Assignment-1

Source Code:

```
#include <stdio.h>
int mid = 0;
int binary(int ar[], int l, int u, int sc)
{
    if (u >= l)
    {
        mid = (l + u) / 2;
        if (ar[mid] == sc)
        return mid;
        if (ar[mid] < sc)
        return binary(ar, mid + 1, u, sc);
        return binary(ar, l, mid - 1, sc);
    }
    return 0;
}
int bubble(int ar[], int n, int s)
{
    int i, j, temp = 0;
    for(i=0;i<n;i++)
    {
        for(j=0;j<n-i-1;j++)
        {
            if(ar[j]>ar[j+1])
            {
                temp=ar[j];
                ar[j]=ar[j+1];
                ar[j+1]=temp;
            }
        }
    }
    binary(ar, 0, n - 1, s);
}
int main()
{
    int sz, sch, i, res = 0;
    printf("Enter the Size of the Array: ");
    scanf("%d", &sz);
    int arr[sz];
    printf("Enter Elements:\n");
    for (i = 0; i < sz; i++)
    scanf("%d", &arr[i]);
    printf("Enter the Search Element: ");
    scanf("%d", &sch);
    res = bubble(arr, sz, sch);
    if (res == 0)
    printf("Search Element Not Found!");
    else
    printf("The Search Element is found at
position %d.", mid + 1);
    return 0;
}
```

Output:

Run 1:

```
Enter the Size of the Array: 6
Enter Elements:
12
78
35
32
159
32
Enter the Search Element: 159
Sorted Array:
12
32
32
35
78
159
The Search Element is found at position 6.
```

Run 2:

```
Enter the Size of the Array: 5
Enter Elements:
8
45
25
16
759
Enter the Search Element: 25
Sorted Array:
8
16
25
45
759
The Search Element is found at position 3.
```

Run 3:

```
Enter the Size of the Array: 5
Enter Elements:
12
97
126
35
45
Enter the Search Element: 2
Sorted Array:
12
35
45
97
126
Search Element Not Found!
```

Assignment-2

Source Code:

```
#include <stdio.h>
#include <stdlib.h>
int n = 0;
void merge(int la[], int ra[], int ar[], int lt, int rt)
{
    int i = 0, l = 0, r = 0;
    while (l < lt && r < rt)
    {
        if (la[l] <= ra[r])
        {
            ar[i++] = la[l++];
        }
        else
        {
            ar[i++] = ra[r++];
        }
    }
    while (l < lt)
    {
        ar[i++] = la[l++];
    }
    while (r < rt)
    {
        ar[i++] = ra[r++];
    }
}

void mergesort(int ar[], int l)
{
    if (l <= 1)
    return;

    int mid = l / 2;
    int *leftar = (int *)malloc(mid * sizeof(int));
    int *rightar = (int *)malloc((l - mid) *
sizeof(int));

    for (int i = 0; i < mid; i++)
    {
        leftar[i] = ar[i];
    }
    for (int i = mid; i < l; i++)
    {
        rightar[i - mid] = ar[i];
    }
    mergesort(leftar, mid);
    mergesort(rightar, l - mid);
    merge(leftar, rightar, ar, mid, l - mid);

    free(leftar);
    free(rightar);
}

int main()
{
    printf("Enter the size of the array: ");
    scanf("%d", &n);
    int ar[n];

    printf("Enter Elements:\n");
    for (int i = 0; i < n; i++)
        scanf("%d", &ar[i]);

    printf("\nORIGINAL ARRAY:\n");
    for (int i = 0; i < n; i++)
        printf("A[%d]=%d\n", i, ar[i]);

    mergesort(ar, n);

    printf("\nSORTED ARRAY:\n");
    for (int i = 0; i < n; i++)
        printf("A[%d]=%d\n", i, ar[i]);

    return 0;
}
```

Output:

Run 1:

```
Enter the size of the array: 5
Enter Elements:
98
21
32
49
35
ORIGINAL ARRAY:
A[0]=98
A[1]=21
A[2]=32
A[3]=49
A[4]=35
SORTED ARRAY:
A[0]=21
A[1]=32
A[2]=35
A[3]=49
A[4]=98
```

Assignmnet-3

Source Code:

```
#include <stdio.h>

int partition(int ar[], int s, int e)
{
    int i = s - 1,temp=0;
    int pvt = ar[e];
    for (int j = s; j < e; j++)
    {
        if (ar[j] < pvt)
        {
            i++;
            temp = ar[i];
            ar[i] = ar[j];
            ar[j] = temp;
        }    }
    i++;
    temp = ar[i];
    ar[i] = ar[e];
    ar[e] = temp;

    return i;
}

void quicksort(int ar[], int s, int e)
{
    if (e <= s)
        return;

    int pivot = partition(ar, s, e);
    quicksort(ar, s, pivot - 1);
    quicksort(ar, pivot + 1, e);
}

int main()
{
    int n = 0;
    printf("Enter the size of the array: ");
    scanf("%d", &n);
    int ar[n];
    printf("Enter Elements:\n");
    for (int i = 0; i < n; i++)
        scanf("%d", &ar[i]);
    printf("\nORIGINAL ARRAY:\n");
    for (int i = 0; i < n; i++)
        printf("A[%d]=%d\n", i, ar[i]);
    quicksort(ar, 0, n - 1);

    printf("\nSORTED ARRAY:\n");
    for (int i = 0; i < n; i++)
        printf("A[%d]=%d\n", i, ar[i]);
    return 0;
}
```

Output:

Run 1:

```
Enter the size of the array: 5
Enter Elements:
21
98
32
47
65
ORIGINAL ARRAY:
A[0]=21
A[1]=98
A[2]=32
A[3]=47
A[4]=65
SORTED ARRAY:
A[0]=21
A[1]=32
A[2]=47
A[3]=65
A[4]=98
```

Run 2:

```
Enter the size of the array: 6
Enter Elements:
31
48
35
67
21
2
ORIGINAL ARRAY:
A[0]=31
A[1]=48
A[2]=35
A[3]=67
A[4]=21
A[5]=2
SORTED ARRAY:
A[0]=2
A[1]=21
A[2]=31
A[3]=35
A[4]=48
A[5]=67
```

Assignment-4

Source Code:

```
#include <stdio.h>

struct Pair
{
    int min;
    int max;
};

struct Pair getMinMax(int arr[], int low, int
high)
{
    struct Pair minmax, mml, mmr;

    if (low == high)
    {
        minmax.min = arr[low];
        minmax.max = arr[low];
        return minmax;
    }

    if (high == low + 1) {
        if (arr[low] > arr[high]) {
            minmax.max = arr[low];
            minmax.min = arr[high];
        } else {
            minmax.max = arr[high];
            minmax.min = arr[low];
        }
        return minmax;
    }

    int mid = low + (high - low) / 2;
    mml = getMinMax(arr, low, mid);
    mmr = getMinMax(arr, mid + 1, high);

    if (mml.max > mmr.max)
        minmax.max = mml.max;
    else
        minmax.max = mmr.max;

    if (mml.min < mmr.min)
        minmax.min = mml.min;
    else
        minmax.min = mmr.min;

    return minmax;
}

int main() {
    int n;
    printf("Enter the number of elements in the
array: ");
    scanf("%d", &n);

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

    struct Pair result = getMinMax(arr, 0, n - 1);
    printf("Minimum element is %d\n",
result.min);
    printf("Maximum element is %d\n",
result.max);

    return 0;
}
```

Output:

Run 1:

```
Enter the number of elements in the array: 5
Enter 5 elements:
65
78
213
65
45
Minimum element is 45
Maximum element is 213
```

Run 2:

```
Enter 6 elements:
324
5184
151
141
855
54
Minimum element is 54
Maximum element is 5184
```

Assignment-5

Source Code:

```
# include<stdio.h>
#define MAX 50
void sort(int weights[], int vals[],int n){
   int i,j;
   for(i=0;i<n-1;i++){
      for(j=0;j<n-i-1;j++){
         if(vals[j]*weights[j+1] <
vals[j+1]*weights[j]){
            int temp1=weights[j];
            int temp2=vals[j];
            weights[j]=weights[j+1];
            vals[j]=vals[j+1];
            weights[j+1]=temp1;
            vals[j+1]=temp2;
         }
      }
   }
}

int knapsack(int x, int weights[], int
vals[], int n){
   sort(weights, vals, n);
   int totalvals = 0;
   int i;
   for(i=0;i<n;i++){
      if(x >= weights[i])
      {
         x -= weights[i];
         totalvals += vals[i];

      }
      else
      {
         totalvals += ((float)vals[i] * x) /
weights[i];
         break;
      }
   }
   return totalvals;
}
int main(){
   int weights[MAX], vals[MAX],x,i,n;
   printf("\n enter the number of
elements:");
   scanf("%d",&n);
   printf("\n enter the weights:");
   for(i=0;i<n;i++){
      printf("\n Weights:");
      scanf("%d",&weights[i]);
   }
   printf("\n enter the vals:");
   for(i=0;i<n;i++){
      printf("\n Values:");
      scanf("%d",&vals[i]);
   }
   printf("\n value and weight ratio:");
   for(i=0;i<n;i++){
      float ratio = (float)vals[i]/weights[i];
      printf("ratio: %.2f\n",ratio);
   }
   printf("\n enter  the capacity:");
   scanf("%d",&x);
   float maxval = knapsack(x, weights,
vals, n);
   printf("max value : %.2f\n", maxval);

   return 0;
}
```

Output:

```
enter the number of elements:3

 enter the weights:
 Weights:10
 Weights:12
 Weights:15

 enter the vals:
 Values:20
 Values:15
 Values:15

 value and weight ratio: ratio: 2.00
ratio: 1.25
ratio: 1.00

 enter  the capacity:15
max value : 26.00
```

Assignment-6

Source Code

```
#include <stdio.h>
#include <stdbool.h>
#define MAX 100

void swap(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}
void sort(int n, int profit[], int deadline[],
int id[]) {
    int i, j;
    for (i = 0; i < n - 1; i++)
        for (j = 0; j < n - 1 - i; j++)
            if (profit[j] < profit[j + 1]) {
                swap(&profit[j], &profit[j + 1]);
                swap(&deadline[j],
&deadline[j + 1]);
                swap(&id[j], &id[j + 1]);
            }}
int get(int deadline[], int n) {
    int max = deadline[0];
    for (int i = 1; i < n; i++)
        if (deadline[i] > max)
            max = deadline[i];
    return max;
}
void func(int n, int profit[], int deadline[],
int id[]) {
    sort(n, profit, deadline, id);
    int max = get(deadline, n);
    int seq[MAX];
    bool slot[MAX] = {false};
    int total = 0;
    for (int i = 0; i < max; i++)
        seq[i] = -1;
    for (int i = 0; i < n; i++) {
      for (int j = deadline[i] - 1; j >= 0; j--)
{
            if (!slot[j]) {
                seq[j] = id[i];
                slot[j] = true;
                total += profit[i];
                break;
            }       }    }

    printf("Selected Job Sequence: ");
    for (int i = 0; i < max; i++)
        if (seq[i] != -1)
            printf("J%d ", seq[i]);
    printf("\nTotal Profit: %d\n", total);
}
int main() {
    int i, n;
    printf("Enter no. of jobs: ");
    scanf("%d", &n);
    int profit[MAX], deadline[MAX],
id[MAX];
    printf("Enter job profits:\n");
    for (i = 0; i < n; i++)
        scanf("%d", &profit[i]);
    printf("Enter job deadlines:\n");
    for (i = 0; i < n; i++)
        scanf("%d", &deadline[i]);

    for (i = 0; i < n; i++)
        id[i] = i + 1;
    func(n, profit, deadline, id);
    return 0;
}
```

Output-

```
Enter no. of jobs: 4
Enter job profits:
100 19 27 25
Enter job deadlines:
2 1 2 1
Selected Job Sequence: J1 J3
Total Profit: 127
```

Assignment-7
Source code

```
#include <stdio.h>

#define SIZE 50

int d[SIZE], m[SIZE][SIZE], s[SIZE][SIZE];

void matrix_chain_order(int len, int p[]) {
    int i, j, k, l, n = len - 1, q;
    for (i = 1; i <= n; i++) m[i][i] = 0;
    for (l = 2; l <= n; l++)
        for (i = 1; i <= n - l + 1; i++) {
            j = i + l - 1; m[i][j] = 1e9;
            for (k = i; k < j; k++) {
                q = m[i][k] + m[k+1][j] + p[i-1]*p[k]*p[j];
                if (q < m[i][j]) { m[i][j] = q; s[i][j] = k; }
            }       }
    printf("\nCost of multiplication = %d\n", m[1][n]);
}

void print_optimal(int i, int j) {
    if (i == j) printf("A[%d] ", i);
    else { printf("( "); print_optimal(i, s[i][j]); print_optimal(s[i][j]+1, j); printf(") "); }
}

int main() {
    int n, i, j;
    printf("Enter number of dimensions: ");
    scanf("%d", &n);
    printf("Enter dimension sequence: ");
    for (i = 0; i < n; i++) scanf("%d", &d[i]);
    matrix_chain_order(n, d);
    printf("Optimal Parenthesization: "); print_optimal(1, n - 1); printf("\n\n");
    printf("Cost Table:\n");
    for (i = 1; i < n; i++) {
        for (j = 1; j < i; j++) printf("\t");
        for (j = i; j < n; j++) printf("%5d\t", m[i][j]);
        printf("\n");
    }
    printf("\nSequence Table:\n");
    for (i = 1; i < n; i++) {
        for (j = 1; j <= i; j++) printf("\t");
        for (j = i + 1; j < n; j++) printf("%5d\t", s[i][j]);
        printf("\n");
    }
    return 0;
}
```

Output-

```
Enter the number of dimention : 6

Enter the dimention sequence: 30 35 15 5 10 20

Cost of multiplication = 11875
( ( A[1] ( A[2] A[3] ) ) ( A[4] A[5] ) )
cost table :
    0   15750    7875    9375   11875
            0    2625    4375    7125
                    0     750    2500
                            0    1000
                                    0

sequence table:
            1       1       3       3
                    2       3       3
                            3       3
                                    4
```

Assignmnet-8

Source code

```
#include<stdio.h>
#include<stdlib.h>
#define inf 99999
int mincost=0,g[20][20],visited[20];
int prims(int n)
{
	int min,i,j,k,v1,v2;
	for(i=1;i<=n;i++)
	visited[i]=0;
		visited[1]=1;
		for(k=1;k<=n-1;k++){
		min=inf;
		for(i=1;i<=n;i++){
			for(j=1;j<=n;j++){
				if(g[i][j]!=inf &&
(visited[i]==1&&visited[j]==0))
				{

	if(g[i][j]<min){

	min=g[i][j];
							v1=i;
							v2=j;
					}
				}

			}
		}
		visited[v1]=visited[v2]=1;
		mincost+=min;
		printf("\nEdge %d=%d---
%d",k,v1,v2);
	}
	return mincost;
}
void display(int n){
	int i,j;
	for(i=1;i<=n;i++){
		for(j=1;j<=n;j++)
		printf("%d\t",g[i][j]);
		printf("\n");
	}
}
int main(){
	int n,i,j;
	printf("\n enter the number of
vertices:");
	scanf("%d",&n);
	for(i=1;i<=n;i++)
	{
	    for(j=1;j<=n;j++){
		printf("\n enter weight of edge
%d----%d",i,j);
		scanf("%d",&g[i][j]);
		if(g[i][j] == 0)
		g[i][j] == inf;
	    }
	}
	printf("\n\n");
	display(n);
	printf("\nmincost=%d",prims(n));
}
```

Output-

```
Enter number of vertices: 4
Enter weight of edge 1 --- 1: 0
Enter weight of edge 1 --- 2: 4
Enter weight of edge 1 --- 3: 0
Enter weight of edge 1 --- 4: 6
Enter weight of edge 2 --- 1: 4
Enter weight of edge 2 --- 2: 0
Enter weight of edge 2 --- 3: 5
Enter weight of edge 2 --- 4: 0
Enter weight of edge 3 --- 1: 0
Enter weight of edge 3 --- 2: 5
Enter weight of edge 3 --- 3: 0
Enter weight of edge 3 --- 4: 7
Enter weight of edge 4 --- 1: 6
Enter weight of edge 4 --- 2: 0
Enter weight of edge 4 --- 3: 7
Enter weight of edge 4 --- 4: 0
Adjacency Matrix:
99999 4     99999 6
4     99999 5     99999
99999 5     99999 7
6     99999 7     99999

Edge 1 = 1 --- 2
Edge 2 = 2 --- 3
Edge 3 = 1 --- 4
Minimum Cost = 15
```

Assignment-10

Source code

```
#include <stdio.h>
#define MAX 10
void dijkstra(int G[MAX][MAX], int n, int start, int dist[MAX]) {
    int vis[MAX] = {0};
    int i, j, u;
    for (i = 0; i < n; i++) dist[i] = -1;
    dist[start] = 0;
    for (i = 0; i < n - 1; i++) {
        int min = -1;
        for (j = 0; j < n; j++)
            if (!vis[j] && dist[j] != -1 && (min == -1 || dist[j] < min))
                min = dist[j], u = j;
        if (min == -1) break;
        vis[u] = 1;
        for (j = 0; j < n; j++)
            if (G[u][j] > 0 && !vis[j])
                if (dist[j] == -1 || dist[u] + G[u][j] < dist[j])
                    dist[j] = dist[u] + G[u][j];
    }
}
int main() {
    int G[MAX][MAX], dist[MAX];
    int n, start, i, j;
    printf("Vertices: ");
    scanf("%d", &n);
    printf("Matrix:\n");
    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
            scanf("%d", &G[i][j]);
    printf("Start: ");
    scanf("%d", &start);
    dijkstra(G, n, start, dist);
    printf("Node\tDist\n");
    for (i = 0; i < n; i++) {
        if (dist[i] == -1)
            printf("%d\tUnreachable\n", i);
        else
            printf("%d\t%d\n", i, dist[i]);
    }
    return 0;
}
```

Output-

```
Vertices: 4
Matrix:
0 5 0 10
0 0 3 0
0 0 0 1
0 0 0 0
Start: 0
Node    Dist
0       0
1       5
2       8
3       9
```

Assignment 12

Source Code

```
#include <stdio.h>
#include <stdlib.h>
#define INF 1000000
int main() {
    int V, E;
    printf("Enter number of vertices and edges: ");
    scanf("%d %d", &V, &E);
    int from[E], to[E], weight[E];
    printf("Enter each edge in format: from to weight\n");
    for (int i = 0; i < E; i++) {
        scanf("%d %d %d", &from[i], &to[i], &weight[i]);
    }
    int source;
    printf("Enter source vertex: ");
    scanf("%d", &source);
    int dist[V];
    for (int i = 0; i < V; i++)
        dist[i] = INF;
    dist[source] = 0;
    for (int i = 1; i <= V - 1; i++) {
        for (int j = 0; j < E; j++) {
            if (dist[from[j]] != INF && dist[from[j]] + weight[j] < dist[to[j]]) {
                dist[to[j]] = dist[from[j]] + weight[j];
            }
        }
    }
    for (int j = 0; j < E; j++) {
        if (dist[from[j]] != INF && dist[from[j]] + weight[j] < dist[to[j]]) {
            printf("Graph contains negative weight cycle\n");
            return 0;
        }
    }
    printf("\nVertex\tDistance from Source %d\n", source);
    for (int i = 0; i < V; i++) {
        if (dist[i] == INF)
            printf("%d\tINF\n", i);
        else
            printf("%d\t%d\n", i, dist[i]);
    }
    return 0;
}
```

Output-

```
Enter number of vertices and edges: 5 8
Enter each edge in format: from to weight
0 1 -1
0 2 4
1 2 3
1 3 2
1 4 2
3 2 5
3 1 1
4 3 -3
Enter source vertex: 0
Vertex  Distance from Source 0
0       0
1       -1
2       2
3       -2
4       1
```