1. Basic

1.1 C++ Solution Template

```
#define _(x) {cout << #x << " = " << x << " ";}
ios::sync_with_stdio(false);
cin.tie(NULL)
cout.tie(NULL)
```

1.2.1 C++ String

read one line -> getline(cin, a);

scanf("%[^\n]s",a);

1.3.1 Permutation

Usage

bool next_permutation (BidirectionalIterator first, BidirectionalIterator last);

bool next_permutation (BidirectionalIterator first, BidirectionalIterator last, Compare comp);

1.3.3 Lower Bound

> Returns an iterator pointing to the first element in the range [first,last) which does not compare less than val.

Usage

ForwardIterator lower_bound (ForwardIterator first, ForwardIterator last, const T& val, Compare comp);

1.3.5 Heap

make_heap(v1.begin(), v1.end());

INS- v1.push_back(50); push_heap(v1.begin(), v1.end());

DEL: pop_heap(v1.begin(), v1.end()); v1.pop_back();

1.4 STL Containers

1.4.5 Queue

```
queue <int> gquiz;
gquiz.push(10);  gquiz.front();  gquiz.back();  gquiz.pop();
```

1.4.7 Stack

```
stack <int> s;
s.push(10);  s.top();//returns data.  s.pop();//pop doesn't return
```

1.4.8 Priority Queue

```
priority_queue <int> gquiz;
gquiz.push(10);  gquiz.size();  gquiz.top();  gquiz.pop();
```

Graph

DFS

Undirected

```
void addEdge(vector<int> adj[], int u, int v)
{
   adj[u].push_back(v);
   adj[v].push_back(u);
}
void DFSUtil(int u, vector<int> adj[],
                  vector<bool> &visited)
{
   visited[u] = true;
   cout << u << " ";
   for (int i=0; i<adj[u].size(); i++)
      if (visited[adj[u][i]] == false)
         DFSUtil(adj[u][i], adj, visited);
}
void DFS(vector<int> adj[], int V)
{
```

```
    vector<bool> visited(V, false);
    for (int u=0; u<V; u++)
        if (visited[u] == false)
            DFSUtil(u, adj, visited);
}
int main()
{
    int V = 5;
    vector<int> adj[V];
    addEdge(adj, 0, 1);
    DFS(adj, V);
    return 0;
```

Weighted Undirected

```
void addEdge(vector <pair<int, int> > adj[], int u,
                                                    int v, int wt)
{
        adj[u].push_back(make_pair(v, wt));
        adj[v].push_back(make_pair(u, wt));
}

// Print adjacency list representation og graph
void printGraph(vector<pair<int,int> > adj[], int V)
{
        int v, w;
        for (int u = 0; u < V; u++)
        {
                cout << "Node " << u << " makes an edge with \n";
                for (auto it = adj[u].begin(); it!=adj[u].end(); it++)
                {
                        v = it->first;
                        w = it->second;
                        cout << "\tNode " << v << " with edge weight ="
                                << w << "\n";
                }
                cout << "\n";
        }
}
int main()
{
        int V = 5;
        vector<pair<int, int> > adj[V];
        addEdge(adj, 0, 1, 10);
        printGraph(adj, V);
        return 0;
}
```

To Print all paths from u to v

```
class Graph
{
        int V;
        list<int> *adj;
        void printAllPathsUtil(int , int , bool [], int [], int &);

public:
        Graph(int V);
        void addEdge(int u, int v);
        void printAllPaths(int s, int d);
};

Graph::Graph(int V)
{
```

```
	this->V = V;
	adj = new list<int>[V];
}

void Graph::addEdge(int u, int v)
{
	adj[u].push_back(v); // Add v to u's list.
}
void Graph::printAllPaths(int s, int d)
{
	bool *visited = new bool[V];
	int *path = new int[V];
	int path_index = 0;
	for (int i = 0; i < V; i++)
		visited[i] = false;
	printAllPathsUtil(s, d, visited, path, path_index);
}
void Graph::printAllPathsUtil(int u, int d, bool visited[],
                                                        int path[], int &path_index)
{
	visited[u] = true;
	path[path_index] = u;
	path_index++;
	if (u == d)
	{
		for (int i = 0; i<path_index; i++)
			cout << path[i] << " ";
		cout << endl;

	}
	else	{
		list<int>::iterator i;
		for (i = adj[u].begin(); i != adj[u].end(); ++i)
			if (!visited[*i])
				printAllPathsUtil(*i, d, visited, path, path_index);
	}
	path_index--;
	visited[u] = false;
}
int main()
{
	// Create a graph given in the above diagram
	Graph g(4);
	g.addEdge(0, 1);
	int s = 2, d = 3;
	g.printAllPaths(s, d);
}
```

BFS (directed)

```
vector<bool> v;
vector<vector<int> > g;
void edge(int a, int b)
{
	g[a].pb(b);
}

void bfs(int u)
{
	queue<int> q;
	q.push(u);
	v[u] = true;
	while (!q.empty()) {
		int f = q.front();
		q.pop();
		cout << f << " ";
		for (auto i = g[f].begin(); i != g[f].end(); i++) {
```

```
                        if (!v[*i]) {
                                q.push(*i);
                                v[*i] = true;
                        }
                }
        }
}
int main()
{
        int n, e;
        cin >> n >> e;
        v.assign(n, false);
        g.assign(n, vector<int>());
        int a, b;
        for (int i = 0; i < e; i++) {
                cin >> a >> b;
                edge(a, b);
        }
        for (int i = 0; i < n; i++) {
                if (!v[i])
                        bfs(i);
        }
        return 0;
}
```

BELLMAN FORD

```
struct Edge
{
        int src, dest, weight;
};
struct Graph
{       int V, E;
        struct Edge* edge;
};
struct Graph* createGraph(int V, int E)
{
        struct Graph* graph = new Graph;
        graph->V = V;
        graph->E = E;
        graph->edge = new Edge[E];
        return graph;
}

void printArr(int dist[], int n)
{
        printf("Vertex Distance from Source\n");
        for (int i = 0; i < n; ++i)
                printf("%d \t\t %d\n", i, dist[i]);
}
void BellmanFord(struct Graph* graph, int src)
{
        int V = graph->V;
        int E = graph->E;
        int dist[V];
        for (int i = 0; i < V; i++)
                dist[i] = INT_MAX;
        dist[src] = 0;
        for (int i = 1; i <= V-1; i++)
        {
                for (int j = 0; j < E; j++)
                {
                        int u = graph->edge[j].src;
                        int v = graph->edge[j].dest;
                        int weight = graph->edge[j].weight;
                        if (dist[u] != INT_MAX && dist[u] + weight < dist[v])
                                dist[v] = dist[u] + weight;
                }
```

```
        }
        for (int i = 0; i < E; i++)
        {
                int u = graph->edge[i].src;
                int v = graph->edge[i].dest;
                int weight = graph->edge[i].weight;
                if (dist[u] != INT_MAX && dist[u] + weight < dist[v])
                        printf("Graph contains negative weight cycle");
        }
        printArr(dist, V);
        return;
}
int main()
{
        int V = 5; // Number of vertices in graph
        int E = 8; // Number of edges in graph
        struct Graph* graph = createGraph(V, E);
        graph->edge[0].src = 0;
        graph->edge[0].dest = 1;
        graph->edge[0].weight = -1;
        BellmanFord(graph, 0);
        return 0;
}
```

Iterative Function to calculate (x^y)%p in O(log y)

```
int power(int x, unsigned int y, int p)
{
    int res = 1;
    x = x % p;
    while (y > 0)
    {
        if (y & 1)
            res = (res*x) % p;
        y = y>>1;
        x = (x*x)%p;
    }
    return res;
}
```

Function for finding sum of larger numbers

```
string findSum(string str1, string str2)
{
    if (str1.length() > str2.length())
        swap(str1, str2);
    string str = "";
    int n1 = str1.length(), n2 = str2.length();
    reverse(str1.begin(), str1.end());
    reverse(str2.begin(), str2.end());
    int carry = 0;
    for (int i=0; i<n1; i++)
    {
        int sum = ((str1[i]-'0')+(str2[i]-'0')+carry);
        str.push_back(sum%10 + '0');
        carry = sum/10;
    }
    for (int i=n1; i<n2; i++)
    {
        int sum = ((str2[i]-'0')+carry);
        str.push_back(sum%10 + '0');
        carry = sum/10;
    }
    if (carry)
        str.push_back(carry+'0');
    reverse(str.begin(), str.end());
    return str;
}
```

PRIM'S

```
#include<bits/stdc++.h>
using namespace std;
# define INF 0x3f3f3f3f
typedef pair<int, int> iPair;
class Graph
{
	int V;
	list< pair<int, int> > *adj;

public:
	Graph(int V);
	void addEdge(int u, int v, int w);
	void primMST();
};
Graph::Graph(int V)
{
	this->V = V;
	adj = new list<iPair> [V];
}
void Graph::addEdge(int u, int v, int w)
{
	adj[u].push_back(make_pair(v, w));
	adj[v].push_back(make_pair(u, w));
}
// Prints shortest paths from src to all other vertices
void Graph::primMST()
{
	priority_queue< iPair, vector <iPair> , greater<iPair> > pq;
	int src = 0;
	vector<int> key(V, INF);
	vector<int> parent(V, -1);
	vector<bool> inMST(V, false);
	pq.push(make_pair(0, src));
	key[src] = 0;
	while (!pq.empty())
	{
		int u = pq.top().second;
		pq.pop();
		inMST[u] = true;
		list< pair<int, int> >::iterator i;
		for (i = adj[u].begin(); i != adj[u].end(); ++i)
		{
			int v = (*i).first;
			int weight = (*i).second;
			if (inMST[v] == false && key[v] > weight)
			{
				key[v] = weight;
				pq.push(make_pair(key[v], v));
				parent[v] = u;
			}
		}
	}
	for (int i = 1; i < V; ++i)
		printf("%d - %d\n", parent[i], i);
}
int main()
{
	int V = 9;
```

```
	Graph g(V);
	g.addEdge(0, 1, 4);
	g.primMST();
	return 0;
}
```

FLOYD-WARSHALL

```
void printSolution(int dist[][V]);
void floydWarshall (int graph[][V])
{
	int dist[V][V], i, j, k;
	for (i = 0; i < V; i++)
		for (j = 0; j < V; j++)
			dist[i][j] = graph[i][j];
	for (k = 0; k < V; k++)
	{
		for (i = 0; i < V; i++)
		{
			for (j = 0; j < V; j++)
			{
				if (dist[i][k] + dist[k][j] < dist[i][j])
					dist[i][j] = dist[i][k] + dist[k][j];
			}
		}
	}
	printSolution(dist);
}
int main()
{
	int graph[V][V] = { {0, 5, INF, 10},
						{INF, 0, 3, INF},
						{INF, INF, 0, 1},
						{INF, INF, INF, 0}
					};
	floydWarshall(graph);
	return 0;
}
```

TOPOLOGICAL SORT

```
class Graph
{
	int V;
	list<int> *adj;

	void topologicalSortUtil(int v, bool visited[], stack<int> &Stack);
public:
	Graph(int V);
	void addEdge(int v, int w);
	void topologicalSort();
};

Graph::Graph(int V)
{
	this->V = V;
	adj = new list<int>[V];
}

void Graph::addEdge(int v, int w)
{
	adj[v].push_back(w); // Add w to v's list.
}
```

```
void Graph::topologicalSortUtil(int v, bool visited[],
                                                        stack<int> &Stack)
{
        visited[v] = true;
        list<int>::iterator i;
        for (i = adj[v].begin(); i != adj[v].end(); ++i)
                if (!visited[*i])
                        topologicalSortUtil(*i, visited, Stack);
        Stack.push(v);
}
void Graph::topologicalSort()
{
        stack<int> Stack;
        bool *visited = new bool[V];
        for (int i = 0; i < V; i++)
                visited[i] = false;
        for (int i = 0; i < V; i++)
        if (visited[i] == false)
                topologicalSortUtil(i, visited, Stack);
        while (Stack.empty() == false)
        {
                cout << Stack.top() << " ";
                Stack.pop();
        }
}

int main()
{
        Graph g(6);
        g.addEdge(5, 2);
        g.topologicalSort();
        return 0;
}
```

DIJKSTRA

```
#include<iostream>
#include<conio.h>
#include<stdio.h>
using namespace std;
int shortest(int ,int);
int cost[10][10],dist[20],i,j,n,k,m,S[20],v,totcost,path[20],p;
int main()
{
        int c;
        cout <<"enter no of vertices";
        cin >> n;
        cout <<"enter no of edges";
        cin >>m;
        cout <<"\nenter\nEDGE Cost\n";
        for(k=1;k<=m;k++)
        {
                cin >> i >> j >>c;
                cost[i][j]=c;
        }
        for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
        if(cost[i][j]==0)
        cost[i][j]=31999;
        cout <<"enter initial vertex";
        cin >>v;
        cout << v<<"\n";
```

```
        shortest(v,n);
 }

int shortest(int v,int n)
{
        int min;
        for(i=1;i<=n;i++)
        {
                S[i]=0;
                dist[i]=cost[v][i];
        }
        path[++p]=v;
        S[v]=1;
        dist[v]=0;
        for(i=2;i<=n-1;i++)
        {
                k=-1;
                min=31999;
                for(j=1;j<=n;j++)
                {
                        if(dist[j]<min && S[j]!=1)
                        {
                        min=dist[j];
                        k=j;
                        }
                }
                if(cost[v][k]<=dist[k])
                p=1;
                path[++p]=k;
                for(j=1;j<=p;j++)
                cout<<path[j];
                cout <<"\n";
                //cout <<k;
                S[k]=1;
                for(j=1;j<=n;j++)
                if(cost[k][j]!=31999 && dist[j]>=dist[k]+cost[k][j] && S[j]!=1)
                        dist[j]=dist[k]+cost[k][j];
        }
}
```

GCD

```
int gcd(int a, int b)
{
   if (a == 0)
      return b;
   return gcd(b % a, a);
}
```

0-1 Knapsack

```
int main()
{
        int n, value[100], capacity, weight[100],i,j,K[100][100];
        int res,w;
        printf("Input number of  items:");
        scanf("%d",&n);
        printf("Enter values:");
        for(i=0;i<n;i++)
                scanf("%d",&value[i]);
        printf("Enter weights:");
        for(i=0;i<n;i++)
                scanf("%d",&weight[i]);
        printf("Enter Capacity:");
```

```
	scanf("%d",&capacity);
	for(i=0;i<=n;i++)
		for(j=0;j<=capacity;j++)
		{
			if(i==0 || j==0)
				K[i][j]=0;
			else if (weight[i-1]<=j)
				K[i][j]=max(K[i-1][j],K[i-1][j-weight[i-1]]+value[i-1]);
			else
				K[i][j]=K[i-1][j];
		}
	printf("The maximum value possible is:%d\n",K[n][capacity]);
	res=K[n][capacity];
	w=capacity;
	for(i=n;i>0 && res>0 ;i--)
	{
		if(K[i-1][w]==res)
			continue;
		else
		{
			printf("%d ",weight[i-1]);
			res=res-value[i-1];
			w=w-weight[i-1];

		}
	}
	return 0;
}
```

ASSEMBLY LINE SCHEDULING

```
int main()
{
	int n,i;
	int s[2][100],t[2][100],e[2],x[2];
	int T1[100],T2[100];
	printf("Input number of stations:");
	scanf(" %d",&n);
	printf("Input line 1: ");
	for(i=1;i<=n;i++)
		scanf(" %d",&s[0][i]);
	printf("Input line 2: ");
	for(i=1;i<=n;i++)
		scanf(" %d",&s[1][i]);
	printf("Input starttimes: ");
	for(i=0;i<=1;i++)
		scanf(" %d",&e[i]);
	printf("Input endtimes: ");
	for(i=0;i<=1;i++)
		scanf(" %d",&x[i]);
	printf("Input time for transferring for line 1 to line 2:");
	for(i=1;i<=n-1;i++)
		scanf(" %d",&t[0][i]);
	printf("Input time for transferring for line 2 to line 1:");
	for(i=1;i<=n-1;i++)
		scanf(" %d",&t[1][i]);
	T1[1]=e[0]+s[0][1];
	T2[1]=e[1]+s[1][1];
	for(i=2;i<=n;i++)
	{
		T1[i]=min(T1[i-1]+s[0][i],T2[i-1]+s[0][i]+t[1][i-1]);
		T2[i]=min(T2[i-1]+s[1][i],T1[i-1]+s[1][i]+t[0][i-1]);
```

```
	}
	printf("Minimum time: %d",min(T1[n]+x[0],T2[n]+x[1]));
	return 0;
}
```

CATALAN NUMBERS

```
void main()
{
	int n,i;
	double cat=1;
	printf("Input n:");
	scanf("%d",&n);
	for(i=1;i<=n;i++)
	{
		cat=(cat*(double)(i+n))/i;
	}
	cat=cat/(n+1);
	printf("%dth Catalan Number is:%.0f",n,cat);
}
```

COIN CHANGE

```
void main()
{
	int S[100];
	int a[100][100];
	int n,x,y,m,i,j;
	printf("Input value of n:");
	scanf("%d",&n);
	printf("Input value of m:");
	scanf("%d",&m);
	printf("Input values of available coins:");
	for(i=1;i<=m;i++)
		scanf(" %d",&S[i]);
	for(i=1;i<n;i++)
		a[0][i]=1;
	for(i=1;i<=n+1;i++)
	{
		if(i%S[1]==0)
			a[i][1]=1;
		else
			a[i][1]=0;
	}
	for(j=2;j<=n;j++)
		for(i=1;i<=(n+1);i++)
		{
			x=(j>1)?a[i][j-1]:0;
			y=((i-S[j])>=0)?a[i-S[j]][j]:0;
			a[i][j]=x+y;
			//printf("(%d,%d)=(%d,%d)+(%d,%d)--%d--%d--%d\n",i,j,i,j-1,i-S[j],j,x,y,a[i][j]);
		}
	/*for(i=1;i<=(n+1);i++)
	{
		for(j=1;j<=n;j++)
			printf("%d ",a[i][j]);
		printf("\n");
	}*/
	printf("Number of ways to make change:%d",a[n][m]);
}
```

CUTTING THE ROD

```
int main()
{
	int price[100],DP[100];
```

```
	int n,i,j,max;
	printf("Input length: ");
	scanf("%d",&n);
	printf("Input prices: ");
	for(i=1;i<=n;i++)
		scanf("%d",&price[i]);
	DP[1]=price[1];
	for(i=2;i<=n;i++)
	{
		max=DP[1]+DP[i-1];
		for(j=2;j<=(i/2);j++)
			if(DP[j]+DP[i-j]>max)
				max=DP[j]+DP[i-j];
		if(max>price[i])
			DP[i]=max;
		else
			DP[i]=price[i];
	}
	printf("Maximum obtainable value: %d",DP[n]);


	return 0;
}
```

EGG DROPPING PUZZLE

```
int main()
{
	int DP[100][100];
	int n,f,i,j,min,k;
	printf("Input number of floors: ");
	scanf("%d",&n);
	printf("Input number of eggs: ");
	scanf("%d",&f);
	for(i=1;i<=f;i++)
	{
		DP[i][0]=0;
		DP[i][1]=1;
	}
	for(i=1;i<=n;i++)
		DP[1][i]=i;
	for(i=2;i<=f;i++)
	{
		for(j=2;j<=n;j++)
		{
			min=1000;
			for(k=1;k<=j;k++)
			{
				if(min>max(DP[i][j-k],DP[i-1][k-1]))
					min=max(DP[i][j-k],DP[i-1][k-1]);
			}
			DP[i][j]=min+1;
		}
	}
	printf("The minimum number of trials required is: %d",DP[f][n]);
}
```

FIBONACCI //nth Fibonacci number

```
void multiply(int M[2][2], int a[2][2])
{
	int w=M[0][0]*a[0][0]+M[1][0]*a[0][1];
	int x=M[0][0]*a[0][1]+M[1][0]*a[1][1];
	int y=M[1][0]*a[0][0]+M[1][1]*a[0][1];
```

```
        int z=M[1][0]*a[0][1]+M[1][1]*a[1][1];

        M[0][0]=w;
        M[0][1]=x;
        M[1][0]=y;
        M[1][1]=z;
}

void main()
{
        int i,n,M[2][2],a[2][2];
        M[0][0]=M[0][1]=M[1][0]=1;
        M[1][1]=0;
        a[0][0]=a[1][0]=a[0][1]=1;
        a[1][1]=0;
        printf("Input n:");
        scanf("%d",&n);
        for(i=1;i<n-1;i++)
        {
                multiply(M,a);
        }
        printf("%dth Fibonacci number is:%d",n,M[0][0]);
}
```

GOLD MINE PROBLEM //Gold_Mine_Problem

```
void main()
{
        int mat[100][100],a[100][100];
        int n,i,j,max_value=0,right,right_up,right_down;
        printf("Input value of n:");
        scanf("%d",&n);
        printf("Input Matrix:");
        for(i=1;i<=n;i++)
                for(j=1;j<=n;j++)
                {
                        scanf(" %d",&mat[i][j]);
                }
        for(i=1;i<=n;i++)
        {
                for(j=1;j<=n;j++)
                        printf(" %d",mat[i][j]);
                printf("\n");
        }
        for(i=1;i<=n;i++)
                a[i][n]=mat[i][n];
        for(j=n-1;j>0;j--)
                for(i=n;i>0;i--)
                {
                        right=a[i][j+1];
                        right_up=((i-1)>0)?a[i-1][j+1]:0;
                        right_down=((i+1)<=n)?a[i+1][j+1]:0;
                        a[i][j]=mat[i][j]+max(right,max(right_up,right_down));
                }
        for(i=1;i<=n;i++)
        {
                max_value=max(max_value,a[i][1]);
        }
        printf("Answer:%d",max_value);
}
```

LARGEST DIVISIBLE PAIR SUBSET

//Length of largest subset such that every pair in the subset is such that the larger element of the pair is divisible by smaller element.

```
 void main()
 {
        int i,j,n,max,a[100],DP[100];
        printf("Input number of elements in array:");
        scanf("%d",&n);
        printf("Input elements:");
        for(i=0;i<n;i++)
                scanf("%d",&a[i]);
        merge_sort(a,0,n-1);
        DP[n-1]=1;
        for(i=n-2;i>=0;i--)
        {
                max=0;
                for(j=i+1;j<n;j++)
                {
                        if(a[j]%a[i]==0)
                        {
                                if(max<DP[j])
                                        max=DP[j];
                        }
                }
                DP[i]=max+1;
        }
        max=DP[0];
        for(i=1;i<n;i++)
                if(DP[i]>max)
                        max=DP[i];
        printf("Length of largest subset is:%d",max);

 }
```

LARGEST COMMON SUBSEQUENCE

```
int main()
{
        char input1[100],input2[100],output[100];
        int DP[100][100];
        int i,j;
        printf("Input string 1:");
        scanf(" %s",input1);
        printf("Input string 2:");
        scanf(" %s",input2);
        int flag=0;
        for(i=0;i<=strlen(input1);i++)
                for(j=0;j<=strlen(input2);j++)
                {
                        if(i==0 || j==0)
                                DP[i][j]=0;
                        else if(input1[i-1]==input2[j-1])
                                DP[i][j]=DP[i-1][j-1]+1;
                        else
                                DP[i][j]=max(DP[i][j-1],DP[i-1][j]);
                }
        int temp=DP[strlen(input1)][strlen(input2)];
        i=strlen(input1);
        j=strlen(input2);
        int k=0;
        while(i>0 && j>0)
        {
                if(input1[i-1]==input2[j-1])
```

```
                {
                        output[k++]=input1[i-1];
                        i--;
                        j--;
                }
                else if(DP[i][j-1]>DP[i-1][j])
                        j--;
                else
                        i--;
        }
        printf("\nLength of LCS is %d",DP[strlen(input1)][strlen(input2)]);
        printf("\nLCS: ");
        for(i=k-1;i>=0;i--)
                printf("%c",output[i]);
        return 0;
}

//To print the LCS, traverse the 2D array starting from DP[strlen(input1)][strlen(input2)]. Do following for every cell L[i][j]
// 1. If characters (in input1 and input2) corresponding to DP[i][j] are same (Or input1[i-1] == input2[j-1]), then include this character as part of LCS.
// 2. Else compare values of DP[i-1][j] and DP[i][j-1] and go in direction of greater value
```

LCS-3-STRINGS

```
int main()
{
        char input1[100],input2[100],input3[100];
        int DP[100][100][100];
        int i,j,k;
        printf("Input string 1:");
        scanf(" %s",input1);
        printf("Input string 2:");
        scanf(" %s",input2);
        printf("Input string 3:");
        scanf(" %s",input3);
        for(i=0;i<=strlen(input1);i++)
                for(j=0;j<=strlen(input2);j++)
                        for(k=0;k<=strlen(input3);k++)
                        {
                                if(i==0 || j==0 || k==0)
                                        DP[i][j][k]=0;
                                else if(input1[i-1]==input2[j-1] && input2[j-1]==input3[k-1])
                                        DP[i][j][k]=DP[i-1][j-1][k-1]+1;
                                else
                                        DP[i][j][k]=max(DP[i][j][k-1],max(DP[i][j-1][k],DP[i-1][j][k]));
                        }
        printf("\nLength of LCS is %d",DP[strlen(input1)][strlen(input2)][strlen(input3)]);
        return 0;
}
```

LONGEST INCREASING SUBSEQUENCE

```
typedef struct _set
{
        int array[100];
        int size;
} set;

int main()
{
        int a[100];
        set DP[100];
        int i,n,j,max,index;
        printf("Input size:");
        scanf(" %d",&n);
        printf("Input numbers:");
```

```
	for(i=1;i<=n;i++)
		scanf("%d",&a[i]);
	DP[1].size=1;
	DP[1].array[1]=a[1];
	for(i=2;i<=n;i++)
	{
		max=0;
		for(j=1;j<i;j++)
			if(a[i]>a[j] && DP[j].size>max)
			{
				index=j;
				max=DP[j].size;
			}
		for(j=1;j<=DP[index].size;j++)
			DP[i].array[j]=DP[index].array[j];
		DP[i].array[j]=a[i];
		DP[i].size=j;
	}
	max=DP[1].size;
	index=1;
	for(i=2;i<=n;i++)
	{
		if(DP[i].size>max)
		{
			max=DP[i].size;
			index=i;
		}
	}
	printf("\nLength of LIS is %d",DP[index].size);
	printf("\nLIS: ");
	for(i=1;i<=DP[index].size;i++)
		printf("%d ",DP[index].array[i]);
	return 0;
}
```

LONGEST REPEATED SUBSEQUENCE

```
int main()
{
	char input[100],output[100];
	int DP[100][100];
	int i,j;
	printf("Input string: ");
	scanf(" %s",input);
	int flag=0;
	for(i=0;i<=strlen(input);i++)
		for(j=0;j<=strlen(input);j++)
		{
			if(i==0 || j==0)
				DP[i][j]=0;
			else if(input[i-1]==input[j-1] && i!=j)
				DP[i][j]=DP[i-1][j-1]+1;
			else
				DP[i][j]=max(DP[i][j-1],DP[i-1][j]);
		}

	int temp=DP[strlen(input)][strlen(input)];
	i=strlen(input);
	i=strlen(input);
	int k=0;
	while(i>0 && j>0)
	{
		if(input[i-1]==input[j-1] && i!=j)
		{
			output[k++]=input[i-1];
			i--;
			j--;
		}
```

```
		else if(DP[i][j-1]>DP[i-1][j])
			j--;
		else
			i--;
	}
	printf("\nLength of LRS is %d",DP[strlen(input)][strlen(input)]);
	printf("\nLRS: ");
	for(i=k-1;i>=0;i--)
		printf("%c",output[i]);
	return 0;
}
```

MATRIX CHAIN MULTIPLICATION

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

void print_brackets(int brackets[100][100],int i, int j)
{
	if(i==j)
		printf("A");
	else
	{
		printf("(");
		print_brackets(brackets,i,brackets[i][j]);
		print_brackets(brackets,brackets[i][j]+1,j);
		printf(")");
	}
}

int main()
{
	int a[100];
	int DP[100][100],brackets[100][100];
	int i,j,k,n,l,q;
	printf("Input number of matrices: ");
	scanf("%d",&n);
	printf("Input array of Dimensions (%d dimensions): ",n+1);
	for(i=0;i<=n;i++)
		scanf("%d",&a[i]);
	for(i=1;i<=n;i++)
		DP[i][i]=0;
	for(l=2;l<=n;l++)
		for(i=1;i<=n-l+1;i++)
		{
			j=l+i-1;
			DP[i][j]=100000;
			for(k=i;k<j;k++)
			{
				q=a[i-1]*a[k]*a[j]+DP[i][k]+DP[k+1][j];
				if(q<DP[i][j])
				{
					DP[i][j]=q;
					brackets[i][j]=k;
				}
			}
		}
	printf("\nMinimum number of operations required is: %d",DP[1][n]);
	printf("\nThe brackets must be printed in the order: ");
	print_brackets(brackets,1,n);
	return 0;
}
```

SUBSET SUM DIVISIBILITY //Subset Sum is divisible by n

```
int main()
{
	int a[100],n,m,table[100],temp[100],i,j,flag=0;
	printf("Input n:");
	scanf("%d",&n);
```

```
	printf("Input numbers into array:");
	for(i=0;i<n;i++)
		scanf("%d",&a[i]);
	printf("Input sum:");
	scanf("%d",&m);

	if(n>m)
	{
		printf("Divisible subset is present.");
		return 0;
	}
	for(i=0;i<m;i++)
		table[i]=0;
	for(i=0;i<n;i++)
	{
		for(j=0;j<m;j++)
			temp[j]=0;
		for(j=0;j<m;j++)
		{
			if (table[j]==1)
			{
				if(table[(j+a[i])%m]==0)
					temp[(j+a[i])%m]=1;
			}
		}
		for(j=0;j<m;j++)
			if(temp[j])
				table[j]=temp[j];
		table[a[i]%m]=1;
		if(table[0]==1)
		{
			printf("Divisible subset is present.");
			return 0;
		}
	}
	printf("Divisible subset is absent.");
	return 0;
}
```

TILE STACKING PROBLEM

```
int main()
{
	int m,n,K,i,j,k,TS[100][100];
	printf("Input height");
	scanf("%d",&n);
	printf("Input the maximum value of tile");
	scanf("%d",&m);
	printf("Input maximum occurrences of a tile");
	scanf("%d",&K);
	for(i=0;i<=n;i++)
		for(j=0;j<=m;j++)
		{
			if(i==0 || j==0)
				TS[i][j]=1;
			else
			{
				TS[i][j]=0;
				for(k=0;k<=K;k++)
					if(i>=k)
						TS[i][j]=TS[i][j]+TS[i-k][j-1];
			}
		}
	for(i=0;i<=n;i++)
	{
		printf("\n");
		for(j=0;j<=m;j++)
			printf(" %d",TS[i][j]);
```

```
        }
        printf("%d",TS[n][m]);
}
```

TILING WITH DOMINOES

```
int main()
{
        int n,i;
        int A[100],B[100];
        printf("Input n: ");
        scanf("%d",&n);
        for(i=0;i<=n;i++)
        {
                A[i]=0;
                B[i]=0;
        }
        A[2]=3;
        B[1]=1;
        for(i=3;i<=n;i++)
        {
                if(i%2==0)
                        A[i]=A[i-2]+2*B[i-1];
                else
                        B[i]=B[i-2]+A[i-1];
        }
        printf("Number of ways to tile the board: %d",A[n]);
        return 0;
}
```

UGLY NUMBERS

```
void main()
{
        int ugly[1000];
        int i,n,i2=0,i3=0,i5=0,next_ugly_2=2,next_ugly_3=3,next_ugly_5=5;
        printf("Input n");
        scanf("%d",&n);
        ugly[0]=1;
        for(i=1;i<n;i++)
        {
                ugly[i]=min(next_ugly_2,min(next_ugly_3,next_ugly_5));
                if(ugly[i]==next_ugly_2)
                        next_ugly_2=ugly[++i2]*2;
                if(ugly[i]==next_ugly_3)
                        next_ugly_3=ugly[++i3]*3;
                if(ugly[i]==next_ugly_5)
                        next_ugly_5=ugly[++i5]*5;
        }
        printf("%dth ugly number is:%d",n,ugly[n-1]);
}
```

UNBOUNDED KNAPSACK

```
int main()
{
        int n, value[100], capacity, weight[100],i,j,K[100][100];
        printf("Input number of  items:");
        scanf("%d",&n);
        printf("Enter values:");
        for(i=0;i<n;i++)
                scanf("%d",&value[i]);
        printf("Enter weights:");
        for(i=0;i<n;i++)
                scanf("%d",&weight[i]);
        printf("Enter Capacity:");
        scanf("%d",&capacity);
        for(i=0;i<=n;i++)
                for(j=0;j<=capacity;j++)
```

```
		{
			if(i==0 || j==0)
				K[i][j]=0;
			else if (weight[i-1]<=j)
				K[i][j]=max(K[i-1][j],K[i][j-weight[i-1]]+value[i-1]);
			else
				K[i][j]=K[i-1][j];
		}
	printf("%d",K[n][capacity]);
	return 0;
}
```

EDIT DISTANCE

```
int editDistDP(string str1, string str2, int m, int n)
{
	int dp[m+1][n+1];
	for (int i=0; i<=m; i++)
	{
		for (int j=0; j<=n; j++)
		{
			if (i==0)
				dp[i][j] = j;
			else if (j==0)
				dp[i][j] = i;
				dp[i][j] = dp[i-1][j-1];
			else
				dp[i][j] = 1 + min(dp[i][j-1],dp[i-1][j],dp[i-1][j-1]);
		}
	}

	return dp[m][n];
}
int main()
{
	string str1 = "sunday";
	string str2 = "saturday";
	cout << editDistDP(str1, str2, str1.length(), str2.length());
	return 0;
}
```

MODULAR INVERSE(if M is prime)

```
void modInverse(int a, int m)
{
	int g = gcd(a, m);
	if (g != 1)
		cout << "Inverse doesn't exist";
	else
	{
		cout << "Modular multiplicative inverse is "
			<< power(a, m-2, m);
	}
}

// To compute x^y under modulo m
int power(int x, unsigned int y, unsigned int m)
{
	if (y == 0)
		return 1;
```

```
        int p = power(x, y/2, m) % m;
        p = (p * p) % m;

        return (y%2 == 0)? p : (x * p) % m;
}
```
ELSE EULER's Extended GCD
```
int gcdExtended(int a, int b, int *x, int *y);
void modInverse(int a, int m)
{
        int x, y;
        int g = gcdExtended(a, m, &x, &y);
        if (g != 1)
                cout << "Inverse doesn't exist";
        else
        {
                // m is added to handle negative x
                int res = (x%m + m) % m;
                cout << "Modular multiplicative inverse is " << res;
        }
}
int gcdExtended(int a, int b, int *x, int *y)
{
        // Base Case
        if (a == 0)
        {
                *x = 0, *y = 1;
                return b;
        }

        int x1, y1;
        int gcd = gcdExtended(b%a, a, &x1, &y1);
        *x = y1 - (b/a) * x1;
        *y = x1;
        return gcd;
}
```
SIEVE OF ERATOSTHENES
```
vector<long long >isprime(MAX_SIZE , true);
vector<long long >prime;
vector<long long >SPF(MAX_SIZE);
void seive(int N)
{
        isprime[0] = isprime[1] = false ;
        for (long long int i=2; i<N ; i++)
        {
                if (isprime[i])
                {
                        prime.push_back(i);
                        SPF[i] = i;
                }
                for (long long int j=0; j < (int)prime.size() && i*prime[j] < N && prime[j] <= SPF[i]; j++)
                {
                        isprime[i*prime[j]]=false;
                        SPF[i*prime[j]] = prime[j] ;
                }
        }
}
int main()
{
        int N = 13 ;
        manipulated_seive(N);
        for (int i=0; i<prime.size() && prime[i] <= N ; i++)
                cout << prime[i] << " ";
        return 0;
```

```
}
```
BINARY SEARCH
```
ll bin_search(ll i,ll j,vi &arr,ll ele)
{
    if(i>j)
    {
        return -1;
    }
    if(i==j)
    {
        if(arr[i]==ele)
        {
            return i;
        }
        return -1;
    }
    ll m=(i+j)/2;
    if(arr[m]<ele)
    {
        return bin_search(m+1,j,arr,ele);
    }
    if(arr[m]>ele)
    {
        return bin_search(i,m-1,arr,ele);
    }
    else
    {
        ll a =bin_search(m+1,j,arr,ele);
        if(a==-1)
        {
            return m;
        }
        return a;
    }
}
ll bin_search2(ll i,ll j,vi &arr,ll ele)//left most occurence
{
    if(i>j)
    {
        return -1;
    }
    if(i==j)
    {
        if(arr[i]==ele)
        {
            return i;
        }
        return -1;
    }
    ll m=(i+j)/2;
    if(arr[m]<ele)
    {
        return bin_search2(m+1,j,arr,ele);
    }
    if(arr[m]>ele)
    {
        return bin_search2(i,m-1,arr,ele);
    }
    else
    {
        ll a =bin_search2(i,m-1,arr,ele);
        if(a==-1)
        {
            return m;
        }
        return a;
```

```
   }
}
```
KMP STRING MATCHING
```
void computeLPSArray(char* pat, int M, int* lps);
void KMPSearch(char* pat, char* txt)
{
	int M = strlen(pat);
	int N = strlen(txt);
	int lps[M];
	computeLPSArray(pat, M, lps);
	int i = 0;
	int j = 0;
	while (i < N) {
		if (pat[j] == txt[i]) {
			j++;
			i++;
		}
		if (j == M) {
			printf("Found pattern at index %d ", i - j);
			j = lps[j - 1];
		}
		else if (i < N && pat[j] != txt[i]) {
			if (j != 0)
				j = lps[j - 1];
			else
				i = i + 1;
		}
	}
}
void computeLPSArray(char* pat, int M, int* lps)
{
	int len = 0;
	lps[0] = 0;
	int i = 1;
	while (i < M) {
		if (pat[i] == pat[len]) {
			len++;
			lps[i] = len;
			i++;
		}
		else
		{
			if (len != 0) {
				len = lps[len - 1];
			}
			else
			{
				lps[i] = 0;
				i++;
			}
		}
	}
}
```
BINOMIAL nCr
```
int binomialCoeff(int n, int k)
{
	int C[n+1][k+1];
	int i, j;
	for (i = 0; i <= n; i++)
	{
		for (j = 0; j <= min(i, k); j++)
		{
			if (j == 0 || j == i)
				C[i][j] = 1;
			else
				C[i][j] = C[i-1][j-1] + C[i-1][j];
```

```
                }
        }
        return C[n][k];
}
```

SEGMENT TREES (1-INDEXED TREE & ARRAY)

```
void build(int node, int start, int end)
{
    if(start == end)
    {
        // Leaf node will have a single element
        tree[node] = A[start];
    }
    else
    {
        int mid = (start + end) / 2;
        // Recurse on the left child
        build(2*node, start, mid);
        // Recurse on the right child
        build(2*node+1, mid+1, end);
        // Internal node will have the sum of both of its children
        tree[node] = tree[2*node] + tree[2*node+1];
    }
}
void update(int node, int start, int end, int idx, int val)
{
    if(start == end)
    {
        // Leaf node
        A[idx] += val;
        tree[node] += val;
    }
    else
    {
        int mid = (start + end) / 2;
        if(start <= idx and idx <= mid)
        {
            // If idx is in the left child, recurse on the left child
            update(2*node, start, mid, idx, val);
        }
        else
        {
            // if idx is in the right child, recurse on the right child
            update(2*node+1, mid+1, end, idx, val);
        }
        // Internal node will have the sum of both of its children
        tree[node] = tree[2*node] + tree[2*node+1];
    }
}
int query(int node, int start, int end, int l, int r)
{
    if(r < start or end < l)
    {
        // range represented by a node is completely outside the given range
        return 0;
    }
    if(l <= start and end <= r)
    {
        // range represented by a node is completely inside the given range
        return tree[node];
    }
    // range represented by a node is partially inside and partially outside the given
range
    int mid = (start + end) / 2;
    int p1 = query(2*node, start, mid, l, r);
    int p2 = query(2*node+1, mid+1, end, l, r);
```

```
    return (p1 + p2);
}
//LAZY PROPAGATION //vi lazy(size_of_tree,0);(initialize)
void updateRange(int node, int start, int end, int l, int r, int val)
{
    if(lazy[node] != 0)
    {
        // This node needs to be updated
        tree[node] += (end - start + 1) * lazy[node]; // Update it
        if(start != end)
        {
            lazy[node*2] += lazy[node];   // Mark child as lazy
            lazy[node*2+1] += lazy[node]; // Mark child as lazy
        }
        lazy[node] = 0;             // Reset it
    }
    if(start > end or start > r or end < l) // Current segment is not within range [l, r]
        return;
    if(start >= l and end <= r)
    {// Segment is fully within range
            tree[node] += (end - start + 1) * val;
        if(start != end) // Not leaf node
        {
            lazy[node*2] += val;
            lazy[node*2+1] += val;
        }
        return;
    }
    int mid = (start + end) / 2;
 updateRange(node*2, start, mid, l, r, val);  // Updating left child
 updateRange(node*2 + 1, mid + 1, end, l, r, val);// Updating right child
 tree[node] = tree[node*2] + tree[node*2+1];//Updating root with max valu
}

int queryRange(int node, int start, int end, int l, int r)
{
    if(start > end or start > r or end < l)
        return 0;          // Out of range
    if(lazy[node] != 0)
    {// This node needs to be updated
            tree[node] += (end - start + 1)*lazy[node]; // Update it
        if(start != end)
        {
            lazy[node*2] += lazy[node];  // Mark child as lazy
            lazy[node*2+1] += lazy[node];// Mark child as lazy
        }
        lazy[node] = 0;                  // Reset it
    }
    if(start >= l and end <= r)// Current segment is totally within range [l, r]
        return tree[node];
    int mid = (start + end) / 2;
    int p1 = queryRange(node*2, start, mid, l, r);// Query left child
    int p2 = queryRange(node*2 + 1, mid + 1, end, l, r);// Query right child
    return (p1 + p2);
}
```