Question Bank for AOA Practical Exam

1. Selection Sort
2. Insertion Sort
3. Merge Sort
4. Quick Sort
5. Longest Common Subsequence (LCS)
6. Single Source Shortest Path (Bellman Ford)
7. 0/1 Knapsack
8. Fractional Knapsack
9. MST – Prims
10. MST – Kruskal's
11. N queen
12. Sum of subset
13. All pair shortest path
14. Rabin Karp
15. KMP

1.Selection

```c
#include<stdio.h>
void display(int arr[],int n){
for (int i=0;i<n;i++){
        printf("%d\t",arr[i]); //1 | 7 2 4 9
    }
}
void selectsort(int arr[10],int n){
    int indexofmin,temp;
for(int i=0;i<n-1;i++){
    indexofmin=i;
    for(int j=i+1;j<n;j++){
        if(arr[j]<arr[indexofmin]){
            indexofmin=j;
        }
    }
    temp=arr[indexofmin];
    arr[indexofmin]=arr[i];
    arr[i]=temp;
    display(arr,n);
    printf("\n");
}
}

void main(){
    int n,arr[10];
    printf("Enter no of elements:");
    scanf("%d",&n);
    printf("Enter elements");
    for (int i=0;i<n;i++){
        scanf("%d",&arr[i]);
    }
    selectsort(arr,n);
    display(arr,n);

}
```

2.Insertion Sort

```c
#include <stdio.h>
void display(int arr[],int n){
for (int i=0;i<n;i++){
        printf("%d\t",arr[i]);
    }
}
void insertsort(int arr[10],int n){
int i,j,key;

for(int i=1;i<n;i++){
    key=arr[i];
    j=i-1;
    while(j>=0 && arr[j]>key){
        arr[j+1]=arr[j];
        j--;
    }
    arr[j+1]=key;
    display(arr,n);
    printf("\n");
}
}

void main(){
    int n,arr[10];
    printf("Enter no of elements:");
    scanf("%d",&n);
    printf("Enter elements");
    for (int i=0;i<n;i++){
        scanf("%d",&arr[i]);
    }
    insertsort(arr,n);
    display(arr,n);
}
```

3. Merge Sort

```
#include<stdio.h>
#include<conio.h>
#define MAX 10
// merge sort algorithm
void merge(int x[],int lb1,int mid,int ub2)
{
 int i,j,k,temp[MAX];
 i=lb1;
 j=mid+1;
 k=0;
 while(i<=mid && j<=ub2)
 {
 if(x[i]<x[j])
 temp[k++]=x[i++];
 else
 temp[k++]=x[j++];
 }
 while(i<=mid)
 temp[k++]=x[i++];
 while(j<=ub2)
 temp[k++]=x[j++];
 for(i=lb1,k=0;i<=ub2;i++,k++)
 x[i]=temp[k];
}
void mergeSort(int x[],int lb,int ub)
{
 int mid;
 if(lb<ub)
 {
 mid=(lb+ub)/2;
 mergeSort(x,lb,mid);
 mergeSort(x,mid+1,ub);
 merge(x,lb,mid,ub);
 }
}
void main()
{
 int x[MAX],n,i;
 printf("\nEnter the number of elements you want to insert:");
 scanf("%d",&n);
 for(i=0;i<n;i++)
 {
 printf("\nPlease enter any number:");
 scanf("%d",&x[i]);
 }
 printf("\nUnsorted array:\n");
 for(i=0;i<n;i++)
```

```
 printf("%d ",x[i]);
 mergeSort(x,0,n-1);
 printf("\nSorted array:\n");
 for(i=0;i<n;i++)
 printf("%d ",x[i]);
 getch();
}
```

4. Quick Sort

```
#include<stdio.h>
#include<conio.h>
#define MAX 10
//quick sort algorithm
int partition(int x[],int lb,int ub)
{
 int pivot,down,up,temp;
 pivot=x[lb];
 down=lb;
 up=ub;
 while(down<up)
 {
 while(x[down]<=pivot && down<ub)
 down++;
 while(x[up]>pivot)
 up--;
 if(down<up)
 {
 temp=x[down];
 x[down]=x[up];
 x[up]=temp;
 }
 }
 x[lb]=x[up];
 x[up]=pivot;
 return up;
}
void quick(int x[],int lb,int ub)
{
 int index;
 if(lb>=ub)
 return;
 index=partition(x,lb,ub);
 quick(x,lb,index-1); //sort left part
 quick(x,index+1,ub); //sort right part
}
void main()
```

```
{
 int x[MAX],n,i;
 printf("\nEnter the number of elements you want to insert:");
 scanf("%d",&n);
 for(i=0;i<n;i++)
 {
 printf("\nPlease enter any number:");
 scanf("%d",&x[i]);
 }
 printf("\nUnsorted array:\n");
 for(i=0;i<n;i++)
 printf("%d ",x[i]);
 quick(x,0,n-1);
 printf("\nSorted array:\n");
 for(i=0;i<n;i++)
 printf("%d ",x[i]);
 getch();
}
```

5.LCS

```
#include <stdio.h>
#include <string.h>
int i, j, m, n, LCS_table[20][20];
char S1[20], S2[20], b[20][20];
void lcsAlgo (char S1[20], char S2[20])
{
  m = strlen (S1);
  n = strlen (S2);
  int k;

  for (i = 0; i <= m; i++)
    LCS_table[i][0] = 0;
  for (i = 0; i <= n; i++)
    LCS_table[0][i] = 0;

  for (i = 1; i <= m; i++)
    for (j = 1; j <= n; j++)
      {
  if (S1[i - 1] == S2[j - 1])
    {
      LCS_table[i][j] = LCS_table[i - 1][j - 1] + 1;
    }
  else if (LCS_table[i - 1][j] >= LCS_table[i][j - 1])
    {
      LCS_table[i][j] = LCS_table[i - 1][j];
    }
  else
```

```
    {
      LCS_table[i][j] = LCS_table[i][j - 1];
    }
      }
  int index = LCS_table[m][n];
  char lcsAlgo[index + 1];
  lcsAlgo[index] = '\0';
  int i = m, j = n;
  while (i > 0 && j > 0)
    {
      if (S1[i - 1] == S2[j - 1])
  {
    lcsAlgo[index - 1] = S1[i - 1];
    i--;
    j--;
    index--;
  }

      else if (LCS_table[i - 1][j] > LCS_table[i][j - 1])
  i--;

      else
  j--;
    }
  k = strlen (lcsAlgo);

  printf ("S1 : %s \nS2 : %s \n", S1, S2);
  printf ("LCS: %s", lcsAlgo);
  printf ("\nThe length of LCS is %d.", k);
  printf ("\n");
  for (int i = 0; i <= m; i++)
    {
      for (int j = 0; j <= n; j++)
  {
    printf ("%d ", LCS_table[i][j]);
  }
      printf ("\n");
    }
}

int main ()
{
  printf ("Enter string S1:\n");
  gets (S1);
  printf ("Enter string S2:\n");
  gets (S2);

  lcsAlgo (S1, S2);
```

```
  printf ("\n");
}
```

6. Bellman Ford

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

struct Edge
{
    // This structure is equal to an edge. Edge contains two end points. These
edges are directed edges so they
//contain source and destination and some weight. These 3 are elements in this
structure
    int source, destination, weight;
};

// a structure to represent a connected, directed and weighted graph
struct Graph
{
    int V, E;
// V is number of vertices and E is number of edges

    struct Edge* edge;
// This structure contain another structure which we already created edge.
};

struct Graph* createGraph(int V, int E)
{
    struct Graph* graph = (struct Graph*) malloc( sizeof(struct Graph));
//Allocating space to structure graph
    graph->V = V;   //assigning values to structure elements that taken form
user.
    graph->E = E;
    graph->edge = (struct Edge*) malloc( graph->E * sizeof( struct Edge ) );
//Creating "Edge" type structures inside "Graph" structure, the number of edge
type structures are equal to number of edges
    return graph;
}

void FinalSolution(int dist[], int n)
{
// This function prints the final solution
    printf("\nVertex\tDistance from Source Vertex\n");
    int i;
    for (i = 0; i < n; ++i){
```

```
printf("%d \t\t %d\n", i, dist[i]);
}
}

void BellmanFord(struct Graph* graph, int source)
{
    int V = graph->V;
    int E = graph->E;
    int StoreDistance[V];
    int i,j;
    // This is initial step that we know , we initialize all distance to
infinity except source.
// We assign source distance as 0(zero)
    for (i = 0; i < V; i++)
        StoreDistance[i] = INT_MAX;
    StoreDistance[source] = 0;
    //The shortest path of graph that contain V vertices, never contain "V-1"
edges. So we do here "V-1" relaxations
    for (i = 1; i <= V-1; i++)
    {
        for (j = 0; j < E; j++)
        {
            int u = graph->edge[j].source;
            int v = graph->edge[j].destination;
            int weight = graph->edge[j].weight;

            if (StoreDistance[u] + weight < StoreDistance[v])
                StoreDistance[v] = StoreDistance[u] + weight;
        }
    }
    // Actually upto now shortest path found. But BellmanFord checks for
negative edge cycle. In this step we check for that
    // shortest distances if graph doesn't contain negative weight cycle.

    // If we get a shorter path, then there is a negative edge cycle.
    for (i = 0; i < E; i++)
    {
        int u = graph->edge[i].source;
        int v = graph->edge[i].destination;
        int weight = graph->edge[i].weight;
        if (StoreDistance[u] + weight < StoreDistance[v])
            printf("This graph contains negative edge cycle\n");
    }
    FinalSolution(StoreDistance, V);
    return;
}

int main()
```

```
{
    int V,E,S;  //V = no.of Vertices, E = no.of Edges, S is source vertex

printf("Enter number of vertices in graph\n");
    scanf("%d",&V);
printf("Enter number of edges in graph\n");
    scanf("%d",&E);
printf("Enter your source vertex number\n");
scanf("%d",&S);

    struct Graph* graph = createGraph(V, E);    //calling the function to
allocate space to these many vertices and edges

    int i;
    for(i=0;i<E;i++){
        printf("\nEnter edge %d properties Source, destination, weight
respectively\n",i+1);
        scanf("%d",&graph->edge[i].source);
        scanf("%d",&graph->edge[i].destination);
        scanf("%d",&graph->edge[i].weight);
    }

    BellmanFord(graph, S);
//passing created graph and source vertex to BellmanFord Algorithm function

    return 0;
}
```

7.0/1 Knapsack

```
/* A Naive recursive implementation
of 0-1 Knapsack problem */
#include <stdio.h>

// A utility function that returns
// maximum of two integers
int max(int a, int b) { return (a > b) ? a : b; }

// Returns the maximum value that can be
// put in a knapsack of capacity W
int knapSack(int W, int wt[], int val[], int n)
{
    // Base Case
    if (n == 0 || W == 0)
        return 0;

    // If weight of the nth item is more than
    // Knapsack capacity W, then this item cannot
```

```
    // be included in the optimal solution
    if (wt[n - 1] > W)
        return knapSack(W, wt, val, n - 1);

    // Return the maximum of two cases:
    // (1) nth item included
    // (2) not included
    else
        return max(
            val[n - 1]+ knapSack(W - wt[n - 1],wt, val, n - 1),
            knapSack(W, wt, val, n - 1));
}

// Driver program to test above function
int main()
{
    int W,n;
    printf("Enter weight of bag: ");
    scanf("%d",&W);
    printf("Enter Number of Elements: ");
    scanf("%d",&n);
    int val[n], wt[n];
    printf("Enter Weight Values\n");
    for(int i=0;i<n;i++)
    {
        printf("Enter weight %d: ",i+1);
        scanf("%d",&wt[i]);
    }
    printf("Enter Profit Values\n");
    for(int i=0;i<n;i++)
    {
        printf("Enter profit %d: ",i+1);
        scanf("%d",&val[i]);
    }
    printf("%d", knapSack(W, wt, val, n));
    return 0;
}
```

8. Fractional Knapsack

```
#include <stdio.h>

int main()
{
    int t,temp1,temp2;
    float m=0,temp,pro=0,b;
    printf("Enter Space of bag: \n");
    scanf("%f",&b);
```

```
    printf("Enter number of elements: \n");
    scanf("%d",&t);
    float s[t];
    float p[t],w[t];
    for(int i=0;i<t;i++)
    {
        printf("\nEnter W%d : ",i+1);
        scanf("%f",&w[i]);
        printf("\nEnter P%d : ",i+1);
        scanf("%f",&p[i]);
    }
    for(int q=0;q<t;q++)
    {
        s[q]=p[q]/w[q];

    }
    for(int c=0;c<=t-1;c++)
    {
        for(int j=0;j<=t-j-1;j++)
        if(s[j]<s[j+1])
        {
            temp=s[j];
            s[j]=s[j+1];
            s[j+1]=temp;

            temp1=w[j];
            w[j]=w[j+1];
            w[j+1]=temp1;


            temp2=p[j];
            p[j]=p[j+1];
            p[j+1]=temp2;
        }
    }
    for(int u=0;u<t;u++)
    {
        printf("\ns: %f w: %f p: %f \n",s[u],w[u],p[u]);
    }
    int k=0;
    while(k<t)
    {
        if(k==0 && w[k]<b)
        {
            pro=pro+p[k];
            printf("pro : %f\n",pro);
            m=m+w[k];
            printf("m : %f\n",m);
            printf("Bag is filled with Profit %f and Weight %f\n",pro,m);
```

```
        }
        else
        {
            float y=((b-m)/w[k]);
            pro=pro+(y*p[k]);
            printf("pro : %f\n",pro);
            m=m+(y*w[k]);
            printf("m : %f\n",m);
            printf("Bag is filled with Profit %f and Weight %f\n",pro,m);
        }
        if(m>b || m==b)
        {
            break;
        }
        k++;
    }
}
```

9. Prims

```
#include<stdio.h>
#include<string.h>
int adj[100][100],visited[10]={0},mincost=0,min;
void input(int n){
for(int i=1;i<=n;i++){
    for(int j=1;j<=n;j++){
        printf("enter element in [%d][%d]:",i,j);
        scanf("%d",&adj[i][j]);
    }
}
}

void display(int n){
for(int i=1;i<=n;i++){
    for(int j=1;j<=n;j++){
        printf("%d\t",adj[i][j]);
    }
    printf("\n");
}
}
void prims(int n){
    int ne=1,a,b,i,j;
    for(int i=1;i<=n;i++){
        for(int j=1;j<=n;j++){
            if(adj[i][j]==0){
                adj[i][j]=999;
            }
        }
```

```
    }

    visited[1]=1;
    while(ne<n){
        for(i=1,min=999;i<=n;i++){
            for(j=1;j<=n;j++){
                if(adj[i][j]<min){
                    if(visited[i]!=0){
                        min=adj[i][j];
                        a=i;
                        b=j;
                    }
                }

            }
        }
        printf("\n Edge %d:(%d - %d) cost:%d",ne++,a,b,min);
        visited[b]=1;
        mincost=mincost+min;
        adj[a][b]=adj[b][a]=999;
    }
    printf("min cost is: %d",mincost);

}
void main(){
    int nodes;
printf("Enter number of vertices");
scanf("%d",&nodes);
input(nodes);
display(nodes);
prims(nodes);
}
```

10. Kruskal

```
#include<stdio.h>
#include<string.h>
int adj[100][100],mincost=0,min,parent[9];

void input(int n){
for(int i=1;i<=n;i++){
    for(int j=1;j<=n;j++){
        printf("enter element in [%d][%d]:",i,j);
        scanf("%d",&adj[i][j]);
    }
}
}
void display(int n){
```

```
for(int i=1;i<=n;i++){
    for(int j=1;j<=n;j++){
        printf("%d\t",adj[i][j]);
    }
    printf("\n");
}
}
int find(int i)
    {
        while(parent[i])
        i=parent[i];
        return i;
    }
int uni(int i,int j)
    {
        if(i!=j)
        {
        parent[j]=i;
        return 1;
        }
        return 0;
    }
void kruskal(int n){
    int i,j,k,a,b,u,v,ne=1;
     for(int i=1;i<=n;i++){
        for(int j=1;j<=n;j++){
            if(adj[i][j]==0){
                adj[i][j]=999;
            }
        }
    }
    while(ne < n)
        {
        for(i=1,min=999;i<=n;i++)
        {
        for(j=1;j <= n;j++)
        {
        if(adj[i][j] < min)
        {
        min=adj[i][j];
        a=u=i;
        b=v=j;
        }
        }
        }
        u=find(u);
        v=find(v);
        if(uni(u,v))
```

```
        {
        printf("%d edge (%d,%d) =%d\n",ne++,a,b,min);
        mincost +=min;
        }
        adj[a][b]=adj[b][a]=999;
        }
        printf("\n\tMinimum cost = %d\n",mincost);
}
void main(){
    int nodes;
printf("Enter number of vertices");
scanf("%d",&nodes);
input(nodes);
display(nodes);
kruskal(nodes);
}
```

11. N Queen

```
#include <stdio.h>
#include<math.h>
int board[20],count;
int main()
{
 int n,i,j;
 void queen(int row,int n);
 printf("\n enter no of queen's ");
 scanf("%d",&n);
 queen(1,n);
 return 0;
}
void print_board(int n)
{
 int i,j;
 printf("\n\n solution %d\n ",++count);

 for(i=1;i<=n;i++)
 {
 printf("\t%d",i);
 }
 for(i=1;i<=n;i++)
 {
 printf("\n\n%d\t",i);
 for(j=1;j<=n;j++)
 {
 if (board[i]==j)
 {
 printf("Q\t");
```

```
 }
 else
 {
 printf("-\t");
 }
 }
 }
}
int place(int row,int column)
{
 int i;
 for(i=1;i<=row-1;i++)
 {
 if(board[i]==column)
 return 0;
 else
 {
 if(abs(board[i]-column)== abs(i-row))
 return 0;
 }
 }
 return 1;
}
void queen(int row,int n)
{
 int column;
 for(column=1;column<=n;column++)
 {
 if ( place(row,column) )
 {
 board[row]=column;
 if(row==n)
 {
 print_board(n);
 }
 else
 {
 queen(row+1,n);
 }
 }
 }
}
```

13. Floyd Warshall Algo

```
// C Program for Floyd Warshall Algorithm
#include<stdio.h>
```

```c
// Number of vertices in the graph
#define V 4

/* Define Infinite as a large enough
  value. This value will be used
  for vertices not connected to each other */
#define INF 99999

// A function to print the solution matrix
void printSolution(int dist[][V]);

// Solves the all-pairs shortest path
// problem using Floyd Warshall algorithm
void floydWarshall (int graph[][V])
{
    /* dist[][] will be the output matrix
      that will finally have the shortest
      distances between every pair of vertices */
    int dist[V][V], i, j, k;

    /* Initialize the solution matrix
      same as input graph matrix. Or
       we can say the initial values of
       shortest distances are based
       on shortest paths considering no
       intermediate vertex. */
    for (i = 0; i < V; i++)
        for (j = 0; j < V; j++)
            dist[i][j] = graph[i][j];

    /* Add all vertices one by one to
      the set of intermediate vertices.
      ---> Before start of an iteration, we
      have shortest distances between all
      pairs of vertices such that the shortest
      distances consider only the
      vertices in set {0, 1, 2, .. k-1} as
      intermediate vertices.
      ----> After the end of an iteration,
      vertex no. k is added to the set of
      intermediate vertices and the set
      becomes {0, 1, 2, .. k} */
    for (k = 0; k < V; k++)
    {
        // Pick all vertices as source one by one
        for (i = 0; i < V; i++)
        {
```

```
            // Pick all vertices as destination for the
            // above picked source
            for (j = 0; j < V; j++)
            {
                // If vertex k is on the shortest path from
                // i to j, then update the value of dist[i][j]
                if (dist[i][k] + dist[k][j] < dist[i][j])
                    dist[i][j] = dist[i][k] + dist[k][j];
            }
        }
    }

    // Print the shortest distance matrix
    printSolution(dist);
}

/* A utility function to print solution */
void printSolution(int dist[][V])
{
    printf ("The following matrix shows the shortest distances"
            " between every pair of vertices \n");
    for (int i = 0; i < V; i++)
    {
        for (int j = 0; j < V; j++)
        {
            if (dist[i][j] == INF)
                printf("%7s", "INF");
            else
                printf ("%7d", dist[i][j]);
        }
        printf("\n");
    }
}

// driver program to test above function
int main()
{
    /* Let us create the following weighted graph
            10
       (0)------->(3)
        |         /|\
      5 |          |
        |          | 1
       \|/         |
       (1)------->(2)
            3           */
    int graph[V][V] = { {0,   5,  INF, 10},
                        {INF, 0,   3, INF},
```

```
                        {INF, INF, 0,   1},
                        {INF, INF, INF, 0}
                    };

    // Print the solution
    floydWarshall(graph);
    return 0;
}
```