item
    float pr;//weight,profit,pw ratio
};
void job_sequence(struct process *p,int nofslots,int n)
{
    vector<int> slots(nofslots,-1);
    for(int i = 0;i<n;i++)
    {
        for(int j = p[i].d-1;j>=0;j--)
        {
            if(slots[j] == -1)
            {
                slots[j] = p[i].id;
                break;
            }
        }
    }
    for(int i = 0;i<nofslots;i++)
    {
        cout<<endl<<"\t"<<i+1<<"\t"<<slots[i]<<"\t"<<p[slots[i]-1].pr;
    }
}
int main()
{
    int n,nofslots;
    cout<<"Enter no of jobs :";
    cin>>n;
    cout<<"Enter no of slots :";
    cin>>nofslots;
    struct process p[n];
    cout<<"Enter Deadlines: ";
    for(int i= 0;i<n;i++) { p[i].id = i+1;cin>>p[i].d; }
    cout<<"Enter profit ";
    for(int i= 0;i<n;i++) { cin>>p[i].pr; }
    //sorting profit based on descending order
    for(int i = 0;i<n;i++)
    {
        for(int j = i+1;j<n;j++)
        {
            if(p[i].pr<p[j].pr)
            {
                struct process t = p[i];
                p[i] = p[j];
                p[j] = t;
            }
        }
    }
    job_sequence(p,nofslots,n);
}
```

5.OPTIMAL BINARY SEARCH TREE

```
#include<iostream>
using namespace std;
void obst(int *p,int *q,int n)
{
    //weight matrix
    int w[n][n];
    for(int l = 1;l<=n;l++)
    {
        for(int i = 0;i<n-l+1;i++)
        {
            int j = i+l-1;
            if(i == j)
               w[i][j] = q[j];
            else
               w[i][j] = w[i][j-1] + p[j] + q[j];
        }
    }
    //cost matrix and r matrix
    int c[n][n],r[n][n];
    int minsum,mink;
     for(int l = 1;l<=n;l++)
    {
        for(int i = 0;i<n-l+1;i++)
        {
            int j = i+l-1;
            if(i == j)
            {
              c[i][j] = r[i][j] = 0;
            }
            else
            {
              minsum = 9999;
              mink   = -1;
              for(int k = i+1;k<=j;k++)
              {
                  int sum = c[i][k-1] + c[k][j] + w[i][j];
                  if(sum<minsum)
                  {
                      minsum = sum;
                      mink = k;
                  }
              }
              c[i][j] = minsum;
              r[i][j] = mink;
            }
        }
    }
    cout<<"weight matrix"<<endl;
    for(int i = 0;i<n;i++)
    {
        for(int j = 0;j<n;j++)
        {
            cout<<w[i][j]<<"   ";
        }
        cout<<"\n";
    }
    cout<<endl;
    cout<<"Cost matrix"<<endl;
    for(int i = 0;i<n;i++)
    {
        for(int j = 0;j<n;j++)
        {
            cout<<c[i][j]<<"   ";
        }
        cout<<"\n";
    }
    cout<<endl;
    cout<<"R matrix"<<endl;
    for(int i = 0;i<n;i++)
    {
        for(int j = 0;j<n;j++)
        {
            cout<<r[i][j]<<"   ";
        }
        cout<<"\n";
    }
```

```
}
int main()
{
   int n;
   cout<<"Enter the no of keys : ";
   cin>>n;
   int p[n],q[n];
   cout<<"Enter qj :";
   for(int i = 0;i<n;i++)
   {
      cin>>q[i];

   }
   cout<<"Enter pj :";
   for(int i = 1;i<n;i++){
      cin>>p[i];

   }
   obst(p,q,n);
}
```

6.ZERO ONE KNAPSACK – DYNAMIC APPROACH

```
#include<iostream>
using namespace std;
void zoknapsack(int *p,int *w,int n,int c)
{
   int m[n+1][c+1];
   for(int i = 0;i<=n;i++)
   {
      for(int j = 0;j<=c;j++)
      {
         if(i == 0 || j == 0){
            m[i][j]=0;
         }
         else
         {
            if(j<w[i-1])
            {
               m[i][j] = m[i-1][j];
            }
            else
            {
               m[i][j] = max(m[i-1][j],p[i-1]+m[i-1][j-w[i-1]]);
            }
         }
      }
   }
   for(int i = 0;i<=n;i++)
```

```
    {
        for(int j = 0;j<=c;j++)
        {
            cout<<m[i][j]<<"  ";
        }
        cout<<"\n";
    }

}
int main()
{
    int n,c;
   cout<<"Enter no of items : ";
   cin>>n;
   cout<<"Enter capacity ";
   cin>>c;
   int w[n],p[n];
   cout<<"Enter the weight :";
   for(int i = 0;i<n;i++){ cin>>w[i]; }
   cout<<"Enter the profit :";
   for(int i = 0;i<n;i++){ cin>>p[i]; }
   zoknapsack(p,w,n,c);
}
```

7.N QUEENS – BACKTRACKING

```
#include<iostream>
#include<vector>
using namespace std;
int s = 0;
bool issafe(vector<vector<int>> board,int r,int c,int n)
{
    int i,j;
    //column wise checking
    for(i = 0;i<r;i++)
    {
        if(board[i][c] == 1)
        {
          return false;
        }
    }
    // upper left diagonal
    for(i = r,j = c;i>=0&&j>=0;--i,--j)
    {
        if(board[i][j]==1)
        {
            return false;
        }
    }
    //upper right diagonal
    for(i = r,j = c;i>=0 && j<n;--i,++j)
    {
        if(board[i][j]==1)
        {
            return false;
        }
    }
    return true;
}
bool nqueensutil(vector<vector<int>> board,int r,int n)
{
    if(r == n)//solution completed
    {
        s++;
        cout<<"Solution "<<s<<"\n";
        for(int i = 0;i<n;i++)
        {
            for(int j = 0;j<n;j++)
            {
                cout<<board[i][j]<<" ";
            }
            cout<<"\n";
        }
        return true;
    }
    bool res = false;
    for(int c = 0;c<n;c++)
    {
        if(issafe(board,r,c,n))
        {
            board[r][c] = 1;
            res = nqueensutil(board,r+1,n) || res;
            board[r][c] = 0;//backtrack
        }
    }
    return res;
}
void nqueens(int n)
{
    vector<vector<int>> board(n,vector<int>(n,0));//create chess board and fill it with zero
    int r = 0;//initially try first row to fill,but programically it is zero
    if(!nqueensutil(board,r,n))
    {
        cout<<"Solution not found";
    }
}
int main()
{
    int n;
    cout<<"Enter no of Queens : ";
    cin>>n;
    nqueens(n);
}
```

8.SUM OF SUBSET PROBLEM

```
#include<iostream>
#include<vector>
using namespace std;
void findallsubsets(vector<int>& a,int target,int index,vector<int> subset)
{
   if(target == 0)
   {
      for(int num:subset)
      {
         cout<<num<<" ";
      }
      cout<<"\n";
      return;
   }
   if(target<0 || index<0){return;}
   subset.push_back(a[index]);
   findallsubsets(a,target-a[index],index-1,subset);
   subset.pop_back();
   findallsubsets(a,target,index-1,subset);
}
int main()
{
   int n,target;
   cout<<"Enter the no of elements : ";
   cin>>n;
   vector<int> a(n);
   vector<int> subset;
   for(int i = 0;i<n;i++)
   {
      cin>>a[i];
   }
   cout<<"Target :";
   cin>>target;
   findallsubsets(a,target,n-1,subset);
}
```

9.TOPOLOGICAL ORDER

```
#include<iostream>
#include<stack>
#include<vector>
using namespace std;
```

```
void dfs_visit(vector<vector<int>>& g,int v,vector<bool>& visited,vector<int>& parent,int&
ct,vector<int>& st,vector<int>& ft,stack<int>&s)
{
    ct++;
    st[v] = ct;
    visited[v] = true;
    for(int i : g[v])
    {
        if(!visited[i])
        {
            parent[i] = v;
            dfs_visit(g,i,visited,parent,ct,st,ft,s);
        }
    }
    ct++;
    ft[v] = ct;
    s.push(v);
}
void topological_order(vector<vector<int>>& g,int v,int e)
{
    vector<bool> visited(v,false);
    vector<int> parent(v,-1);
    vector<int> st(v);
    vector<int> ft(v);
    stack<int> s;
    int ct = 0;
    for(int i = 0;i<v;i++)
    {
      if(!visited[i])
      {
          dfs_visit(g,i,visited,parent,ct,st,ft,s);
      }
    }
    while(!s.empty())
    {
        cout<<s.top()<<"-->"<<endl;
        s.pop();
    }
    cout<<"vertices  parent  starting time   finishing time";
    cout<<endl;
    for(int i = 0;i<v;i++)
    {
        cout<<"\t"<<i+1<<"\t"<<parent[i]<<"\t"<<st[i]<<"\t"<<ft[i]<<endl;
    }
}
int main()
{
    int v,e;
    cout<<"Enter no of vertices and  edges";
    cin>>v>>e;
    vector<vector<int>> g(v);
    for(int i = 0;i<e;i++)
    {
        int u,v;
        cout<<"From/to :";
        cin>>u>>v;
        g[u].push_back(v);
    }
    topological_order(g,v,e);
}
```

10.KRUSHKALS ALGORITHM

```
#include<iostream>
#include<vector>
#include<algorithm>
#include<numeric>
using namespace std;
struct edge
{
    int u,v,w;
    edge(int a, int b,int c):u(a),v(b),w(c){}
};
void krushkals(vector<edge>& edges,int v)
{
    sort(edges.begin(),edges.end(),[](const edge& a,const edge &b){ return a.w<b.w;});
    vector<edge> mst;
    vector<int> parent(v);
    iota(parent.begin(),parent.end(),0);
    for(int i = 0;i<edges.size();i++)
    {
        int pu = edges[i].u;
        int pv = edges[i].v;
        while(parent[pu] != pu) pu = parent[pu];
        while(parent[pv] != pv) pv = parent[pv];
        if(pu!=pv)
        {
            parent[pv] = pu;
            mst.push_back(edges[i]);
        }
    }
    for(int i = 0;i<mst.size();i++)
    {
        cout<<"\t"<<mst[i].u<<"\t"<<mst[i].v<<"\t"<<mst[i].w<<endl;
    }
}
int main()
{
    int v,e;
```

```
  cout<<"No of vertices and Edges";
  cin>>v>>e;
  vector<edge> edges;
  for(int i = 0;i<e;i++)
  {
     int from,to,weight;
     cout<<"From/to/weight : ";
     cin>>from>>to>>weight;
     edges.push_back(edge(from,to,weight));
  }
  krushkals(edges,v);
}
```

11.BELL MAN FORD

```
#include<iostream>
#include<vector>
using namespace std;
struct edge
{
   int u,v,w;
};
void bellmanford(vector<edge>& edges,int s,int v)
{
   vector<int> parent(v,-1);
   vector<int> distance(v,9999);
   distance[s]=0;
   for(int i = 0;i<v-1;i++)
   {
      for(int j = 0;j<edges.size();j++)
      {
         int a = edges[j].u;
         int b = edges[j].v;
         int c = edges[j].w;
         if(distance[a]+c<distance[b])
         {
            parent[b] = a;
            distance[b] = distance[a]+c;
         }
      }
   }
       for(int j = 0;j<edges.size();j++)
      {
         int a = edges[j].u;
         int b = edges[j].v;
         int c = edges[j].w;
         if(distance[a]+c<distance[b])
         {
           cout<<"Negative cycle";
```

```
            return;
          }
        }
    cout<<"Vertex  "<<"Parent  "<<"Distance  ";
    for(int i = 0;i<v;i++)
    {
        cout<<i<<"\t"<<parent[i]<<"\t"<<distance[i]<<endl;
    }
}
int main()
{
    int v,e,s;
    cout<<"No of vertices and edges : ";
    cin>>v>>e;
    vector<edge> edges(e);
    for(int i = 0;i<e;i++)
    {
        cin>>edges[i].u>>edges[i].v>>edges[i].w;
    }
    cout<<"Enter source :";
    cin>>s;
    bellmanford(edges,s,v);
}
```

12.FLOYD WASHALL

```
#include<iostream>
#include<vector>
#include<limits>
using namespace std;
int INF = 9999;
void floydwashall(vector<vector<int>>& d,int v)
{
    vector<vector<int>> g(v,vector<int>(v,INF));
    vector<vector<int>> p(v,vector<int>(v,-1));
    for(int i = 0;i<v;i++)
    {
        for(int j = 0;j<v;j++)
        {
            if(i == j)
            {
                g[i][j] =0;
            }
            else if (d[i][j]!=0)
            {
                g[i][j] = d[i][j];
            }
        }
    }
    //parent initialization
```

```
  for(int i = 0;i<v;i++)
{
    for(int j = 0;j<v;j++)
    {
        if(i != j && g[i][j] != INF)
        {
            p[i][j] =i;
        }
    }
}
for(int k = 0;k<v;k++)
    {
      for(int i = 0;i<v;i++)
    {
          for(int j = 0;j<v;j++)
          {
            if(g[i][k]!=INF && g[k][j]!=INF && g[i][k]+g[k][j]<g[i][j])
            {
                g[i][j] = g[i][k]+g[k][j];
                p[i][j] = p[k][j];
            }
          }
    }
}
cout<<"Distance matrix"<<endl;
 for(int i = 0;i<v;i++)
    {
          for(int j = 0;j<v;j++)
          {
            if(g[i][j] == INF)
            {
               cout<<"INF\t";
            }
            else
            {
                cout<<g[i][j]<<"\t";
            }
          }
          cout<<endl;
    }
    cout<<"parent matrix"<<endl;
    for(int i = 0;i<v;i++)
    {
          for(int j = 0;j<v;j++)
          {

                cout<<p[i][j]<<"\t";
          }
          cout<<endl;
    }
```

```
}
int main()
{
   int v;
   cout<<"Enter the no of vertices : ";
   cin>>v;
   vector<vector<int>> graph(v,vector<int>(v,0));
   for(int i = 0;i<v;i++)
   {
      for(int j = 0;j<v;j++)
      {
        cin>>graph[i][j];
      }
   }
   floydwashall(graph,v);
}
```

13. 0 1 KNAPSACK – BRANCH AND BOUND

```
#include<iostream>
#include<vector>
#include<algorithm>
using namespace std;
struct item{
  int w,p;
};
bool compare(item& a,item& b)
{
   double A = (double)a.w/a.p;
   double B = (double)b.w/b.p;
   return A>B;
}
void branchandbound(vector<item>& items,int c)
{
   sort(items.begin(),items.end(),compare);
   int maxp = 0;
   int n = items.size();
   function<void(int,int,int)> explore = [&](int l,int cw,int cv)
   {
      //termination for recursive
     if(l==n || cw>c)
     {
        if(cv>maxp)
        {
           maxp = cv;
        }
        return;
     }
```

```
      //include this level
    if(cw+items[l].w<=c)
      explore(l+1,cw+items[l].w,cv+items[l].p);
      //exclude this level
    explore(l+1,cw,cv);
    };
    explore(0,0,0); //level,currentweight,currentvalue
    cout<<"max profit is"<<maxp;
}
int main()
{
  int n,c;
  cout<<"No of items : ";
  cin>>n;
  cout<<endl<<"Enter capacity : ";
  cin>>c;
  vector<item> items(n);
  cout<<endl<<"Weight : ";
  for(int i = 0;i<n;i++){ cin>>items[i].w;}
  cout<<endl<<"Profit : ";
  for(int i = 0;i<n;i++){ cin>>items[i].p;}
  branchandbound(items,c);
}
```

14. PRIMS ALGORITHM

```
#include <iostream>
#include <vector>
#include <queue>
#include <utility>
#include <limits>
using namespace std;

struct edge
{
    int to, weight;
    edge(int a, int b) : to(a), weight(b) {}
};

void prims(vector<vector<edge>> &g, int start, int n)
{
```

```
vector<bool> visited(n);
vector<int> parent(n, -1);
vector<int> cost(n, INT_MAX); // Initialize with infinity
priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>> pq;
cost[start] = 0;
pq.push({0, start});
while (!pq.empty())
{
   int s = pq.top().second;
   pq.pop();
   if (visited[s])
      continue;
   visited[s] = true;
   for (int i = 0; i < g[s].size(); i++)
   {
      int v = g[s][i].to;
      int w = g[s][i].weight;
      if (!visited[v] && w < cost[v])
      {
         parent[v] = s;
         cost[v] = w;
         pq.push({cost[v], v});
      }
   }
}
int sum = 0;
for (int i = 0; i < n; i++)
{
   sum = sum + cost[i];
}
cout << "MIN cost " << sum;
```

```
}

int main()
{
    int v, e;
    cout << "Enter no of vertices and  edges";
    cin >> v >> e;
    vector<vector<edge>> g(v);
    for (int i = 0; i < e; i++)
    {
        int u, v, w;
        cout << "From/to/weight :";
        cin >> u >> v >> w;
        g[u].push_back(edge(v, w));
    }
    prims(g, 0, v);
}
```