VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Analysis and Design of Algorithms

Submitted by

Aditya Ram S H (1BM22CS019)

in partial fulfillment for the award of the degree of
BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE AND ENGINEERING

B.M.S. COLLEGE OF ENGINEERING
(Autonomous Institution under VTU)
BENGALURU-560019
April-2024 to August-2024

B. M. S. College of Engineering,

Bull Temple Road, Bangalore 560019

(Affiliated to Visvesvaraya Technological University, Belgaum)

Department of Computer Science and Engineering

<u>CERTIFICATE</u>

This is to certify that the Lab work entitled “**Analysis and Design of Algorithms**” carried out by **Aditya Ram S H (1BM22CS019),** who is a bonafide student of **B.M.S. College of Engineering.** It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the academic semester April-2024 to August-2024. The Lab report has been approved as it satisfies the academic requirements in respect of an **Analysis and Design of Algorithms (23CS4PCADA)** work prescribed for the said degree.

Vikranth B.M	Dr. Jyothi S Nayak
Assistant Professor	Professor and Head
Department of CSE	Department of CSE
BMSCE, Bengaluru	BMSCE, Bengaluru

Index Sheet

Course Outcome

CO1	Analyze time complexity of Recursive and Non-recursive algorithms using asymptotic notations.
CO2	Apply various design techniques for the given problem.
CO3	Apply the knowledge of complexity classes P, NP, and NP-Complete and prove certain problems are NP-Complete
CO4	Design efficient algorithms and conduct practical experiments to solve problems.

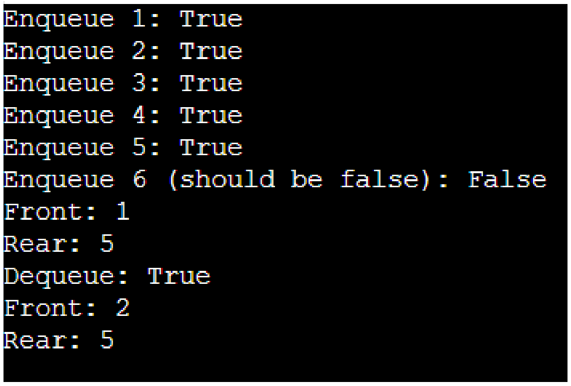
1)Leetcode exercises on Stacks, Queues, Circular Queues, Priority Queues.

Implementation of a Circular Queue in C

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

typedef struct {
    int* queue;
    int front;
    int rear;
    int size;
    int count;
} MyCircularQueue;

// Function declarations
MyCircularQueue* myCircularQueueCreate(int k);
bool myCircularQueueEnQueue(MyCircularQueue* obj, int value);
bool myCircularQueueDeQueue(MyCircularQueue* obj);
int myCircularQueueFront(MyCircularQueue* obj);
int myCircularQueueRear(MyCircularQueue* obj);
bool myCircularQueueIsEmpty(MyCircularQueue* obj);
bool myCircularQueueIsFull(MyCircularQueue* obj);
void myCircularQueueFree(MyCircularQueue* obj);

// Function definitions
MyCircularQueue* myCircularQueueCreate(int k) {
    MyCircularQueue* obj = (MyCircularQueue*)malloc(sizeof(MyCircularQueue));
    obj->queue = (int*)malloc(k * sizeof(int));
    obj->front = -1;
    obj->rear = -1;
    obj->size = k;
    obj->count = 0;
    return obj;
}

bool myCircularQueueEnQueue(MyCircularQueue* obj, int value) {
    if (myCircularQueueIsFull(obj)) return false;
    if (myCircularQueueIsEmpty(obj)) obj->front = 0;
    obj->rear = (obj->rear + 1) % obj->size;
    obj->queue[obj->rear] = value;
    obj->count++;
    return true;
}

bool myCircularQueueDeQueue(MyCircularQueue* obj) {
    if (myCircularQueueIsEmpty(obj)) return false;
    if (obj->front == obj->rear) obj->front = obj->rear = -1;
    else obj->front = (obj->front + 1) % obj->size;
    obj->count--;
    return true;
}

int myCircularQueueFront(MyCircularQueue* obj) {
    if (myCircularQueueIsEmpty(obj)) return -1;
    return obj->queue[obj->front];
}
```

```
int myCircularQueueRear(MyCircularQueue* obj) {
    if (myCircularQueueIsEmpty(obj)) return -1;
    return obj->queue[obj->rear];
}

bool myCircularQueueIsEmpty(MyCircularQueue* obj) {
    return obj->count == 0;
}

bool myCircularQueueIsFull(MyCircularQueue* obj) {
    return obj->count == obj->size;
}

void myCircularQueueFree(MyCircularQueue* obj) {
    free(obj->queue);
    free(obj);
}

// Example of usage
int main() {
    int k = 5;
    MyCircularQueue* obj = myCircularQueueCreate(k);

    printf("Enqueue 1: %s\n", myCircularQueueEnQueue(obj, 1) ? "True" : "False");
    printf("Enqueue 2: %s\n", myCircularQueueEnQueue(obj, 2) ? "True" : "False");
    printf("Enqueue 3: %s\n", myCircularQueueEnQueue(obj, 3) ? "True" : "False");
    printf("Enqueue 4: %s\n", myCircularQueueEnQueue(obj, 4) ? "True" : "False");
    printf("Enqueue 5: %s\n", myCircularQueueEnQueue(obj, 5) ? "True" : "False");
    printf("Enqueue 6 (should be false): %s\n", myCircularQueueEnQueue(obj, 6) ? "True" : "False");

    printf("Front: %d\n", myCircularQueueFront(obj));
    printf("Rear: %d\n", myCircularQueueRear(obj));

    printf("Dequeue: %s\n", myCircularQueueDeQueue(obj) ? "True" : "False");
    printf("Front: %d\n", myCircularQueueFront(obj));
    printf("Rear: %d\n", myCircularQueueRear(obj));

    myCircularQueueFree(obj);

    return 0;
}
```

OUTPUT:



```
Enqueue 1: True
Enqueue 2: True
Enqueue 3: True
Enqueue 4: True
Enqueue 5: True
Enqueue 6 (should be false): False
Front: 1
Rear: 5
Dequeue: True
Front: 2
Rear: 5
```

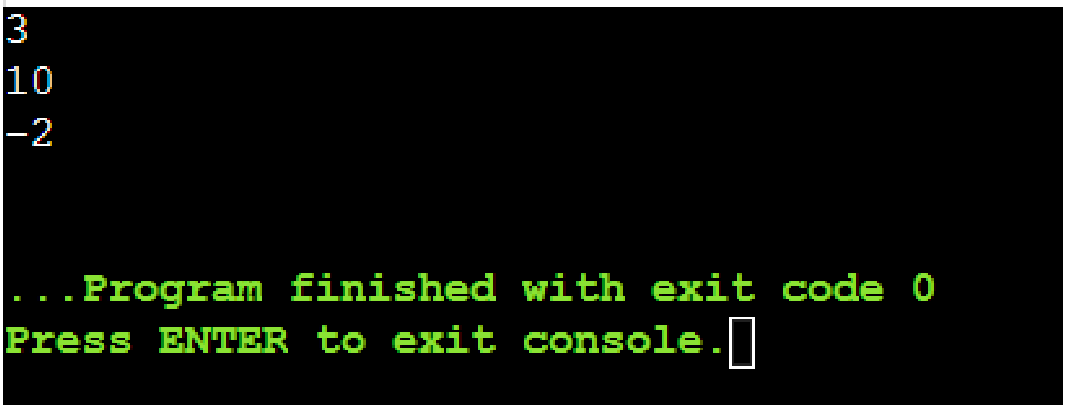
1) **Leetcode exercises on Stacks, Queues, Circular Queues, Priority Queues.**

Given a circular integer array nums of length n, return the maximum possible sum of a non-empty subarray of nums

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

// Function to find the maximum of two integers
int max(int a, int b) {
   return (a > b) ? a : b;
}

// Function to find the minimum of two integers
int min(int a, int b) {
   return (a < b) ? a : b;
}

// Function to find the maximum subarray sum using Kadane's algorithm
int kadane(int nums[], int size, int isMin) {
   int result = isMin ? INT_MAX : INT_MIN;
   int current_sum = 0;
   for (int i = 0; i < size; i++) {
      if (isMin) {
         current_sum = min(nums[i], current_sum + nums[i]);
         result = min(result, current_sum);
      } else {
         current_sum = max(nums[i], current_sum + nums[i]);
         result = max(result, current_sum);
      }
```

```
    }
    return result;
}

// Function to find the maximum circular subarray sum
int maxSubarraySumCircular(int* nums, int numsSize) {
    int max_sum_non_circular = kadane(nums, numsSize, 0);
    int min_sum_non_circular = kadane(nums, numsSize, 1); // For finding the minimum sum
    int total_sum = 0;

    // Calculate total sum of the array
    for (int i = 0; i < numsSize; i++) {
        total_sum += nums[i];
    }

    // Find the maximum circular subarray sum
    int max_sum_circular = total_sum - min_sum_non_circular;
    if (min_sum_non_circular != total_sum) {
        max_sum_non_circular = max(max_sum_non_circular, max_sum_circular);
    }

    return max_sum_non_circular;
}

int main() {
    int nums1[] = {1, -2, 3, -2};
    int nums2[] = {5, -3, 5};
    int nums3[] = {-3, -2, -3};
    int size1 = sizeof(nums1) / sizeof(nums1[0]);
    int size2 = sizeof(nums2) / sizeof(nums2[0]);
    int size3 = sizeof(nums3) / sizeof(nums3[0]);

    printf("%d\n", maxSubarraySumCircular(nums1, size1)); // Output: 3
    printf("%d\n", maxSubarraySumCircular(nums2, size2)); // Output: 10
    printf("%d\n", maxSubarraySumCircular(nums3, size3)); // Output: -2

    return 0;
}
```

OUTPUT:

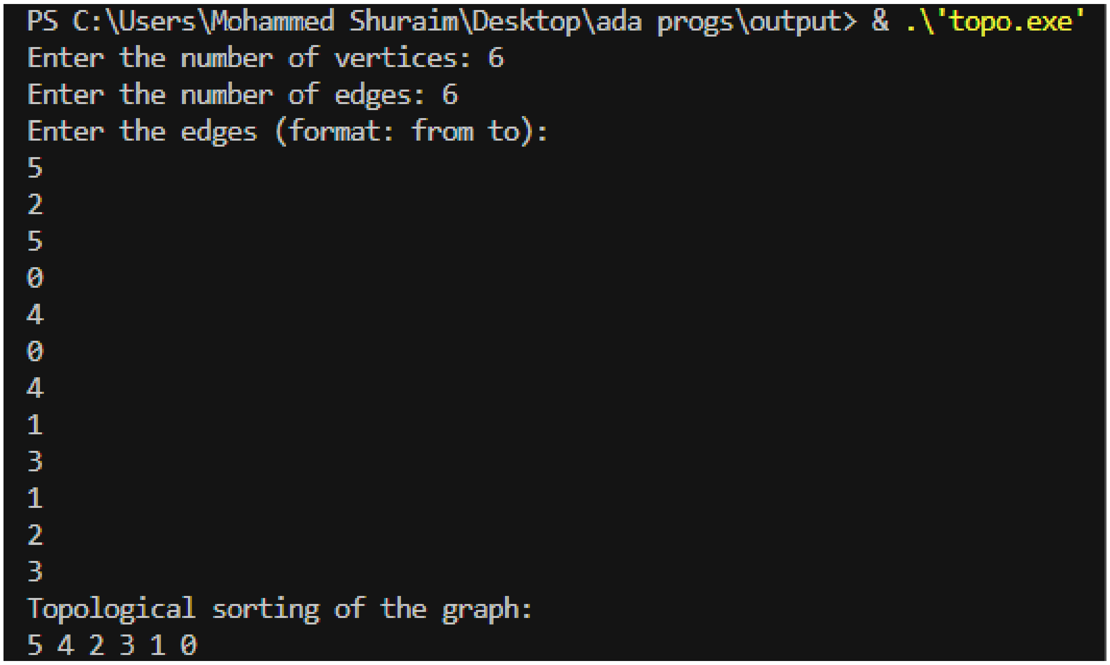


```
3
10
-2


...Program finished with exit code 0
Press ENTER to exit console.
```

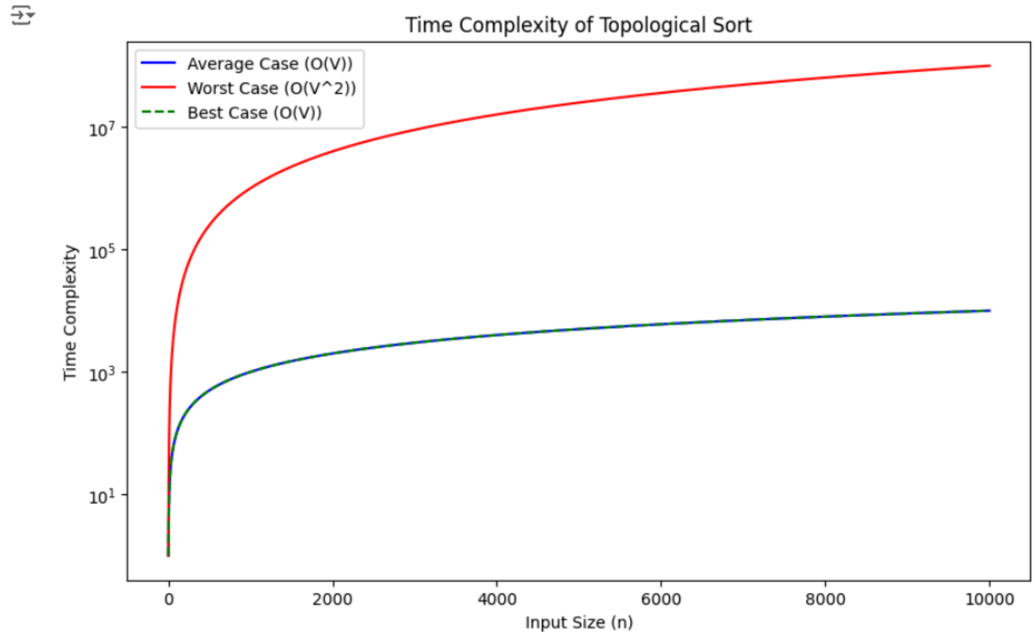
2)Write a program to obtain the Topological ordering of vertices in a given digraph.

```
#include <stdio.h>

#include <stdbool.h>


#define MAX_V 106


void topoutil(int v, int adj[][MAX_V], bool vis[], int *st, int *top) {

    vis[v] = true;


    for (int i = 0; i < MAX_V; i++) {

        if (adj[v][i] && !vis[i]) {

            topoutil(i, adj, vis, st, top);

        }

    }


    st[(*top)++] = v;

}



void topo(int adj[][MAX_V], int V) {

    int st[MAX_V];

    bool vis[MAX_V] = { false };

    int top = 0;


    for (int i = 0; i < V; i++) {

        if (!vis[i]) {

            topoutil(i, adj, vis, st, &top);

        }
    }

    printf("Topological sorting of the graph:\n");
    for (int i = top - 1; i >= 0; i--) {
        printf("%d ", st[i]);
    }
    printf("\n");
}



int main() {
    int V, E;

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

    int adj[MAX_V][MAX_V] = {0}; // Adjacency matrix initialized to 0
    printf("Enter the number of edges: ");
    scanf("%d", &E);
    printf("Enter the edges (format: from to):\n");
    for (int i = 0; i < E; i++) {
        int from, to;        scanf("%d %d", &from, &to);
        adj[from][to] = 1;
    }
    topo(adj, V);
    return 0;
}
```

OUTPUT:



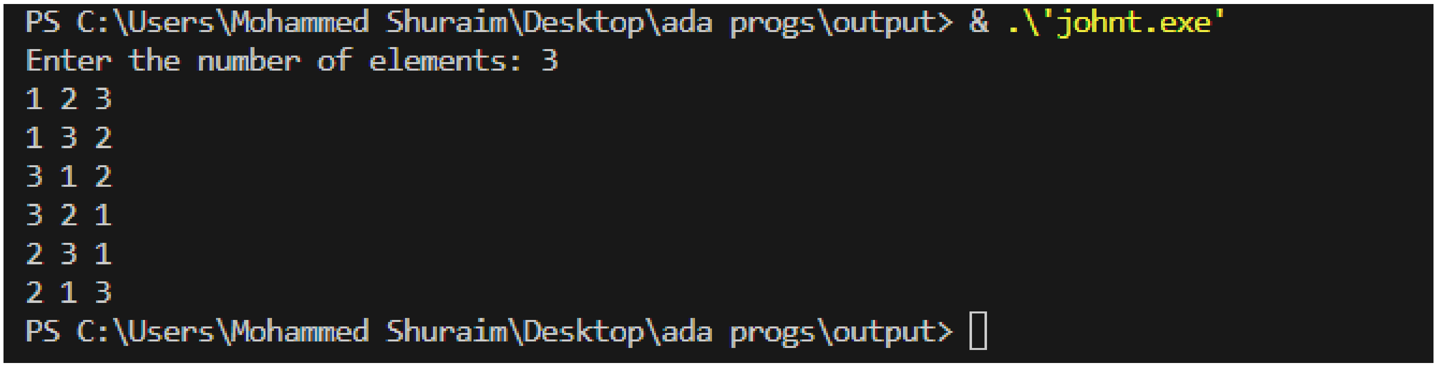
```
PS C:\Users\Mohammed Shuraim\Desktop\ada progs\output> & .\'topo.exe'
Enter the number of vertices: 6
Enter the number of edges: 6
Enter the edges (format: from to):
5
2
5
0
4
0
4
1
3
1
2
3
Topological sorting of the graph:
5 4 2 3 1 0
```



3)Implement Johnson Trotter algorithm to generate permutations.

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

#define LEFT_TO_RIGHT 1
#define RIGHT_TO_LEFT 0

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

void printPermutation(int* perm, int n) {
    for (int i = 0; i < n; i++) {
        printf("%d ", perm[i]);
    }
    printf("\n");
}

int findMobile(int* perm, int* dir, int n) {
    int mobile_prev = 0, mobile = 0;
    for (int i = 0; i < n; i++) {
        if (dir[perm[i] - 1] == RIGHT_TO_LEFT && i != 0) {
            if (perm[i] > perm[i - 1] && perm[i] > mobile_prev) {
                mobile = perm[i];
                mobile_prev = mobile;
            }
        }
        if (dir[perm[i] - 1] == LEFT_TO_RIGHT && i != n - 1) {
            if (perm[i] > perm[i + 1] && perm[i] > mobile_prev) {
                mobile = perm[i];
                mobile_prev = mobile;
            }
        }
    }
    return mobile;
}

int findPosition(int* perm, int n, int mobile) {
    for (int i = 0; i < n; i++) {
        if (perm[i] == mobile) {
            return i + 1;
        }
    }
    return -1;
}

void changeDirection(int* perm, int* dir, int n, int mobile) {
    for (int i = 0; i < n; i++) {
        if (perm[i] > mobile) {
            if (dir[perm[i] - 1] == LEFT_TO_RIGHT) {
                dir[perm[i] - 1] = RIGHT_TO_LEFT;
            } else if (dir[perm[i] - 1] == RIGHT_TO_LEFT) {
                dir[perm[i] - 1] = LEFT_TO_RIGHT;
            }
        }
    }
}

void johnsonTrotter(int n) {

    int* perm = (int*)malloc(n * sizeof(int));
    int* dir = (int*)malloc(n * sizeof(int));

    for (int i = 0; i < n; i++) {
        perm[i] = i + 1;
        dir[i] = RIGHT_TO_LEFT;
    }

    printPermutation(perm, n);

    while (1) {

        int mobile = findMobile(perm, dir, n);
        if (mobile == 0) {
            break;
        }

        int pos = findPosition(perm, n, mobile);
        if (dir[perm[pos - 1] - 1] == RIGHT_TO_LEFT) {
            swap(&perm[pos - 1], &perm[pos - 2]);
        } else if (dir[perm[pos - 1] - 1] == LEFT_TO_RIGHT) {
            swap(&perm[pos], &perm[pos - 1]);
        }

        // Print the current permutation
        printPermutation(perm, n);

        // Reverse the direction of all elements larger than the largest mobile element
        changeDirection(perm, dir, n, mobile);
    }
    // Free allocated memory
    free(perm);
    free(dir);
}
int main() {
    int n;
    printf("Enter the number of elements: ");
    scanf("%d", &n);
    johnsonTrotter(n);
    return 0;
}
```

OUTPUT:



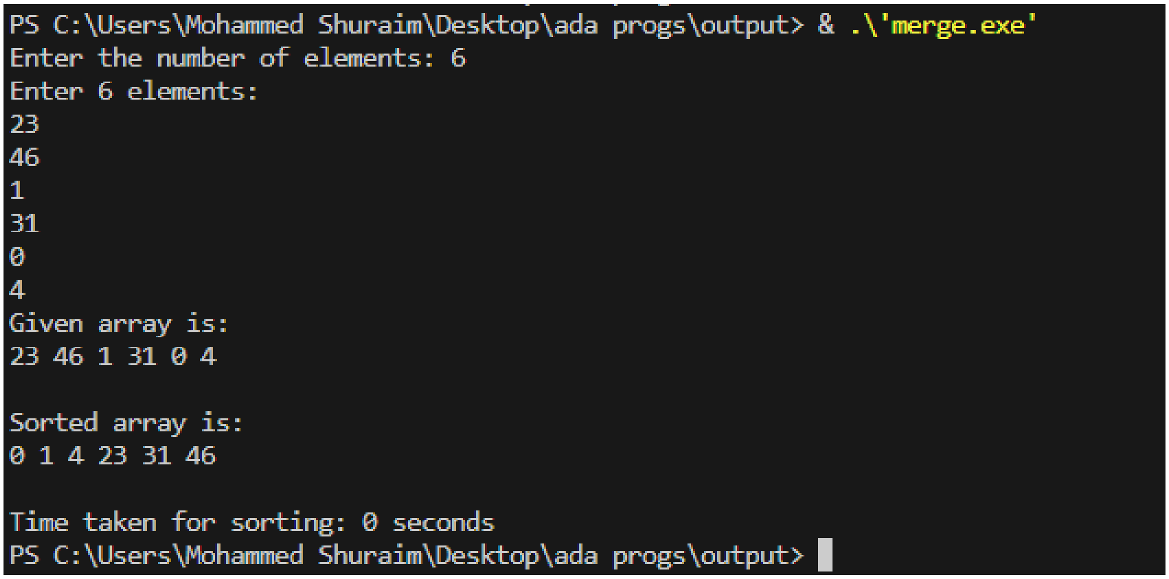
```
PS C:\Users\Mohammed Shuraim\Desktop\ada progs\output> & .\'johnt.exe'
Enter the number of elements: 3
1 2 3
1 3 2
3 1 2
3 2 1
2 3 1
2 1 3
PS C:\Users\Mohammed Shuraim\Desktop\ada progs\output> 
```

4) Sort a given set of N integer elements using the Merge Sort technique and compute its time taken. Run the program for different values of N and record the time taken to sort.

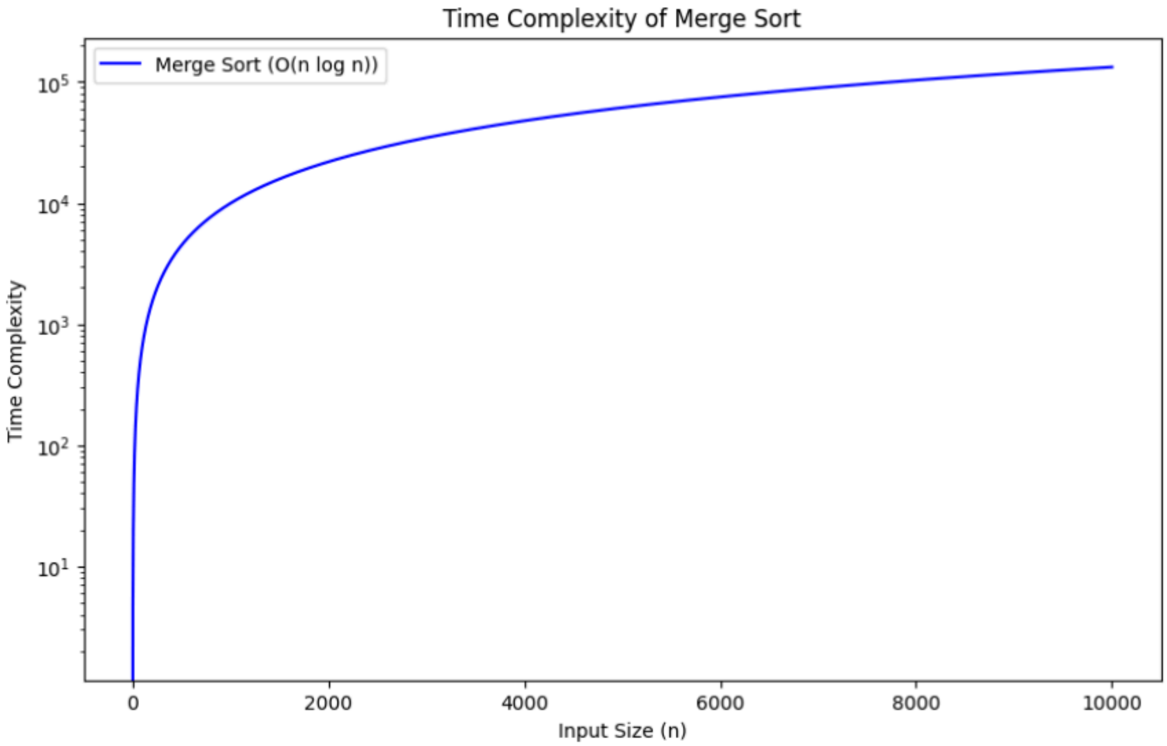
```
#include <stdio.h>

#include <stdlib.h>

#include <time.h>


void merge(int arr[], int l, int m, int r) {

    int i, j, k;

    int n1 = m - l + 1;

    int n2 = r - m;


    int L[n1], R[n2];


    for (i = 0; i < n1; i++)

        L[i] = arr[l + i];

    for (j = 0; j < n2; j++)

        R[j] = arr[m + 1 + j];


    i = 0;

    j = 0;

    k = l;

    while (i < n1 && j < n2) {

        if (L[i] <= R[j]) {

            arr[k] = L[i];

            i++;

        } else {

            arr[k] = R[j];

            j++;
        }
        k++;
    }

    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }

    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }
}

void mergeSort(int arr[], int l, int r) {
    if (l < r) {
        int m = l + (r - l) / 2;
        mergeSort(arr, l, m);
        mergeSort(arr, m + 1, r);
        merge(arr, l, m, r);
    }
}

void printArray(int A[], int size) {
    for (int i = 0; i < size; i++)
```

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



int main() {
    int n;
    printf("Enter the number of elements: ");
    scanf("%d", &n);
    int *arr = (int *)malloc(n * sizeof(int));
    printf("Enter %d elements:\n", n);
    for (int i = 0; i < n; i++)
        scanf("%d", &arr[i]);
    printf("Given array is:\n");
    printArray(arr, n);
    clock_t start, end;
    double cpu_time_used;
    start = clock();
    mergeSort(arr, 0, n - 1);
    end = clock();
    cpu_time_used = ((double)(end - start)) / CLOCKS_PER_SEC;
    printf("\nSorted array is:\n");
    printArray(arr, n);
    printf("\nTime taken for sorting: %d seconds\n", cpu_time_used);
    free(arr);
    return 0;
}
```

OUTPUT:



```
PS C:\Users\Mohammed Shuraim\Desktop\ada progs\output> & .\'merge.exe'
Enter the number of elements: 6
Enter 6 elements:
23
46
1
31
0
4
Given array is:
23 46 1 31 0 4

Sorted array is:
0 1 4 23 31 46

Time taken for sorting: 0 seconds
PS C:\Users\Mohammed Shuraim\Desktop\ada progs\output>
```

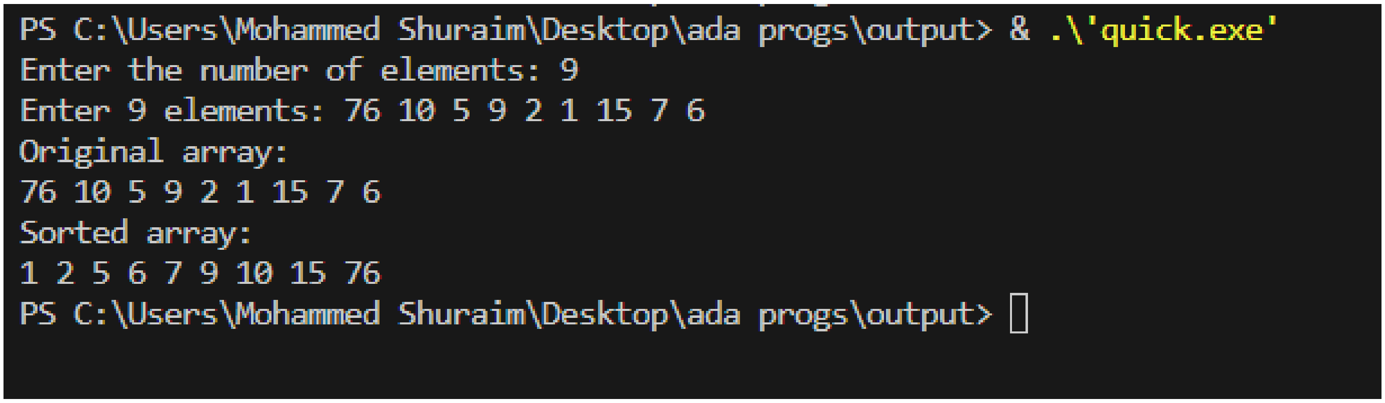


5) Sort a given set of N integer elements using Quick Sort technique and compute its time taken.

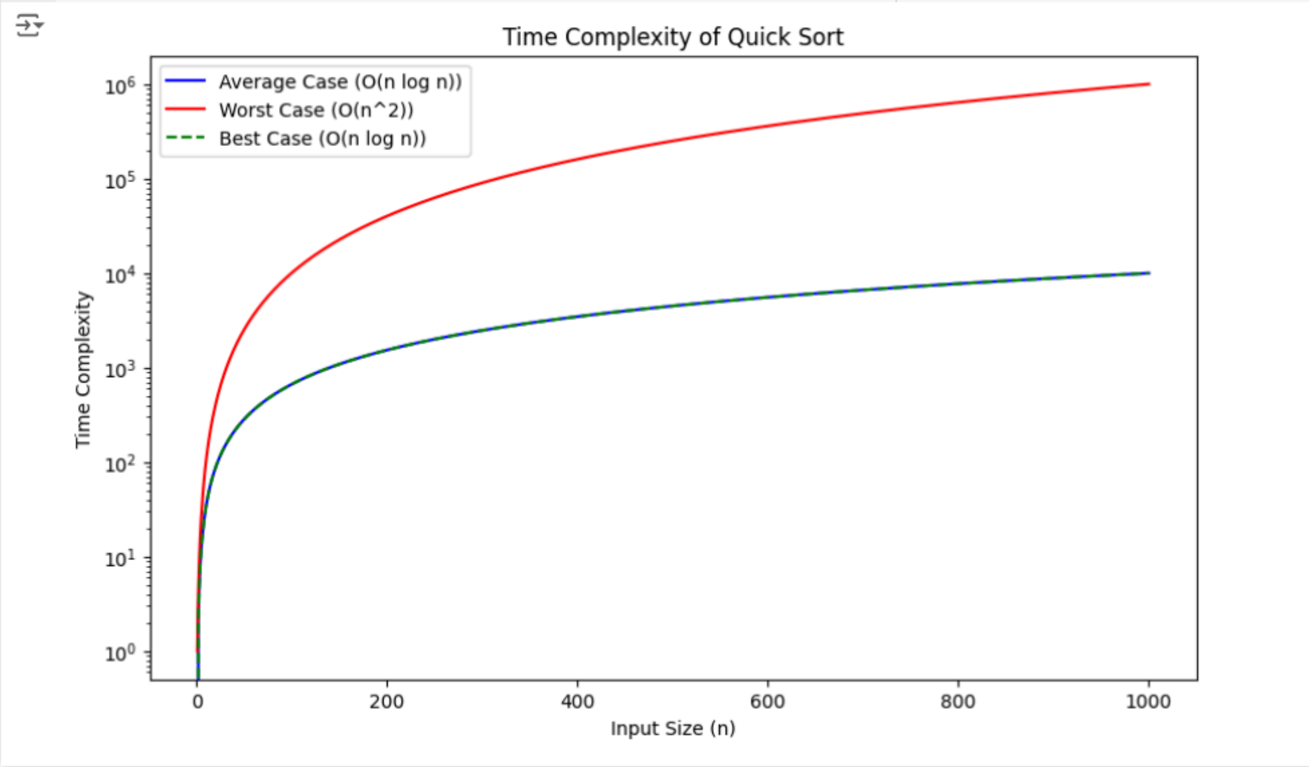
```
#include <stdio.h>


void print_array(int array[], int size) {
   for (int i = 0; i < size; i++) {
      printf("%d ", array[i]);
   }
   printf("\n");
}
int partition(int array[], int low, int high) {
   int pivot = array[low];
   int start = low + 1;
   int end = high;
   while (1) {
      while (start <= end && array[start] <= pivot) {
         start++;
      }
      while (array[end] >= pivot && end >= start) {
         end--;
      }
      if (end < start) {
         break;
      } else {
         int temp = array[start];
         array[start] = array[end];
         array[end] = temp;
      }
   }
```

```
    int temp = array[low];
    array[low] = array[end];
    array[end] = temp;
    return end;
}
void quick_sort(int array[], int low, int high) {
    if (low < high) {
        int p = partition(array, low, high);
        quick_sort(array, low, p - 1);
        quick_sort(array, p + 1, high);
    }
}
int main() {
    int n;
    printf("Enter the number of elements: ");
    scanf("%d", &n);
    int arr[n];
    printf("Enter %d elements: ", n);
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }
    printf("Original array:\n");
    print_array(arr, n);
    quick_sort(arr, 0, n - 1);
    printf("Sorted array:\n");
    print_array(arr, n);
    return 0;
}
```

OUTPUT:



```
PS C:\Users\Mohammed Shuraim\Desktop\ada progs\output> & .\'quick.exe'
Enter the number of elements: 9
Enter 9 elements: 76 10 5 9 2 1 15 7 6
Original array:
76 10 5 9 2 1 15 7 6
Sorted array:
1 2 5 6 7 9 10 15 76
PS C:\Users\Mohammed Shuraim\Desktop\ada progs\output> 
```



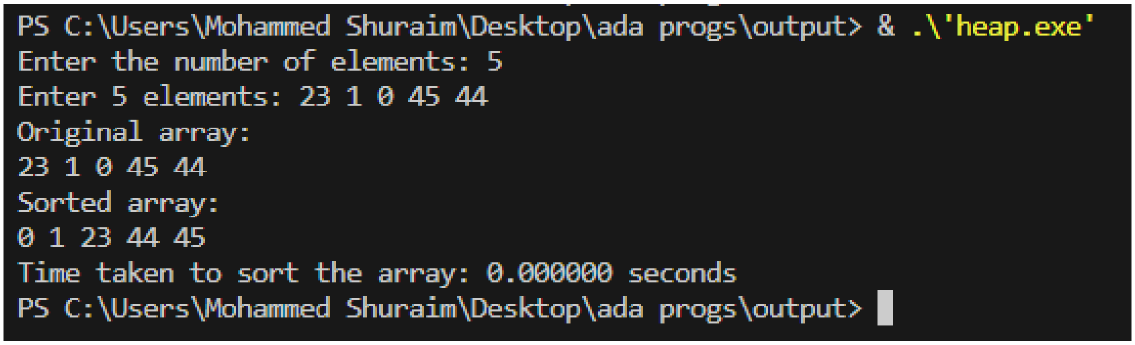
6) Sort a given set of N integer elements using Heap Sort technique and compute its time taken.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
void Insert(int A[], int n) {
    int i = n, temp;
    temp = A[i];
    while (i > 1 && temp > A[i / 2]) {
        A[i] = A[i / 2];
        i = i / 2;
    }
    A[i] = temp;
}
int Delete(int A[], int n) {
    int i, j, x, temp, val;
    val = A[1];
    x = A[n];
    A[1] = A[n];
    A[n] = val;
    i = 1;
    j = i * 2;
    while (j <= n - 1) {
        if (j < n - 1 && A[j + 1] > A[j])
            j = j + 1;
        if (A[i] < A[j]) {
            temp = A[i];
            A[i] = A[j];
            A[j] = temp;
            i = j;
            j = 2 * j;
        } else {
            break;
        }
    }
    return val;
}

void heap_sort(int A[], int n) {
    for (int i = 2; i <= n; i++) {
        Insert(A, i);
    }
    for (int i = n; i > 1; i--) {
        Delete(A, i);
    }
}

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

    int *A = (int *)malloc((n + 1) * sizeof(int));
    if (A == NULL) {
        printf("Memory allocation failed!\n");
        return 1;
    }
```

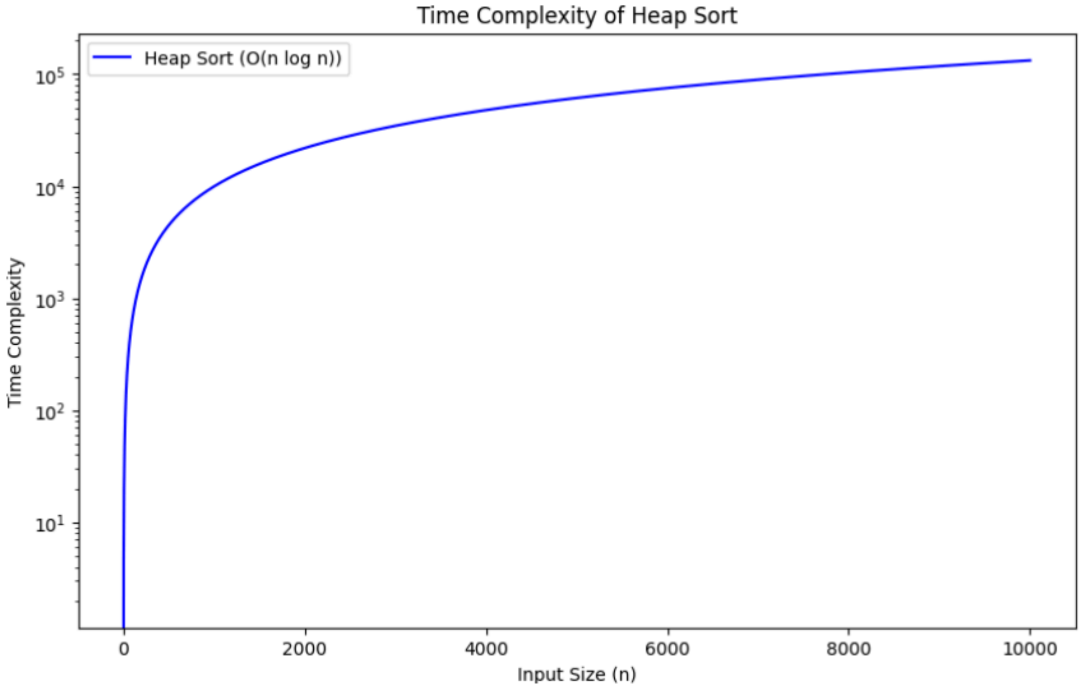
```
printf("Enter %d elements: ", n);
for (int i = 1; i <= n; i++) {
    scanf("%d", &A[i]);
}

printf("Original array:\n");
for (int i = 1; i <= n; i++) {
    printf("%d ", A[i]);
}
printf("\n");

clock_t start, end;
double cpu_time_used;

start = clock();
heap_sort(A, n);
end = clock();

cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;

printf("Sorted array:\n");
for (int i = 1; i <= n; i++) {
    printf("%d ", A[i]);
}
printf("\n");

printf("Time taken to sort the array: %f seconds\n", cpu_time_used);
```

```
    free(A);
    return 0;
}
```

OUTPUT:



```
PS C:\Users\Mohammed Shuraim\Desktop\ada progs\output> & .\'heap.exe'
Enter the number of elements: 5
Enter 5 elements: 23 1 0 45 44
Original array:
23 1 0 45 44
Sorted array:
0 1 23 44 45
Time taken to sort the array: 0.000000 seconds
PS C:\Users\Mohammed Shuraim\Desktop\ada progs\output>
```



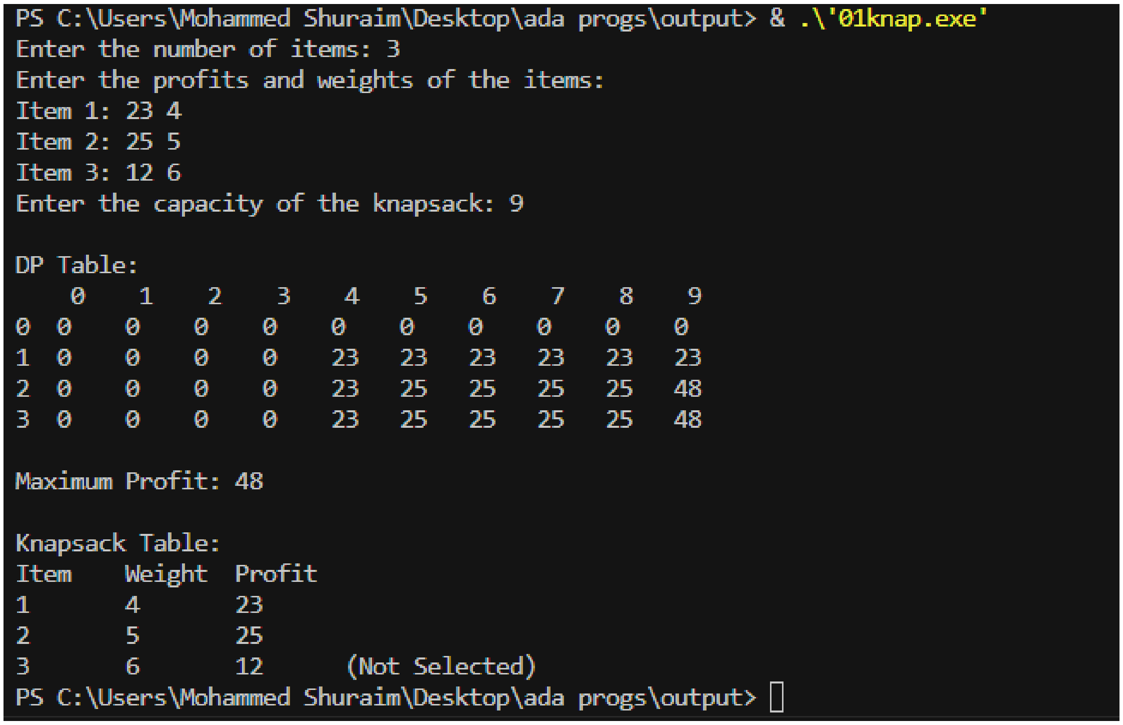
7) Implement 0/1 Knapsack problem using dynamic programming.

```
#include <stdio.h>

#include <stdbool.h>


// A utility function that returns maximum of two integers

int max(int a, int b) { return (a > b) ? a : b; }




int knapSack(int W, int wt[], int val[], int n, bool selected[])

{


    int dp[n + 1][W + 1];


    // Build table dp[][] in bottom up manner

    for (int i = 0; i <= n; i++) {

        for (int w = 0; w <= W; w++) {

            if (i == 0 || w == 0)

                dp[i][w] = 0;

            else if (wt[i - 1] <= w)

                dp[i][w] = max(val[i - 1] + dp[i - 1][w - wt[i - 1]], dp[i - 1][w]);

            else

                dp[i][w] = dp[i - 1][w];

        }

    }
```

```
// dp[n][W] contains the maximum profit
int maxProfit = dp[n][W];

// Finding the selected items using backtracking
int remainingWeight = W;
for (int i = n; i > 0 && maxProfit > 0; i--) {
    if (maxProfit != dp[i - 1][remainingWeight]) {
        selected[i - 1] = true;
        maxProfit -= val[i - 1];
        remainingWeight -= wt[i - 1];
    }
}

// Print the dp table
printf("\nDP Table:\n");
printf("    ");
for (int w = 0; w <= W; w++) {
    printf("%-5d", w);
}
printf("\n");
for (int i = 0; i <= n; i++) {
    printf("%-3d", i);
    for (int w = 0; w <= W; w++) {
        printf("%-5d", dp[i][w]);
    }
```

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

    return dp[n][W];
}

void printKnapsackTable(int W, int wt[], int val[], int n, bool selected[])
{
    printf("\nKnapsack Table:\n");
    printf("Item\tWeight\tProfit\n");
    for (int i = 0; i < n; i++) {
        if (selected[i])
            printf("%d\t%d\t%d\n", i + 1, wt[i], val[i]);
        else
            printf("%d\t%d\t%d\t(Not Selected)\n", i + 1, wt[i], val[i]);
    }
}

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

    int profit[n], weight[n];
    bool selected[n];

    printf("Enter the profits and weights of the items:\n");
    for (int i = 0; i < n; i++) {
        printf("Item %d: ", i + 1);
        scanf("%d %d", &profit[i], &weight[i]);
        selected[i] = false;
    }

    int capacity;
    printf("Enter the capacity of the knapsack: ");
    scanf("%d", &capacity);

    int maxProfit = knapSack(capacity, weight, profit, n, selected);

    printf("\nMaximum Profit: %d\n", maxProfit);

    printKnapsackTable(capacity, weight, profit, n, selected);

    return 0;
}
```

OUTPUT:

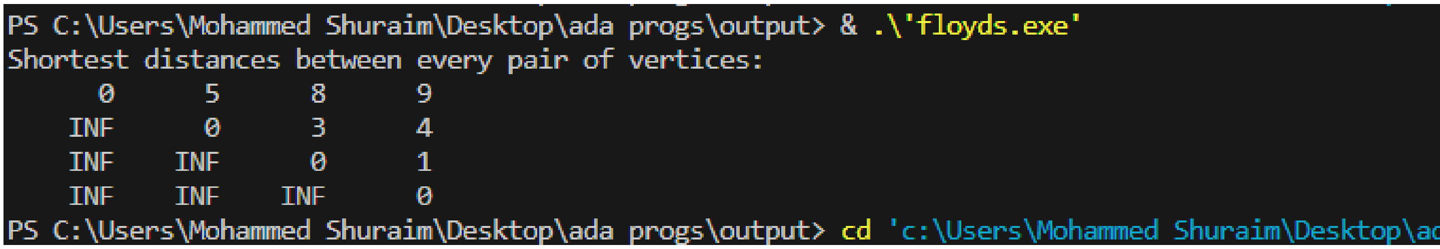


```
PS C:\Users\Mohammed Shuraim\Desktop\ada progs\output> & .\'01knap.exe'
Enter the number of items: 3
Enter the profits and weights of the items:
Item 1: 23 4
Item 2: 25 5
Item 3: 12 6
Enter the capacity of the knapsack: 9

DP Table:
    0    1    2    3    4    5    6    7    8    9
0  0    0    0    0    0    0    0    0    0    0
1  0    0    0    0    23   23   23   23   23   23
2  0    0    0    0    23   25   25   25   25   48
3  0    0    0    0    23   25   25   25   25   48

Maximum Profit: 48

Knapsack Table:
Item    Weight  Profit
1       4       23
2       5       25
3       6       12      (Not Selected)
PS C:\Users\Mohammed Shuraim\Desktop\ada progs\output>
```

8) Implement All Pair Shortest paths problem using Floyd’s algorithm.

```
#include <stdio.h>


#define INF 99999
#define V 4  // Number of vertices
void floydWarshall(int graph[][V]) {
    int dist[V][V];
    int i, j, k;
  for (i = 0; i < V; i++) {
        for (j = 0; j < V; j++) {
            dist[i][j] = graph[i][j];
        }
    }

    // Applying Floyd-Warshall algorithm to find shortest paths
    for (k = 0; k < V; k++) {

        for (i = 0; i < V; i++) {
            // Pick all vertices as destination for the above picked source
            for (j = 0; j < V; j++) {
                if (dist[i][k] != INF && dist[k][j] != INF && dist[i][k] + dist[k][j] < dist[i][j]) {
                    dist[i][j] = dist[i][k] + dist[k][j];
                }
            }
        }
    }

    printf("Shortest distances between every pair of vertices:\n");
    for (i = 0; i < V; i++) {
        for (j = 0; j < V; j++) {
            if (dist[i][j] == INF) {
                printf("%7s", "INF");
            } else {
                printf("%7d", dist[i][j]);
            }
        }
        printf("\n");
    }
}

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;

}
```

OUTPUT:



```
PS C:\Users\Mohammed Shuraim\Desktop\ada progs\output> & .\'floyds.exe'
Shortest distances between every pair of vertices:
      0      5      8      9
    INF      0      3      4
    INF    INF      0      1
    INF    INF    INF      0
PS C:\Users\Mohammed Shuraim\Desktop\ada progs\output> cd 'c:\Users\Mohammed Shuraim\Desktop\a
```

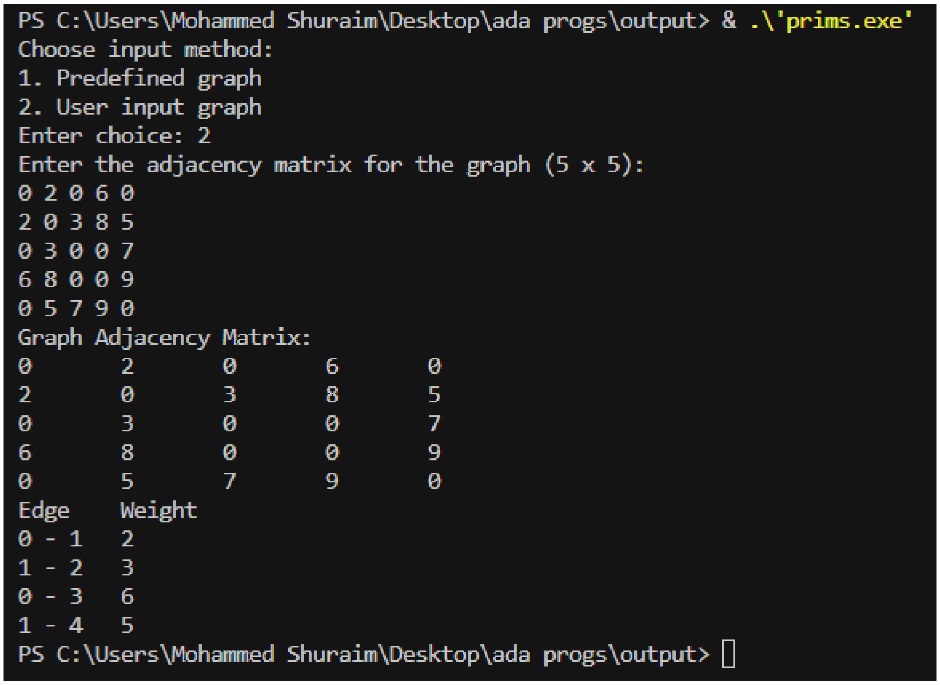
9) Find Minimum Cost Spanning Tree of a given undirected graph using Prim’s algorithm.

```
#include <limits.h>

#include <stdbool.h>

#include <stdio.h>

#define V 5 // Number of vertices in the graph

int minKey(int key[], bool mstSet[]);

void primMST(int graph[V][V]);

void printMST(int parent[], int graph[V][V]);

void printGraph(int graph[V][V]);

int minKey(int key[], bool mstSet[])

{

   int min = INT_MAX, min_index;

   for (int v = 0; v < V; v++)

      if (mstSet[v] == false && key[v] < min)

         min = key[v], min_index = v;

   return min_index;

}

void printMST(int parent[], int graph[V][V])

{

   printf("Edge \tWeight\n");

   for (int i = 1; i < V; i++)

      printf("%d - %d \t%d \n", parent[i], i, graph[i][parent[i]]);

}


void primMST(int graph[V][V])

{

   int parent[V];
```

```
    int key[V];
    bool mstSet[V];
    for (int i = 0; i < V; i++)
        key[i] = INT_MAX, mstSet[i] = false;
    key[0] = 0;
    parent[0] = -1;
    for (int count = 0; count < V - 1; count++) {
        int u = minKey(key, mstSet);
        mstSet[u] = true;
        for (int 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, graph);
}
void printGraph(int graph[V][V])
{
    printf("Graph Adjacency Matrix:\n");
    for (int i = 0; i < V; i++) {
        for (int j = 0; j < V; j++)
            printf("%d\t", graph[i][j]);
        printf("\n");
    }
}

int main()
{
    int graph[V][V];
int choice;
printf("Choose input method:\n");
printf("1. Predefined graph\n");
printf("2. User input graph\n");
printf("Enter choice: ");
scanf("%d", &choice);

switch (choice) {
case 1:

   graph[0][1] = 2;
   graph[0][3] = 6;
   graph[1][0] = 2;
   graph[1][2] = 3;
   graph[1][3] = 8;
   graph[1][4] = 5;
   graph[2][1] = 3;
   graph[2][4] = 7;
   graph[3][0] = 6;
   graph[3][1] = 8;
   graph[3][4] = 9;
   graph[4][1] = 5;
   graph[4][2] = 7;
   graph[4][3] = 9;
   break;
case 2:

   printf("Enter the adjacency matrix for the graph (%d x %d):\n", V, V);
```

```
        for (int i = 0; i < V; i++)
            for (int j = 0; j < V; j++)
                scanf("%d", &graph[i][j]);
        break;
    default:
        printf("Invalid choice.\n");
        return 1;
    }
    printGraph(graph);
    primMST(graph);
    return 0;
```

OUTPUT:



```
PS C:\Users\Mohammed Shuraim\Desktop\ada progs\output> & .\'prims.exe'
Choose input method:
1. Predefined graph
2. User input graph
Enter choice: 2
Enter the adjacency matrix for the graph (5 x 5):
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
Graph 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
Edge    Weight
0 - 1   2
1 - 2   3
0 - 3   6
1 - 4   5
PS C:\Users\Mohammed Shuraim\Desktop\ada progs\output>
```

```
}
```

9) Find Minimum Cost Spanning Tree of a given undirected graph using Kruskal’s algorithm.

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

int comparator(const void* p1, const void* p2)
{
   const int(*x)[3] = p1;
   const int(*y)[3] = p2;

   return (*x)[2] - (*y)[2];
}

void makeSet(int parent[], int rank[], int n)
{
   for (int i = 0; i < n; i++) {
      parent[i] = i;
      rank[i] = 0;
   }
}

int findParent(int parent[], int component)
{
   if (parent[component] == component)
      return component;

   return parent[component] = findParent(parent, parent[component]);
}
```

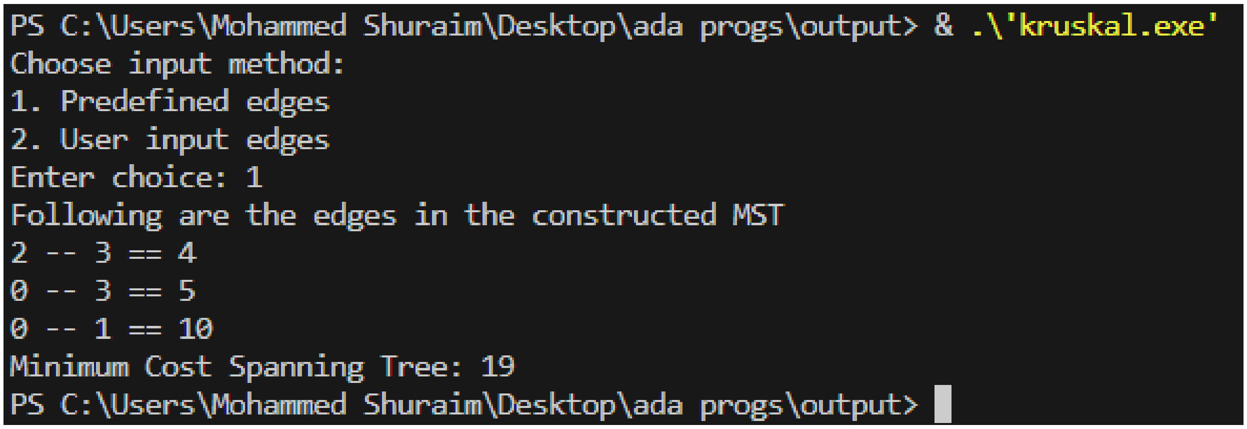
```
void unionSet(int u, int v, int parent[], int rank[], int n)
{
   u = findParent(parent, u);
   v = findParent(parent, v);

   if (rank[u] < rank[v]) {
      parent[u] = v;
   } else if (rank[u] > rank[v]) {
      parent[v] = u;
   } else {
      parent[v] = u;
      rank[u]++;
   }
}

void kruskalAlgo(int n, int edges[][3], int edgeCount)
{
   qsort(edges, edgeCount, sizeof(edges[0]), comparator);

   int parent[n];
   int rank[n];

   makeSet(parent, rank, n);

   int minCost = 0;

   printf("Following are the edges in the constructed MST\n");
```

```
    for (int i = 0; i < edgeCount; i++) {
        int v1 = findParent(parent, edges[i][0]);
        int v2 = findParent(parent, edges[i][1]);
        int wt = edges[i][2];

        if (v1 != v2) {
            unionSet(v1, v2, parent, rank, n);
            minCost += wt;
            printf("%d -- %d == %d\n", edges[i][0], edges[i][1], wt);
        }
    }

    printf("Minimum Cost Spanning Tree: %d\n", minCost);
}

int main()
{
    int V = 4; // Number of vertices
    int E = 5; // Number of edges

    printf("Choose input method:\n");
    printf("1. Predefined edges\n");
    printf("2. User input edges\n");
    printf("Enter choice: ");
    int choice;
    scanf("%d", &choice);

    int edges[E][3]; // Declare edges array here
```

```
if (choice == 2) {
    printf("Enter the number of vertices: ");
    scanf("%d", &V);
    printf("Enter the number of edges: ");
    scanf("%d", &E);
    printf("Enter the edges (src dest weight):\n");

    for (int i = 0; i < E; i++) {
        scanf("%d %d %d", &edges[i][0], &edges[i][1], &edges[i][2]);
    }

    kruskalAlgo(V, edges, E); // Pass edges to the algorithm
} else {

    int predefined_edges[][3] = {
        {0, 1, 10},
        {0, 2, 6},
        {0, 3, 5},
        {1, 3, 15},
        {2, 3, 4}
    };

    for (int i = 0; i < E; i++) {
        edges[i][0] = predefined_edges[i][0];
        edges[i][1] = predefined_edges[i][1];
        edges[i][2] = predefined_edges[i][2];
    }
```

```
        kruskalAlgo(V, edges, E); // Pass edges to the algorithm
    }

    return 0;
}
```

OUTPUT:

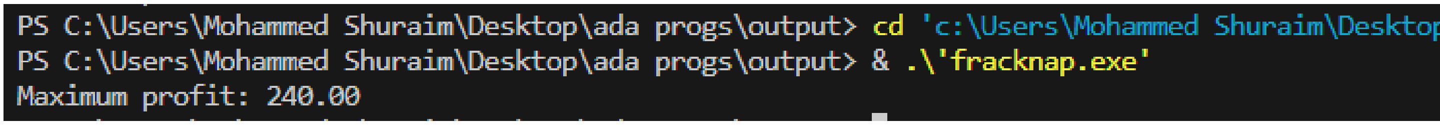


```
PS C:\Users\Mohammed Shuraim\Desktop\ada progs\output> & .\'kruskal.exe'
Choose input method:
1. Predefined edges
2. User input edges
Enter choice: 1
Following are the edges in the constructed MST
2 -- 3 == 4
0 -- 3 == 5
0 -- 1 == 10
Minimum Cost Spanning Tree: 19
PS C:\Users\Mohammed Shuraim\Desktop\ada progs\output>
```

10) Implement Fractional Knapsack using Greedy technique.

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

struct Item {
    int profit, weight;
};

int cmp(const void* a, const void* b) {
    struct Item* item1 = (struct Item*)a;
    struct Item* item2 = (struct Item*)b;

    double r1 = (double)item1->profit / item1->weight;
    double r2 = (double)item2->profit / item2->weight;

    if (r1 < r2) return 1;
    else if (r1 > r2) return -1;
    else return 0;
}

double fractionalKnapsack(int W, struct Item arr[], int N) {

    qsort(arr, N, sizeof(struct Item), cmp);

    double finalvalue = 0.0;


    for (int i = 0; i < N; i++) {
        if (arr[i].weight <= W) {
            W -= arr[i].weight;
            finalvalue += arr[i].profit;
        }

        else {
            finalvalue += arr[i].profit * ((double)W / arr[i].weight);
            break;
        }
    }
    return finalvalue;
}

int main() {
    int N, W;

    printf("Enter the maximum weight of the knapsack: ");
    scanf("%d", &W);

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

    struct Item* arr = (struct Item*)malloc(N * sizeof(struct Item));

    printf("Enter the profit and weight of each item:\n");
    for (int i = 0; i < N; i++) {
        printf("Item %d:\n", i + 1);
        printf("Profit: ");
        scanf("%d", &arr[i].profit);
        printf("Weight: ");
        scanf("%d", &arr[i].weight);
    }



    double maxProfit = fractionalKnapsack(W, arr, N);
    printf("Maximum profit: %.2f\n", maxProfit);

    free(arr);

    return 0;
}
```

OUTPUT:



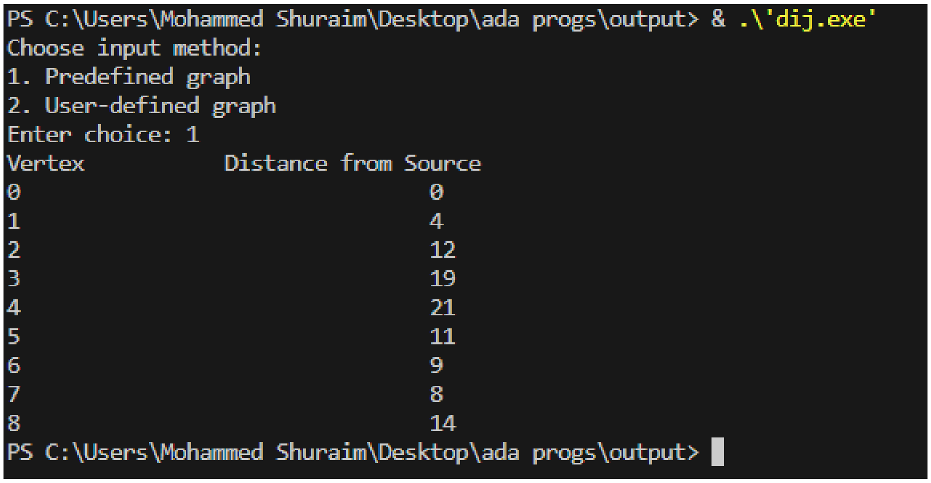
```
PS C:\Users\Mohammed Shuraim\Desktop\ada progs\output> cd 'c:\Users\Mohammed Shuraim\Desktop
PS C:\Users\Mohammed Shuraim\Desktop\ada progs\output> & .\'fracknap.exe'
Maximum profit: 240.00
```

11) From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra’s algorithm.

```
#include <limits.h>
#include <stdbool.h>
#include <stdio.h>
#define V 9
int minDistance(int dist[], bool sptSet[])
{
   int min = INT_MAX, min_index;
   for (int v = 0; v < V; v++)
      if (sptSet[v] == false && dist[v] <= min)
         min = dist[v], min_index = v;
   return min_index;
}

void printSolution(int dist[])
{
   printf("Vertex \t\t Distance from Source\n");
   for (int i = 0; i < V; i++)
      printf("%d \t\t\t\t %d\n", i, dist[i]);
}

void dijkstra(int graph[V][V], int src)
{
   int dist[V];
   bool sptSet[V];
   for (int i = 0; i < V; i++)
```

```
        dist[i] = INT_MAX, sptSet[i] = false;
    dist[src] = 0;
    for (int count = 0; count < V - 1; count++) {
        int u = minDistance(dist, sptSet);
        sptSet[u] = true;
        for (int v = 0; v < V; v++)
            if (!sptSet[v] && graph[u][v] && dist[u] != INT_MAX && dist[u] + graph[u][v] <
dist[v])
                dist[v] = dist[u] + graph[u][v];
    }
    printSolution(dist);
}

int main()
{
    int graph[V][V];
    int choice;
    printf("Choose input method:\n");
    printf("1. Predefined graph\n");
    printf("2. User-defined graph\n");
    printf("Enter choice: ");
    scanf("%d", &choice);

    switch (choice) {
    case 1:
        {
            int predefined_graph[V][V] = {
                { 0, 4, 0, 0, 0, 0, 0, 8, 0 },
                { 4, 0, 8, 0, 0, 0, 0, 11, 0 },
                { 0, 8, 0, 7, 0, 4, 0, 0, 2 },
                { 0, 0, 7, 0, 9, 14, 0, 0, 0 },
                { 0, 0, 0, 9, 0, 10, 0, 0, 0 },
                { 0, 0, 4, 14, 10, 0, 2, 0, 0 },
                { 0, 0, 0, 0, 0, 2, 0, 1, 6 },
                { 8, 11, 0, 0, 0, 0, 1, 0, 7 },
                { 0, 0, 2, 0, 0, 0, 6, 7, 0 }
            };
            for (int i = 0; i < V; i++)
                for (int j = 0; j < V; j++)
                    graph[i][j] = predefined_graph[i][j];
        }
        break;
    case 2:
        printf("Enter the adjacency matrix of the graph:\n");
        for (int i = 0; i < V; i++)
            for (int j = 0; j < V; j++)
                scanf("%d", &graph[i][j]);
        break;
    default:
        printf("Invalid choice\n");
        return 1;
    }
    dijkstra(graph, 0);
    return 0;}
```

OUTPUT:



```
PS C:\Users\Mohammed Shuraim\Desktop\ada progs\output> & .\'dij.exe'
Choose input method:
1. Predefined graph
2. User-defined graph
Enter choice: 1
Vertex           Distance from Source
0                               0
1                               4
2                               12
3                               19
4                               21
5                               11
6                               9
7                               8
8                               14
PS C:\Users\Mohammed Shuraim\Desktop\ada progs\output>
```