1. QUICK SORT

```
#include <stdio.h>
void swap(int *a, int *b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
}

int partition(int array[], int low, int high)
{
    int pivot = array[high];
     int i = (low - 1);

    for (int j = low; j < high; j++)
    {
        if (array[j] <= pivot)
        {
            i++;
            swap(&array[i], &array[j]);
        }
    }
    swap(&array[i + 1], &array[high]);
    return (i + 1);
}

void quickSort(int array[], int low, int high)
{
    if (low < high)
    {
        int pi = partition(array, low, high);
        quickSort(array, low, pi - 1);
        quickSort(array, pi + 1, high);
    }
}

int main()
{
    int n,  array[100];
    printf("Enter the number of elements: ");
    scanf("%d", &n);

   for (int i = 0; i < n; i++)
    {
        printf("Enter element %d: ", i + 1);
        scanf("%d", &array[i]);
     }

    printf("\nBefore sorting: ");
    for (int i = 0; i < n; i++)
    {
        printf("%d ", array[i]);
     }

     quickSort(array, 0, n - 1);

    printf("\nAfter Sorting: ");
    for (int i = 0; i < n; i++)
    {
        printf("%d ", array[i]);
     }

    return 0;
}
```

2. MERGE SORT

```
#include <stdio.h>
#include <stdlib.h>
void merge(int arr[], int temp[], int left, int mid, int
right)
{
    int i, j, k;
    int left_end = mid - 1;
    int num_elements = right - left + 1;
    i = left,  j = mid;
    k = left;

    while (i <= left_end && j <= right)
    {
        if (arr[i] <= arr[j])
            temp[k++] = arr[i++];
        else
            temp[k++] = arr[j++];
    }

    while (i <= left_end)
        temp[k++] = arr[i++];

    while (j <= right)
        temp[k++] = arr[j++];

    for (i = 0; i < num_elements; i++, right--)
        arr[right] = temp[right];
}

void mergeSort(int arr[], int temp[], int left, int
right)
{
    int mid;
    if (right > left)
    {
        mid = (left + right) / 2;
        mergeSort(arr, temp, left, mid);
        mergeSort(arr, temp, mid + 1, right);
        merge(arr, temp, left, mid + 1, right);
    }
}

void mergeSortWrapper(int arr[], int n)
{
    int *temp = (int *)malloc(n * sizeof(int));
    if (temp != NULL)
    {
        mergeSort(arr, temp, 0, n - 1);
        free(temp);
    }
    else
    {
        printf("Memory allocation failed.\n");
        exit(1);
    }
}

int main()
{
    int n, i, arr[100];
    printf("Enter the number of elements: ");
    scanf("%d", &n);
    for (i = 0; i < n; i++)
    {
        printf("Enter element %d: ", i + 1);
        scanf("%d", &arr[i]);
    }
    mergeSortWrapper(arr, n);
    printf("Sorted array:\n");
    for (i = 0; i < n; i++)
    {
        printf("%d ", arr[i]);
    }
    return 0;
```

3. BINARY SEARCH

```
#include <stdio.h>

int binarySearch(int array[], int size, int search)
{
    int low = 0;
    int high = size - 1;

    while (low <= high)
    {
        int mid = low + (high - low) / 2;

        if (array[mid] == search)
            return mid;

        if (array[mid] < search)
            low = mid + 1;
        else
            high = mid - 1;
    }

    return -1;
}

void bubbleSort(int array[], int size)
{
    for (int i = 0; i < size - 1; ++i)
    {
        for (int j = 0; j < size - i - 1; ++j)
        {
            if (array[j] > array[j + 1])
            {
                int temp = array[j];
                array[j] = array[j + 1];
                array[j + 1] = temp;
            }
        }
    }
}

int main()
{
    int n, search, result, array[100];

    printf("Enter the number of elements: ");
    scanf("%d", &n);

    for (int i = 0; i < n; i++)
    {
        printf("Enter element %d: ", i + 1);
        scanf("%d", &array[i]);
    }

    printf("Enter the element to search: ");
    scanf("%d", &search);

    bubbleSort(array, n);

    result = binarySearch(array, n, search);

    if (result != -1)
        printf("Element %d found at index %d\n",
search, result);
    else
        printf("Element not found\n");

    return 0;
}
```

4. FRACTIONAL KNAPSACK

```c
#include <stdio.h>
struct knap
{
   float w, c, p, f;
   int item;
};

void swap(struct knap *a, struct knap *b)
{
   struct knap temp = *a;
   *a = *b;
   *b = temp;
}

void bubbleSort(struct knap arr[], int n)
{
   for (int i = 0; i < n - 1; i++)
   {
      for (int j = 0; j < n - i - 1; j++)
      {
         if (arr[j].p < arr[j + 1].p)
         {
            swap(&arr[j], &arr[j + 1]);
         }
      }
   }
}

int main()
{
   struct knap a[10];
   int n;
   float k, tp = 0;

   printf("Enter the number of items: ");
   scanf("%d", &n);

   for (int i = 0; i < n; i++)
   {
      printf("Enter the weight of item %d: ", i + 1);
      scanf("%f", &a[i].w);
      printf("Enter the cost of item %d: ", i + 1);
      scanf("%f", &a[i].c);
      a[i].p = a[i].c / a[i].w;
      a[i].f = 0;
      a[i].item = i + 1;
   }

   printf("\nItems before sorting:\n");
printf("Item\tWeight\tCost\tProfit\n");
   for (int i = 0; i < n; i++)
   {
      printf("%d\t%.2f\t%.2f\t%.2f\n", a[i].item, a[i].w, a[i].c, a[i].p);
   }

   bubbleSort(a, n);

   printf("\nItems after sorting:\n");

printf("Item\tWeight\tCost\tProfit\n");

for (int i = 0; i < n; i++)
  {
   printf("%d\t%.2f\t%.2f\t%.2f\n", a[i].item, [i].w, a[i].c, a[i].p);
   }

   printf("\nEnter the knapsack limit: ");
   scanf("%f", &k);

   for (int i = 0; i < n && k > 0; i++)
   {
      if (a[i].w <= k)
      {
         a[i].f = 1;
         tp += a[i].c;
         k -= a[i].w;
      }
      else
      {
         a[i].f = k / a[i].w;
         tp += a[i].c * a[i].f;
         k = 0;
      }
   }

   printf("\nTotal profit: %.2f\n", tp);
   printf("Items selected:\n");
   printf("Item\tFraction\n");
   for (int i = 0; i < n; i++)
      printf("%d\t%.2f\n", a[i].item, a[i].f);

   return 0;
}
```

------------------------ KNAPSACK END ---------------------------

5. JOB SEQUENCING

```c
#include <stdio.h>
struct Job
{
   int id, deadline, profit;
};

int main()
{
   int n, i, j, time_slots, total_profit = 0;
   printf("Enter the number of jobs: ");
   scanf("%d", &n);

   struct Job jobs[n];

   for (i = 0; i < n; i++)
   {
      jobs[i].id = i + 1;
      printf("Enter profit for job %d: ", jobs[i].id);
      scanf("%d", &jobs[i].profit);
      printf("Enter deadline for job %d: ", jobs[i].id);
      scanf("%d", &jobs[i].deadline);
   }

   time_slots = 0;
   for (i = 0; i < n; i++)
   {
      if (jobs[i].deadline > time_slots)
      {
         time_slots = jobs[i].deadline;
      }
   }

    int scheduled[time_slots];
   for (i = 0; i < time_slots; i++)
      scheduled[i] = -1;
  for (i = 0; i < n; i++)
   {
      for (j = jobs[i].deadline - 1; j >= 0; j--)
      {
if (scheduled[j] == -1)
         {
            scheduled[j] = i;
            total_profit += jobs[i].profit;
            break;
         }
      }
   }

   printf("\n>> Job Sequence: ");
   for (i = 0; i < time_slots; i++)
   {
      if (scheduled[i] != -1)
      {
         printf(" %d ,", jobs[scheduled[i]].id);
      }
   }
   printf("\b\b  ");

   printf("\n>> Maximum profit: %d\n", total_profit);

   return 0;
}
```

-------------------- JOB SEQUENCING END ----------------------

6. PRIM'S ALGORITHM

```c
#include <stdio.h>
#include <limits.h>

#define MAX_VERTICES 100

int minKey(int key[], int mstSet[], int V)
{
   int min = INT_MAX, min_index, v;

   for (v = 0; v < V; v++)
      if (mstSet[v] == 0 && key[v] < min)
         min = key[v], min_index = v;

   return min_index;
}

void printMST(int parent[], int V, int graph[MAX_VERTICES][MAX_VERTICES])
{
   printf("Edge   Weight\n");
   int i;
   int maxW = 0;
   for (i = 1; i < V; i++)
   {
      printf("%d - %d    %d \n", parent[i], i, graph[i][parent[i]]);
      maxW += graph[i][parent[i]];
   }
   printf("\nMaximum weight of MST is : %d", maxW);
}

void primMST(int V, int graph[MAX_VERTICES][MAX_VERTICES])
{
   int parent[V];
   int key[V];
   int mstSet[V], i;

   for (i = 0; i < V; i++)
      key[i] = INT_MAX, mstSet[i] = 0;

   key[0] = 0;
   parent[0] = -1;
int count;
    for (count = 0; count < V - 1; count++)
    {
        int u = minKey(key, mstSet, V), v;

        mstSet[u] = 1;

        for (v = 0; v < V; v++)
            if (graph[u][v] && mstSet[v] == 0 && graph[u][v] < key[v])
                parent[v] = u, key[v] = graph[u][v];
    }
    printMST(parent, V, graph);
}

int main()
{
    int V;

    printf("Enter the number of vertices: ");
    scanf("%d", &V);

    int graph[MAX_VERTICES][MAX_VERTICES];

    printf("Enter the adjacency matrix of the graph:\n");
    int i, j;
    for (i = 0; i < V; i++)
    {
        printf("Enter row %d: ", i + 1);
        for (j = 0; j < V; j++)
        {
            scanf("%d", &graph[i][j]);
        }
    }

    primMST(V, graph);

    return 0;
}
```

-------------------- PRIM's ALGO END --------------------

7. BFS (Breadth First Search)

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

int queue[100], f = 0, r = 0, size = 0;

void enqueue(int val)
{
    if (r == size)
    {
        printf("This Queue is full\n");
    }
    else
    {
        r++;
        queue[r] = val;
    }
}

int dequeue()
{
    int a = -1;
    if (f == r)
    {
        printf("This Queue is empty\n");
    }
    else
    {
        f++;
        a = queue[f];
    }
    return a;
}

int isEmpty()
{
    if (r == f)
    {
        return 1;
    }
    return 0;
}

int main()
{
    int n, visited[100];
    int G[100][100];

    printf("Enter the number of vertices: ");
    scanf("%d", &n);

    size = n; // setting the size of the queue

    printf("\nEnter the adjacency matrix:");
    for (int i = 0; i < n; i++)
    {
        visited[i] = 0;
        printf("\nEnter the row %d: \n", i);
        for (int j = 0; j < n; j++)
        {
            scanf("%d", &G[i][j]);
        }
    }

    printf("\nThe adjacency matrix is: \n");
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            printf("%d ", G[i][j]);
        }
        printf("\n");
    }

    int i = 0;
    printf("Enter the starting node: ");
    scanf("%d", &i);

    printf("BFS Traversal: %d ", i);
    visited[i] = 1;
    enqueue(i);

    while (! isEmpty())
    {
        int node = dequeue();
        for (int j = 0; j < n; j++)
        {
            if (G[node][j] == 1 && visited[j] == 0)
            {
                printf("%d ", j);
                visited[j] = 1;
                enqueue(j);
            }
        }
    }
    return 0;
}
```

8. DFS (Depth First Search)

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

int visited[100], n, G[100][100];

void DFS(int node)
{
    printf("%d ", node);
    visited[node] = 1;

    for (int i = 0; i < n; i++)
    {
        if (G[node][i] == 1 && visited[i] == 0)
        {
            DFS(i);
        }
    }
}

int main()
{

    printf("\n\t ---- DFS (Depth First Search) ----\n\n");

    printf("Enter the number of vertices: ");
    scanf("%d", &n);

    printf("\nEnter the adjacency matrix:");
    for (int i = 0; i < n; i++)
    {
        visited[i] = 0;
        printf("\nEnter the row %d: \n", i);
        for (int j = 0; j < n; j++)
        {
            scanf("%d", &G[i][j]);
        }
    }

    printf("\nThe adjacency matrix is: \n");
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            printf("%d ", G[i][j]);
        }
        printf("\n");
    }

    int starting_node;
    printf("\nEnter the starting node: ");
    scanf("%d", &starting_node);

    printf("DFS Traversal: ");
    DFS(starting_node);

    printf("\n");
    return 0;
}
```

9. TRAVELLING SALESMAN

```
#include <stdio.h>

int matrix[25][25], visited_cities[10], limit, cost = 0;

int tsp(int c)
{
   int count, nearest_city = 999;
   int minimum = 999, temp = 0; // Initialize temp to 0
   for (count = 0; count < limit; count++)
   {
      if ((matrix[c][count] != 0) && (visited_cities[count] == 0))
      {
         if (matrix[c][count] + matrix[count][0] < minimum)
         {
            minimum = matrix[c][count] + matrix[count][0];
            temp = matrix[c][count];
            nearest_city = count;
         }
      }
   }
   if (minimum != 999)
   {
      cost = cost + temp;
   }
   return nearest_city;
}

void minimum_cost(int city)
{
   int nearest_city;
   visited_cities[city] = 1;
   printf("%d ", city + 1);
   nearest_city = tsp(city);
   if (nearest_city == 999)
   {
      nearest_city = 0;
      printf("%d", nearest_city + 1);
      cost = cost + matrix[city][nearest_city];
      return;
   }
   minimum_cost(nearest_city);
}

int main()
{
   int i, j;
   printf("Enter Total Number of Cities:\t");
   scanf("%d", &limit);

   printf("\nEnter Cost Matrix\n");
   for (i = 0; i < limit; i++)
   {
      printf("\nEnter %d Elements in Row[%d]\n", limit, i + 1);
      for (j = 0; j < limit; j++)
      {
         scanf("%d", &matrix[i][j]);
      }
      visited_cities[i] = 0;
   }
printf("\nEntered Cost Matrix\n");
   for (i = 0; i < limit; i++)
   {
      printf("\n");
      for (j = 0; j < limit; j++)
      {
         printf("%d ", matrix[i][j]);
      }
   }

   printf("\n\nPath:\t");
   minimum_cost(0);
   printf("\n\nMinimum Cost: \t");
   printf("%d\n", cost);
   return 0;
}
```