ALGORITHM - SELECTION SORT(arr, n)

Step 1: Repeat Steps 2 and 3 for i = 0 to n-1

Step 2: CALL SELECTIONFUN (arr, i, n, pos)

Step 3: SWAP arr[i] with arr[pos]

[END OF LOOP]

Step 4: EXIT

SELECTIONFUN (arr, i, n, pos)

Step 1: [INITIALIZE] SET SMALL = arr[i]

Step 2: [INITIALIZE] SET pos = i

Step 3: Repeat for j = i+1 to n

if (SMALL > arr[j])

SET SMALL = arr[j]

SET pos = j

[END OF if]

[END OF LOOP]

Step 4: RETURN pos

1A.Write a program to sort list of n elements using selection sort.

```
#include <stdio.h>
void selectionSort(int arr[], int n) {
 int i, j, minIndex, temp;
 // Loop over each element in the array
 for (i = 0; i < n-1; i++) {
 // Assume the current element is the minimum
 minIndex = i;
 // Find the minimum element in the remaining unsorted part of the
array
 for (j = i+1; j < n; j++) {
 if (arr[j] < arr[minIndex]) {
 minIndex = j;
 }
 }
 // Swap the found minimum element with the first element of the
unsorted part
 if (minIndex != i) {
 temp = arr[i];
 arr[i] = arr[minIndex];
 arr[minIndex] = temp;
 }
 }
}
void printArray(int arr[], int n) {
```

```
 for (int i = 0; i < n; i++) {
 printf("%d ", arr[i]);
 }
 printf("\n");
}
int main() {
 int n;
 // Get the number of elements
 printf("Enter the number of elements: ");
 scanf("%d", &n);
 int arr[n];
 // Get the array elements
 printf("Enter the elements of the array:\n");
 for (int i = 0; i < n; i++) {
 scanf("%d", &arr[i]);
 }
 // Call the selection sort function
 selectionSort(arr, n);
 // Print the sorted array
 printf("Sorted array: \n");
 printArray(arr, n);
 return 0;
}
```

Output:

Enter the number of elements: 5

Enter the elements of the array:

33 5 66 7 9

Sorted array:

5 7 9 33 66

Algorithm:(travelling salesman problem)

1. Start at an arbitrary city.
2. At each step, visit the nearest unvisited city.
3. Repeat until all cities are visited.
4. Return to the starting city.

2A. Write a program to perform travelling salesman problem.

```
#include <stdio.h>
int tsp_g[20][20], visited[10], n, cost = 0;
void travellingsalesman(int c, int n)
{
 int k, adj_vertex = 999;
 int min = 999;
 visited[c] = 1;
printf("%d -> ", c+1);
 for(k=0; k<n; k++)
 {
 if((tsp_g[c][k] != 0) && (visited[k] == 0))
 {
 if(tsp_g[c][k] < min)
 {
 min = tsp_g[c][k];
 }
 adj_vertex = k;
 }
 }
if(min != 999)
 {
 cost = cost + min;
 }
if(adj_vertex == 999)
```

```
{
 adj_vertex = 0;
 printf("%d", adj_vertex + 1);
 cost = cost + tsp_g[c][adj_vertex];
 return;
}
travellingsalesman(adj_vertex,n);
}
int main()
{
 int i, j,n;
 printf("Enter the number of villages: ");
 scanf("%d",&n);
 printf("\nEnter the Cost Matrix\n");
 for(i=0;i < n;i++)
 {
printf("\nEnter Elements of Row: %d\n",i+1);
 for( j=0;j < n;j++)
 scanf("%d",&tsp_g[i][j]);
visited[i]=0;
 }

 printf("\n\nThe cost list is:");
for( i=0;i < n;i++)
 {
```

```
printf("\n");
 for(j=0;j < n;j++)
printf("\t%d",tsp_g[i][j]);
 }
 printf("\nShortest Path: ");
travellingsalesman(0,n);
printf("\nMinimum Cost: ");
printf("%d\n", cost);
 return 0;
}
```

Output:

Enter the number of villages: 4

Enter the Cost Matrix

Enter Elements of Row: 1

0 4 3 8

Enter Elements of Row: 2

4 0 6 7

Enter Elements of Row: 3

3 6 0 9

Enter Elements of Row: 4

8 7 9 0

The cost list is:

0 4 3 8

4 0 6 7

3 6 0 9

8 7 9 0

Shortest Path: 1-> 4-> 3-> 2-> 1

Minimum Cost : 20

Algorithm for Greedy Knapsack :

1. **Compute the value-to-weight ratio** (ri=vi / wi) for each item.
2. **Sort the items** in descending order of ri
3. **Initialize** the knapsack's total weight and value to 0.
4. **Iterate through the sorted items:**

   If the item's weight can fit entirely in the knapsack, add it to the knapsack.

   If the item's weight cannot fit, take the fractional part of the item that fits, and stop further iterations.

5. **Return the total value of the knapsack**

3A. write a program to perform knapsack problem using greedy solution.

```
#include <stdio.h>
typedef struct
{ int weight; int value;
double valuePerWeight; // Value-to-Weight ratio
} Item;
void fractionalKnapsack(Item items[], int n, int capacity) {
// Calculate value-to-weight ratio for each item
for (int i = 0; i < n; i++) {
items[i].valuePerWeight = (double) items[i].value / items[i].weight;
}
// Sort items based on value-to-weight ratio (descending order)
for (int i = 0; i < n - 1; i++) {
for (int j = 0; j < n - i - 1; j++) {
if (items[ j ].valuePerWeight < items[j + 1].valuePerWeight) {
// Swap
Item temp = items[j];
items[j] = items[j + 1];
items[j + 1] = temp;
}
}
}
double totalValue = 0.0;
// Fill the knapsack
for (int i = 0; i < n && capacity > 0; i++) {
```

```
if (items[i].weight <= capacity) {
// Take the whole item
totalValue += items[i].value;
capacity -= items[i].weight;
} else {
// Take a fraction of the item
double fraction = (double) capacity / items[i].weight;
totalValue += fraction * items[i].value;
capacity = 0;
}
printf("Maximum value in the knapsack: %.2f\n", totalValue);
}
int main() {
int n, capacity;
printf("Enter the number of items: ");
scanf("%d", &n);
Item items[n];
printf("Enter the weight and value for each item:\n");
for (int i = 0; i < n; i++)
{
printf("Item %d:\n", i + 1);
printf("Weight: ");
scanf("%d", &items[i].weight);
printf("Value: ");
scanf("%d", &items[i].value);
```

```
}
printf("Enter the knapsack capacity: ");
scanf("%d", &capacity);
fractionalKnapsack(items, n, capacity);
return 0;
}
```

Output:

Enter the number of items: 3

Item 1: Weight: 10

Value: 60

Item 2: Weight: 20

Value: 100

Item 3: Weight: 30

Value: 120

Enter the knapsack capacity: 50

Maximum value in the knapsack: 240.0

Algorithm DFS :

1. First, create a stack with the total number of vertices in the graph.
2. Now, choose any vertex as the starting point of traversal, and push that vertex into the stack.
3. After that, push a non-visited vertex (adjacent to the vertex on the top of the stack) to the top of the stack.
4. Now, repeat steps 3 and 4 until no vertices are left to visit from the vertex on the stack's top.
5. If no vertex is left, go back and pop a vertex from the stack.
6. Repeat steps 2, 3, and 4 until the stack is empty.

4A. Write Program to perform DFS

```
#include <stdio.h>
int top = -1, a[20][20] = {0}, vis[20] = {0}, stack[20];
void dfs(int s, int n);
void push(int item);
int pop();
void main() {
 int n, e, i, s, j, u, v;
 printf("Enter the number of vertices in the graph: ");
 scanf("%d", &n);
 printf("Enter the number of edges in the graph: ");
 scanf("%d", &e);
 printf("Enter the edges (u, v):\n");
 for (i = 1; i <= e; i++) {
 scanf("%d %d", &u, &v);
 a[u][v] = 1;
 }
 printf("THE ADJACENCY MATRIX IS\n");
 for (i = 1; i <= n; i++) {
 for (j = 1; j <= n; j++) {
 printf(" %d", a[i][j]);
 }
 printf("\n");
 }
 for (i = 1; i <= n; i++)
```

```
 vis[i] = 0;
 printf("ENTER THE SOURCE VERTEX: ");
 scanf("%d", &s);
 dfs(s, n);
}
void push(int item) {
 if (top == 19) {
 printf("Stack overflow\n");
 } else {
 stack[++top] = item;
 }
}
int pop() {
 if (top == -1) {
 return 0;
 } else {
 return stack[top--];
 }
}
void dfs(int s, int n) {
 int i, k;
 push(s);
 vis[s] = 1;

 while ((k = pop()) != 0) {
```

```
printf(" %d ", k);
for (i = 1; i <= n; i++) {
if (a[k][i] != 0 && vis[i] == 0) {
push(i);
vis[i] = 1;
}
}
}
for (i = 1; i <= n; i++) {
if (vis[i] == 0) {
dfs(i, n);
}
}
}
```

OUTPUT:

Enter the number of vertices in the graph: 5

Enter the number of edges in the graph: 4

Enter the edges (u, v):

1 2

1 3

3 4

3 5

THE ADJACENCY MATRIX IS

0 1 1 0 0

0 0 0 0 0

0 0 0 1 1

0 0 0 0 0

0 0 0 0 0

ENTER THE SOURCE VERTEX : 1

1 3 5 4 2

Algorithm BFS :

Step 1: SET STATUS = 1 (ready state) for each node in G

Step 2: Enqueue the starting node A and set its STATUS = 2 (waiting state)

Step 3: Repeat Steps 4 and 5 until QUEUE is empty

Step 4: Dequeue a node N. Process it and set its STATUS = 3 (processed state).

Step 5: Enqueue all the neighbours of N that are in the ready state (whose STATUS = 1) and set

their STATUS = 2 (waiting state)

[END OF LOOP]

Step 6: EXIT

5A: Write program to implement the BFS algorithm for a graph.

```
#include<stdio.h>
int q[20],front=-1,rear=-1,a[20][20],vis[20];
int delete();
void add(int item);
void bfs(int s,int n);
void main()
{
int n,e,i,s,j,u,v;
printf("Enter the number of vertices in the graph: ");
scanf("%d", &n);
printf("Enter the number of edges in the graph: ");
scanf("%d", &e);
printf("Enter the edges (u, v):\n");
for (i = 1; i<=e; i++)
{
scanf("%d %d", &u, &v);
a[u][v] = 1;
}
printf("THE ADJACENCY MATRIX IS\n");
for(i=1;i<=n;i++)
{
for(j=1;j<=n;j++)
{
printf(" %d",a[i][j]);
```

```
}
printf("\n");
}
for(i=1;i<=n;i++)
vis[i]=0;
printf("ENTER THE SOURCE VERTEX :");
scanf("%d",&s);
bfs(s,n);
}
void bfs(int s,int n)
{
int p,i;
add(s);
vis[s]=1;
p=delete();
if(p!=0)
printf(" %d",p);
while(p!=0)
{
for(i=1;i<=n;i++)
if((a[p][i]!=0)&&(vis[i]==0))
{
add(i);
vis[i]=1;
}
```

```
p=delete();
if(p!=0)
printf(" %d ",p);
}
for(i=1;i<=n;i++)
if(vis[i]==0)
bfs(i,n);
}
void add(int item)
{
if(rear==19)
printf("QUEUE FULL");
else
{
if(rear==-1)
{
q[++rear]=item;
front++;
}
else
q[++rear]=item;
}
}
int delete()
{
```

```
int k;
if((front>rear)||(front==-1))
return(0);
else
{
k=q[front++];
return(k);
}
}
```

OUTPUT :

Enter the number of vertices in the graph: 5

Enter the number of edges in the graph: 4

Enter the edges (u, v):

1 2

1 3

3 4

3 5

THE ADJACENCY MATRIX IS

0 1 1 0 0

0 0 0 0 0

0 0 0 1 1

0 0 0 0 0

0 0 0 0 0

ENTER THE SOURCE VERTEX :1

1 2 3 4 5

Algorithm MaxMin :

```
def find_max_min(arr, low, high):
    if low == high:
        return arr[low], arr[low]  # Base case: single element
    mid = (low + high) // 2
    left_max, left_min = find_max_min(arr, low, mid)
    right_max, right_min = find_max_min(arr, mid + 1, high)
    return max(left_max, right_max), min(left_min, right_min)
```

6A: Program to find minimum and maximum value in an array using divide and conquer technique

```
#include<stdio.h>
int max, min;
int a[100];
void maxmin(int i, int j)
{
int max1, min1, mid;
if(i==j)
{
max = min = a[i];
}
else
{
if(i == j-1)
{
if(a[i] <a[j])
{
max = a[j];
min = a[i];
}
else
{
max = a[i];
min = a[j];
```

```
}
}
else
{
mid = (i+j)/2;
maxmin(i, mid);
max1 = max; min1 = min;
maxmin(mid+1, j);
if(max <max1)
max = max1;
if(min > min1)
min = min1;
}
}
}
int main ()
{
int i, num;
printf ("\nEnter the total number of numbers : ");
scanf ("%d",&num);
printf ("Enter the numbers : \n");
for (i=1;i<=num;i++)
scanf ("%d",&a[i]);
max = a[0];
min = a[0];
```

```
maxmin(1, num);
printf ("Minimum element in an array : %d\n", min);
printf ("Maximum element in an array : %d\n", max);
return 0;
}
```

OUTPUT :

Enter the total number of numbers : 6

Enter the numbers :

4

8

2

1

0

9

Minimum element in an array : 0

Maximum element in an array : 9

Algorithm Quick Sort :

```
1. QUICKSORT (array A, start, end)
2. {
3.  if (start < end)
4.  {
5.  p = partition(A, start, end)
6.  QUICKSORT (A, start, p - 1)
7.  QUICKSORT (A, p + 1, end)
8.  }
9. }
```

Partition(arr A,start,end)

```
1. PARTITION (array A, start, end)
2. {
3.   pivot ? A[end]
4.   i ? start-1
5.   for j ? start to end -1 {
6.   do if (A[j] < pivot) {
7.   then i ? i + 1
8.   swap A[i] with A[j]
9.    }}
10. swap A[i+1] with A[end]
11. return i+1
12.}
```

7A. Write a test program to implement Divide and Conquer Strategy for Quick sort algorithm

```
#include <stdio.h>
void swap(int* a, int* b)
{
int temp = *a;
*a = *b;
*b=temp; }
int partition(int arr[], int low, int high)
{ int pivot = arr[high];
int i = low - 1;
for (int j = low; j < high; j++)
{
if (arr[ j ] < pivot)
{ i++;
swap(&arr[i], &arr[j]);
} }
swap(&arr[i + 1], &arr[high]);
return i + 1;
}
void quickSort(int arr[], int low, int high)
{
if (low < high)
{
int pi = partition(arr, low, high);
```

```
quickSort(arr, low, pi - 1);
quickSort(arr, pi + 1, high);
}
}
int main()
{ int n;
printf("Enter the number of elements in the array: ");
scanf("%d", &n);
int arr[n];
printf("Enter the elements of the array:\n");
for (int i = 0; i < n; i++)
{
scanf("%d", &arr[i]);
}
quickSort(arr, 0, n - 1);
printf("Sorted array using Quick Sort:\n");
for (int i = 0; i < n; i++)
{
printf("%d ", arr[i]);
}
return 0;
}
```

Output:

Enter the number of elements in the array: 5

Enter the elements of the array:

12 5 23 8 18

Sorted array using Quick Sort:

5 8 12 18 23

ALGORITHM-MERGE SORT

1. If p<r
2. Then q → (p+ r)/2
3. MERGE-SORT (A, p, q)
4. MERGE-SORT (A, q+1,r)
5. MERGE (A, p, q, r)

FUNCTIONS: MERGE (A, p, q, r)

1. n 1 = q-p+1
2. n 2= r-q
3. create arrays [1.....n 1 + 1] and R [1.....n 2 +1]
4. for i ← 1 to n 1
5. do [i] ← A [p+ i-1]
6. for j ← 1 to n2
7. do R[j] ← A[q + j]
8. L [n 1+ 1] ← ∞
9. R[n 2+ 1] ← ∞
10. I ← 1
11. J ← 1
12. For k ← p to r
13. Do if L [i] ≤ R[j]
14. then A[k] ← L[i]
15. i ← i +1
16. else A[k] ← R[j]
17. j ← j+1

8A. Write a program to implement Merge sort algorithm for sorting a list of integers in ascending order

```
#include <stdio.h>
void merge(int arr[], int left, int mid, int right)
{
int n1 = mid - left + 1;
int n2 = right - mid;
int leftArr[n1], rightArr[n2];
for (int i = 0; i < n1; i++)
{
leftArr[i] = arr[left + i];
}
for (int j = 0; j < n2; j++)
{
rightArr[j] = arr[mid + 1 + j];
}
int i = 0, j = 0, k = left;
while (i < n1 && j < n2)
{
if (leftArr[i] <= rightArr[j])
{
arr[k++] = leftArr[i++];
}
else
{
arr[k++] = rightArr[j++];
```

```
} }
while (i < n1)
{ arr[k++] = leftArr[i++];
}
while (j < n2)
{ arr[k++] = rightArr[j++];
}
}
void mergeSort(int arr[], int left, int right)
{
if (left < right)
{ int mid = left + (right - left) / 2;
mergeSort(arr, left, mid);
mergeSort(arr, mid + 1, right);
merge(arr, left, mid, right);
} }
int main()
{ int n;
printf("Enter the number of elements in the array: ");
scanf("%d", &n);
int arr[n];
printf("Enter the elements of the array:\n");
for (int i = 0; i < n; i++)
{ scanf("%d", &arr[i]);
} mergeSort(arr, 0, n - 1);
```

```
printf("Sorted array using Merge Sort:\n");
for (int i = 0; i < n; i++)
{ printf("%d ", arr[i]);
}
return 0;
}
```

Output:

Enter the number of elements in the array: 5

Enter the elements of the array:

12 5 23 8 18

Sorted array using Merge Sort:

5 8 12 18 23

PART- B

LAB 1B: Write C program that accepts the vertices and edges for a graph and stores it as an adjacency matrix.

```
#include <stdio.h>
#define MAX_VERTICES 100
int main()
{
int adjMatrix[MAX_VERTICES][MAX_VERTICES] = {0};
int numVertices, numEdges;
int i, j, u, v;
printf("Enter the number of vertices in the graph: ");
scanf("%d", &numVertices);
printf("Enter the number of edges in the graph: ");
scanf("%d", &numEdges);
printf("Enter the edges (u, v):\n");
for (i = 1; i <=numEdges; i++)
{
scanf("%d %d", &u, &v);
adjMatrix[u][v] = 1;
adjMatrix[v][u] = 1;
}
printf("\nAdjacency Matrix:\n");
for (i = 1; i <= numVertices; i++)
{
for (j = 1; j <= numVertices; j++)
{
```

```
printf("%d ", adjMatrix[i][j]);
}
printf("\n");
}
return 0;
}
```

OUTPUT :

Enter the number of vertices in the graph: 5 Enter

the number of edges in the graph: 8

Enter the edges (u, v):

1 2

2 5

5 4

4 1

1 3

2 3

4 3

5 3

Adjacency Matrix:

0 1 1 1 0

1 0 1 0 1

1 1 0 1 1

1 0 1 0 1

0 1 1 1 0

LAB 2B: Implement function to print In-Degree, Out-Degree and to display that adjacency matrix

```
#include <stdio.h>
#define MAX_VERTICES 100
void calc_out_degree(int G[MAX_VERTICES][MAX_VERTICES], int n)
{
int i,j,sum;
for(i=1;i<=n;i++)
{
sum=0;
for(j=1;j<=n;j++)
{
sum = sum + G[i][j];
}
printf("out-deg(V%d)=%d\n",i,sum);
}
}
void calc_in_degree(int G[MAX_VERTICES][MAX_VERTICES], int n)
{
int i,j,sum;
for(i=1;i<=n;i++)
{
sum=0;
for(j=1;j<=n;j++)
{
```

```
sum = sum + G[j][i];
}
printf("in-deg(V%d)=%d\n",i,sum);
}
}
int main()
{
int adjMatrix[MAX_VERTICES][MAX_VERTICES] = {0};
int numVertices, numEdges;
int i, j, u, v;
printf("Enter the number of vertices in the graph: ");
scanf("%d", &numVertices);
printf("Enter the number of edges in the graph: ");
scanf("%d", &numEdges);
printf("Enter the edges (u, v):\n");
for (i = 1; i <=numEdges; i++)
{
scanf("%d %d", &u, &v);
adjMatrix[u][v] = 1;
}
printf("\nAdjacency Matrix:\n");
for (i = 1; i <= numVertices; i++)
{
for (j = 1; j <= numVertices; j++)
{
```

```
printf("%d ", adjMatrix[i][j]);
}
printf("\n");
}
printf("Out degree:\n");
calc_out_degree(adjMatrix,numVertices);
printf("\n\nIn degree:\n");
calc_in_degree(adjMatrix,numVertices);
return 0;
}
```

OUTPUT :

Enter the number of vertices in the graph: 2

Enter the number of edges in the graph: 2

Enter the edges (u, v):

1 1

1 2

Adjacency Matrix:

1 1

0 0

Out degree:

out-deg(V1)=2

out-deg(V2)=0

In degree:

in-deg(V1)=1

in-deg(V2)=1

LAB 3B: Write program to implement backtracking algorithm for solving problems like N queens

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

int board[20],count;

int main()
{
int n,i,j;
void queen(int row,int n);

printf(" - N Queens Problem Using Backtracking -");
printf("\n\nEnter number of Queens:");
scanf("%d",&n);
queen(1,n); return 0;
}
void print(int n)
{ int i,j;
printf("\n\nSolution %d:\n\n",++count);

for(i=1;i<=n;++i)
 printf("\t%d",i);

for(i=1;i<=n;++i)
```

```
{ printf("\n\n%d",i);
 for(j=1;j<=n;++j) //for nxn board
 {
 if(board[i]==j)
 printf("\tQ"); //queen at i,j position
 else
 printf("\t-"); //empty slot
 }
}
}

int place(int row,int column)
{
int i;
for(i=1;i<=row-1;++i)
{
 //checking column and digonal conflicts
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; //no conflicts so place queen
 if(row==n) //dead end
 print(n); //printing the board configuration
 else //try queen with next position
queen(row+1,n);
 }
}
}
```

OUTPUT :

- N Queens Problem Using Backtracking -

Enter number of Queens: 4

Solution 1:

1 2 3 4

1 - Q - -

2 - - - Q

3 Q - - -

4 - - Q -

Solution 2:

1 2 3 4

1 - - Q -

2 Q - - -

3 - - - Q

4 - Q - -

4. Write a program to implement the backtracking algorithm for the sum of subsets problem

```
#include <stdio.h>
#include <stdbool.h>
void printSubset(int subset[], int size) {
printf("Subset: { ");
for (int i = 0; i < size; i++) {
printf("%d ", subset[i]);
}
printf("}\n");
}
void sumOfSubsetsUtil(int weights[], int targetSum, int n, int subset[], int
subsetSize, int sum,
int index) {
if (sum == targetSum)
{
 printSubset(subset, subsetSize);
return;
}
for (int i = index; i < n; i++) {
if (sum + weights[i] <= targetSum) {
subset[subsetSize] = weights[i];
sumOfSubsetsUtil(weights, targetSum, n, subset, subsetSize + 1, sum +
weights[i], i + 1);
}
```

```
}
}
void sumOfSubsets(int weights[], int targetSum, int n) {
int subset[n];
sumOfSubsetsUtil(weights, targetSum, n, subset, 0, 0, 0);
}
int main() {
int n, targetSum;
printf("Enter the number of elements: ");
scanf("%d", &n);
int weights[n];
printf("Enter the elements: ");
for (int i = 0; i < n; i++) {
scanf("%d", &weights[i]);
}
printf("Enter the target sum: ");
scanf("%d", &targetSum);
sumOfSubsets(weights, targetSum, n);
return 0;
}
```

Output:

Enter the number of elements: 5

Enter the elements: 10 7 5 1 3

Enter the target sum: 15

Sample Output (All Possible Subsets with Sum 15):

Subset: { 10 5 } Subset: { 7 5 3 }

5. Write program to implement Dynamic Programming algorithm for the 0/1 Knapsack problem.

```
#include <stdio.h>
#define MAX_ITEMS 100
int max(int a, int b) {
return (a > b) ? a : b;
}
int knapsack(int values[], int weights[], int n, int capacity) {
int dp[MAX_ITEMS + 1][capacity + 1];
for (int i = 0; i <= n; i++) {
for (int w = 0; w <= capacity; w++) {
if (i == 0 || w == 0) {
dp[i][w] = 0;
} else if (weights[i - 1] <= w) {
dp[i][w] = max(values[i - 1] + dp[i - 1][w - weights[i - 1]], dp[i - 1][w]);
} else {
dp[i][w] = dp[i - 1][w];
}
}
}
return dp[n][capacity];
}
int main() {
int n, capacity;
printf("Enter the number of items: ");
```

```
scanf("%d", &n);
int values[MAX_ITEMS], weights[MAX_ITEMS];
printf("Enter the values of items:\n");
for (int i = 0; i < n; i++) {
scanf("%d", &values[i]);
}
printf("Enter the weights of items:\n");
for (int i = 0; i < n; i++) {
scanf("%d", &weights[i]);
}
printf("Enter the capacity of the knapsack: ");
scanf("%d", &capacity);
int result = knapsack(values, weights, n, capacity);
printf("Maximum value in the knapsack: %d\n", result);
return 0;
}
```

Output:

Enter the number of items: 4

Enter the values of items:

2 3 1 4

Enter the weights of items:

3 4 2 5

Enter the capacity of the knapsack: 5

Maximum value in the knapsack: 4

6. Write program to implement Dynamic Programming algorithm for the Optimal Binary Search Tree Problem.

```
#include <stdio.h>
#include<limits.h>
#define MAX_KEYS 100
int optimalBST(int keys[], int freq[], int n) {
int dp[MAX_KEYS + 1][MAX_KEYS + 1];
for (int i = 0; i <= n; i++) {
for (int j = 0; j <= n; j++) {
dp[i][j] = 0;
}
}
for (int len = 1; len <= n; len++) {
for (int i = 0; i <= n - len + 1; i++) {
int j = i + len - 1;
dp[i][j] = INT_MAX;
for (int r = i; r <= j; r++) {
int c = ((r > i) ? dp[i][r - 1] : 0) + ((r < j) ? dp[r + 1][j] : 0) + freq[r];
if (c < dp[i][j]) {
dp[i][j] = c;
}
}
}
}
return dp[0][n - 1];
```

```
}
int main() {
int n;
printf("Enter the number of keys: ");
scanf("%d", &n);
int keys[MAX_KEYS], freq[MAX_KEYS];
printf("Enter the keys:\n");
for (int i = 0; i < n; i++)
{ scanf("%d", &keys[i]);
}
printf("Enter the frequencies:\n");
for (int i = 0; i < n; i++) {
scanf("%d", &freq[i]);
}
int result = optimalBST(keys, freq, n);
printf("Optimal cost of a Binary Search Tree: %d\n", result);
return 0;
}
```

Output:

Enter the number of keys: 4

Enter the keys:

10 12 20 25

Enter the frequencies:

34 8 50 12

Optimal cost of a Binary Search Tree: 104

LAB 7B: Write a program that implements Prim's algorithm to generate minimum cost spanning tree.

```
#include <limits.h>
#include <stdbool.h>
#include <stdio.h>
int V, graph[20][20];
int minKey(int key[], bool mstSet[])
{
 int v, min = INT_MAX, min_index;
 for (v = 0; v < V; v++)
 if (mstSet[v] == false && key[v] < min)
 min = key[v], min_index = v;
 return min_index;
}
int printMST(int parent[])
{
 int i,cost=0;
 for (i = 1; i < V; i++)
 cost = cost + graph[i][parent[i]];
 printf("Edge \tWeight\n");
 for (i = 1; i < V; i++)
 printf("%d - %d \t%d \n", parent[i]+1, i+1, graph[i][parent[i]]);
 printf("Minimum cost spanning tree = %d", cost);
}
```

```
void primMST()
{
 int parent[V], i, v, count;
 int key[V];
 bool mstSet[V];
for ( i = 0; i < V; i++)
 key[i] = INT_MAX, mstSet[i] = false;
 key[0] = 0;
 parent[0] = -1;
 for (count = 0; count < V - 1; count++)
 {
 int u = minKey(key, mstSet);
 mstSet[u] = true;
 for (v = 0; v < V; v++)
 if (graph[u][v] && mstSet[v] == false && graph[u][v] < key[v])
 parent[v] = u, key[v] = graph[u][v];
 }
 printMST(parent);
}
int main()
{
 int i, j;
 printf("Enter number of a vertices in graph:\n");
scanf("%d", &V);
 printf("Enter adjacency matrix for a graph:\n");
```

```
 for(i = 0;i < V; i++)
 {
 printf("Enter %d row values:\n", i+1);
 for(j = 0;j < V; j++)
 scanf("%d", &graph[i][j]);
 }
 primMST();
 return 0;
}
```

OUTPUT :

Enter number of a vertices in graph:

5

Enter adjacency matrix for a graph:

Enter 1 row values:

0 2 0 6 0

Enter 2 row values:

2 0 3 8 5

Enter 3 row values:

0 3 0 0 7

Enter 4 row values:

6 8 0 0 9

Enter 5 row values:

0 5 7 9 0

Edge Weight

1 - 2 2

2 - 3 3

1 - 4 6

2 - 5 5

Minimum cost spanning tree = 16

LAB 8B: Write a program that implements Kruskal's algorithm to generate minimum cost spanning tree.

```
#include<stdio.h>
int i,j,k,a,b,u,v,n,ne=1;
int min,mincost=0,cost[9][9],parent[9];
int find(int);
int uni(int,int);
void main()
{
printf("\n\tImplementation of Kruskal's algorithm\n");
printf("\nEnter the no. of vertices:");
scanf("%d",&n);
printf("\nEnter the cost adjacency matrix:\n");
for(i=1;i<=n;i++)
{
for(j=1;j<=n;j++)
{
scanf("%d",&cost[i][j]);
if(cost[i][j]==0)
cost[i][j]=999;
}
}
printf("The edges of Minimum Cost Spanning Tree are\n");
while(ne < n)
```

```
{
for(i=1,min=999;i<=n;i++)
{
for(j=1;j <= n;j++)
{
if(cost[i][j] < min)
{
min=cost[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;
}
cost[a][b]=cost[b][a]=999;
}
printf("\n\tMinimum cost = %d\n",mincost);
}
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;
}
```

OUTPUT :

Implementation of Kruskal's algorithm

Enter the no. of vertices:5

Enter the cost adjacency matrix:

0 2 0 6 0

2 0 3 8 5

0 3 0 0 7

6 8 0 0 9

0 5 7 9 0

The edges of Minimum Cost Spanning Tree are

1 edge (1,2) =2

2 edge (2,3) =3

3 edge (2,5) =5

4 edge (1,4) =6

Minimum cost = 16