VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.

LAB REPORT
On

ANALYSIS AND DESIGN OF ALGORITHMS (23CS4PCADA)

Submitted by

Shashank U (1BM23CS314)

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
February-May 2025

B. M. S. College of Engineering,
Bull Temple Road, Bangalore 560019
(Affiliated To Visvesvaraya Technological University, Belgaum)
Department of Computer Science and Engineering

This is to certify that the Lab work entitled **"ANALYSIS AND DESIGN OF ALGORITHMS"** carried out by Shashank U **(1BM23CS314)**, who is 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 year 2024-25. The Lab report has been approved as it satisfies the academic requirements in respect of Analysis and Design of Algorithms Lab - **(23CS4PCADA)** work prescribed for the said degree.

Prof. Sarala D.V	**Dr. Kavitha Sooda**
Assistant Professor	Professor and Head
Department of CSE	Department of CSE
BMSCE, Bengaluru	BMSCE, Bengaluru

Index Sheet

Course outcomes:

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.

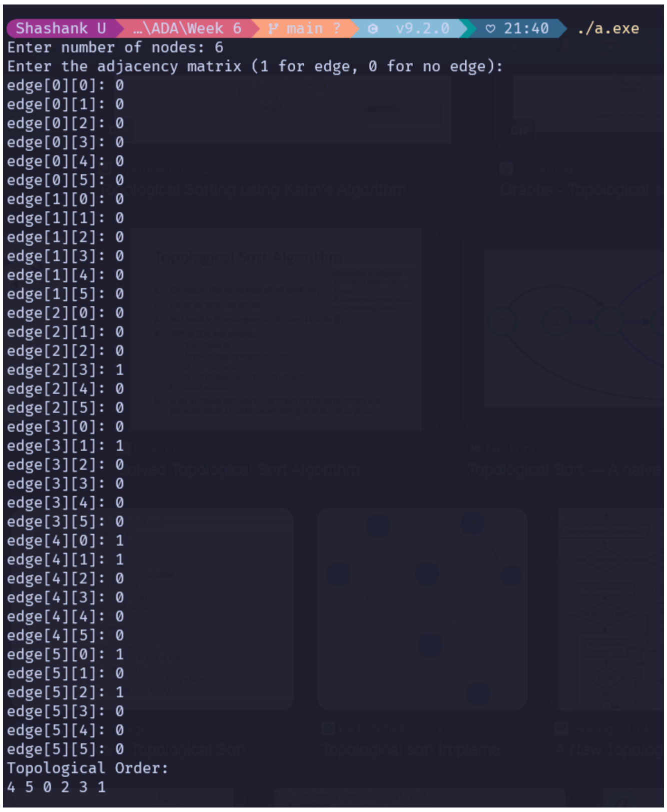
Lab program 1:

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

```
#include <stdio.h>

#include <stdlib.h>

// Create adjacency matrix

int** create_graph(int n){

        int** graph = (int**)calloc(n, sizeof(int*));

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

                graph[i] = (int*)calloc(n, sizeof(int));

        }


        printf("Enter the adjacency matrix (1 for edge, 0 for no edge):\n");

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

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

                        printf("edge[%d][%d]: ", i, j);

                        scanf("%d", &graph[i][j]);

                }

        }

        return graph;

}


// Calculate in-degrees of all nodes

void compute_indegree(int** graph, int* indegree, int n){

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

                indegree[i] = 0;

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

                        if(graph[j][i] == 1){

                                indegree[i]++;

                        }

                }

        }

}
```

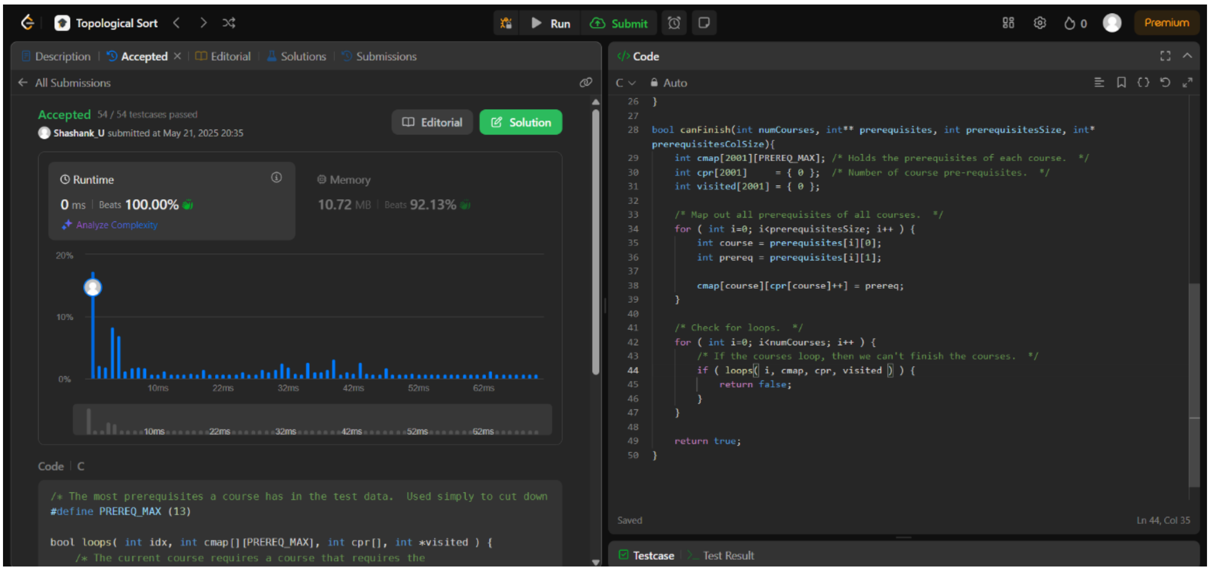
```
// Perform topological sort using source removal
void topo_sort(int** graph, int n){
        int* indegree = (int*)calloc(n, sizeof(int));
        int* visited = (int*)calloc(n, sizeof(int));
        int count = 0;

        compute_indegree(graph, indegree, n);

        printf("Topological Order:\n");
        while(count < n){
                int found = 0;
                for(int i = 0; i < n; i++){
                        if(indegree[i] == 0 && visited[i] == 0){
                                // Node with no incoming edges → print and remove
                                printf("%d ", i);
                                visited[i] = 1;
                                count++;
                                found = 1;
                                // Reduce indegree of its neighbors
                                for(int j = 0; j < n; j++){
                                        if(graph[i][j] == 1){
                                                indegree[j]--;
                                        }
                                }
                                break;
                        }
                }
                if(!found){
                        printf("\nCycle detected. Topological sort not possible.\n");
                        break;
                }
        }
        printf("\n");

        free(indegree);
        free(visited);
}
void free_graph(int** graph, int n){
        for(int i = 0; i < n; i++){
                free(graph[i]);
        }
        free(graph);
}
int main(){
        int n;
        printf("Enter number of nodes: ");
        scanf("%d", &n);

        int** graph = create_graph(n);
        topo_sort(graph, n);
        free_graph(graph, n);
        return 0;
}
```

OUTPUT:



```
Shashank U  …\ADA\Week 6  main ?  v9.2.0  ♡ 21:40  ./a.exe
Enter number of nodes: 6
Enter the adjacency matrix (1 for edge, 0 for no edge):
edge[0][0]: 0
edge[0][1]: 0
edge[0][2]: 0
edge[0][3]: 0
edge[0][4]: 0
edge[0][5]: 0
edge[1][0]: 0
edge[1][1]: 0
edge[1][2]: 0
edge[1][3]: 0
edge[1][4]: 0
edge[1][5]: 0
edge[2][0]: 0
edge[2][1]: 0
edge[2][2]: 0
edge[2][3]: 1
edge[2][4]: 0
edge[2][5]: 0
edge[3][0]: 0
edge[3][1]: 1
edge[3][2]: 0
edge[3][3]: 0
edge[3][4]: 0
edge[3][5]: 0
edge[4][0]: 1
edge[4][1]: 1
edge[4][2]: 0
edge[4][3]: 0
edge[4][4]: 0
edge[4][5]: 0
edge[5][0]: 1
edge[5][1]: 0
edge[5][2]: 1
edge[5][3]: 0
edge[5][4]: 0
edge[5][5]: 0
Topological Order:
4 5 0 2 3 1
```

LeetCode Program related to Topological sorting



Accepted 54 / 54 testcases passed

Shashank_U submitted at May 21, 2025 20:35

Runtime 0 ms | Beats 100.00%

Memory 10.72 MB | Beats 92.13%

Code | C

```c
/* The most prerequisites a course has in the test data.  Used simply to cut down
#define PREREQ_MAX (13)

bool loops( int idx, int cmap[][PREREQ_MAX], int cpr[], int *visited ) {
    /* The current course requires a course that requires the
```

Code

C Auto

```c
}

bool canFinish(int numCourses, int** prerequisites, int prerequisitesSize, int* prerequisitesColSize){
    int cmap[2001][PREREQ_MAX]; /* Holds the prerequisites of each course.  */
    int cpr[2001]     = { 0 };  /* Number of course pre-requisites.  */
    int visited[2001] = { 0 };

    /* Map out all prerequisites of all courses.  */
    for ( int i=0; i<prerequisitesSize; i++ ) {
        int course = prerequisites[i][0];
        int prereq = prerequisites[i][1];

        cmap[course][cpr[course]++] = prereq;
    }

    /* Check for loops.  */
    for ( int i=0; i<numCourses; i++ ) {
        /* If the courses loop, then we can't finish the courses.  */
        if ( loops( i, cmap, cpr, visited ) ) {
            return false;
        }
    }

    return true;
}
```

Saved

Ln 44, Col 35

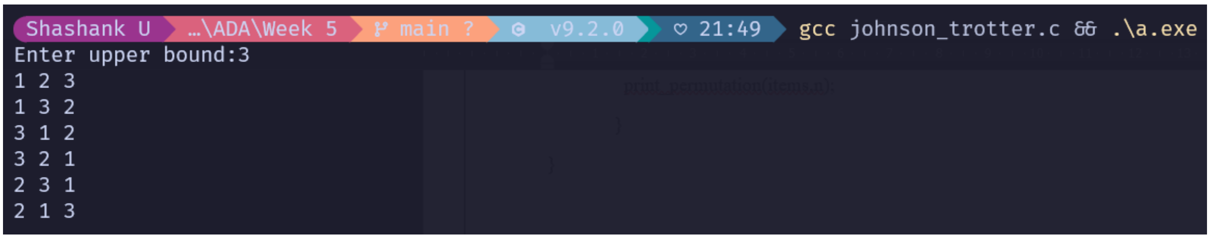
Lab program 2:

Implement Johnson Trotter algorithm to generate permutations.

```c
#include<stdio.h>

#include<stdlib.h>

#include<stdbool.h>

#define LEFT -1

#define RIGHT 1


typedef struct{

        int direction,value;

}element;


void init(element items[],int n){

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

                items[i].value = i+1;

                items[i].direction = LEFT;

        }

}


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


int largest_mobile_integer(element items[],int n,int* adj_index){
    int largest_mi = -1; //index
    int adj;
    for(int i=0;i<n;i++){
        int curr_dir = items[i].direction;
        adj = i + curr_dir; //adjacent index wrt mobile integer
        if(adj>=0 && adj<n){
         if(items[i].value > items[adj].value){
          if(largest_mi==-1 || items[i].value > items[largest_mi].value){
             largest_mi = i;
            }
        }
     }
    }
    if(largest_mi!=-1){
      *adj_index = largest_mi + items[largest_mi].direction;
    }
    return largest_mi;
}

void print_permutation(element items[],int n){
    for(int i=0;i<n;i++){
        printf("%d ",items[i].value);
    }
    printf("\n");
```

```
}

void reverse_direction(element items[],int n,int index){
        int mi = items[index].value; //mobile integer
        for(int i=0;i<n;i++){
                if(items[i].value>mi){
                        items[i].direction*=-1;
                }
        }
}



void johnsonn_trotter(element items[],int n){
        init(items,n);
        print_permutation(items,n);
        int adj,index;
        while(true){
          //Step 1: find out largest mobile integer
          index = largest_mobile_integer(items,n,&adj);
          if(index==-1){ break; }
         //Step 2: Swap it with it's adjacent
          swap(&items[index],&items[adj]);
         //Step 3: Reverse directions of integer > mobile integer
         //Since index was swapped so adj took the value of index
          reverse_direction(items,n,adj);
         //Step 4: Print the permutation obtained
          print_permutation(items,n);
        }
}
int main(){
        int n;
        printf("Enter upper bound:");

        scanf("%d",&n);

        element items[n];

        johnsonn_trotter(items,n);

        return 0;

}
```

OUTPUT:



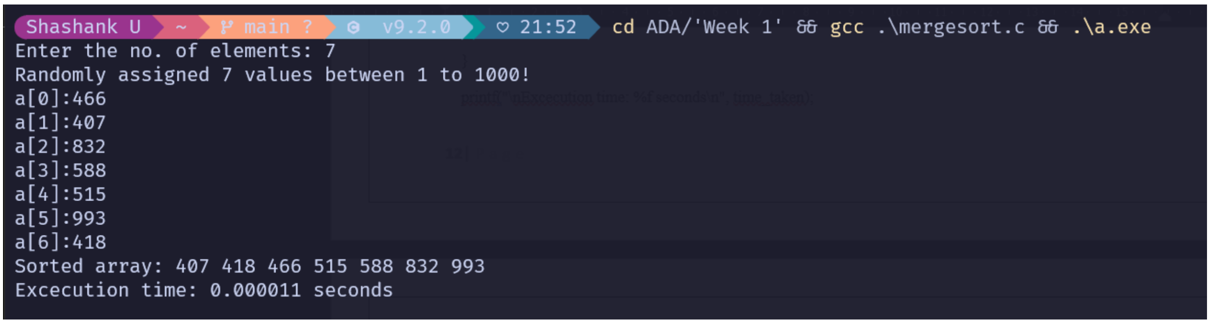
```
Shashank U  …\ADA\Week 5  main ?  v9.2.0  21:49  gcc johnson_trotter.c && .\a.exe
Enter upper bound:3
1 2 3
1 3 2
3 1 2
3 2 1
2 3 1
2 1 3
```

Lab Program 3

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

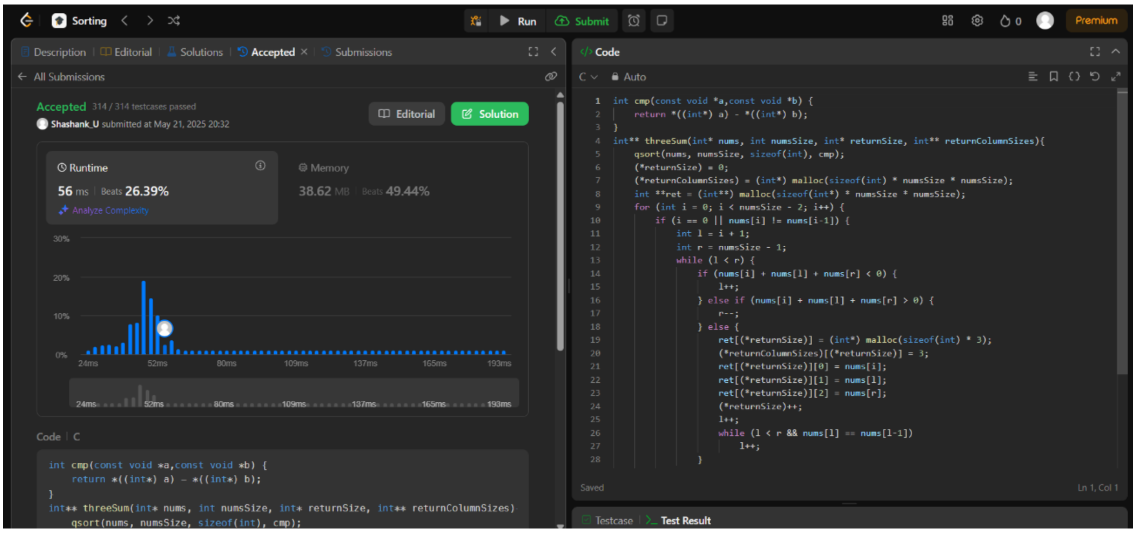
```
#include <stdio.h>

#include <stdlib.h>

#include<time.h>

#include<windows.h>



void merge(int a[], int low, int mid, int high) {

    int *c = (int*)malloc( (high-low+1)*sizeof(int) );

    int i = low, j = mid+1, k = 0;

    while (i <=mid && j <=high) {

        if (a[i] <= a[j]) {

            c[k++] = a[i++];

        } else {

            c[k++] = a[j++];

        }

    }

    while (i <=mid) {

        c[k++] = a[i++];
    }
    while (j <=high) {
        c[k++] = a[j++];
    }
    for(i=0;i<high-low+1;i++){
            a[low+i]=c[i];
    }
    free(c);
}

void mergesort(int a[], int low, int high) {
    if (low < high) {
        int mid = (low + high) / 2;
        mergesort(a, low, mid);
        mergesort(a, mid + 1, high);
        merge(a, low, mid, high);
    }
}
int main() {
    int n, i;
    LARGE_INTEGER frequency, start, end;
    QueryPerformanceFrequency(&frequency);
    printf("Enter the no. of elements: ");
    scanf("%d", &n);
    int a[n];
    srand((unsigned int)time(NULL));
    printf("Randomly assigned %d values between 1 to 1000!\n",n);
    for(int i=0;i<n;i++){
          a[i] = (rand()%1000) + 1;
          printf("a[%d]:%d\n",i,a[i]);
    }
    QueryPerformanceCounter(&start);
    mergesort(a,0,n-1);
    QueryPerformanceCounter(&end);
    double time_taken = (double)(end.QuadPart - start.QuadPart) / frequency.QuadPart;
    printf("Sorted array: ");
    for (i = 0; i < n; i++) {
        printf("%d ", a[i]);
    }
    printf("\nExcecution time: %f seconds\n", time_taken);

    return 0;
}
```

OUTPUT:



```
Shashank U  ~  main ?   v9.2.0   21:52  cd ADA/'Week 1' && gcc .\mergesort.c && .\a.exe
Enter the no. of elements: 7
Randomly assigned 7 values between 1 to 1000!
a[0]:466
a[1]:407
a[2]:832
a[3]:588
a[4]:515
a[5]:993
a[6]:418
Sorted array: 407 418 466 515 588 832 993
Excecution time: 0.000011 seconds
```

LeetCode Program related to sorting.



Accepted 314 / 314 testcases passed

Shashank_U submitted at May 21, 2025 20:32

Runtime: 56 ms | Beats 26.39%

Memory: 38.62 MB | Beats 49.44%

Code | C

```c
int cmp(const void *a,const void *b) {
    return *((int*) a) - *((int*) b);
}
int** threeSum(int* nums, int numsSize, int* returnSize, int** returnColumnSizes)
    qsort(nums, numsSize, sizeof(int), cmp);
```

```c
int cmp(const void *a,const void *b) {
    return *((int*) a) - *((int*) b);
}
int** threeSum(int* nums, int numsSize, int* returnSize, int** returnColumnSizes){
    qsort(nums, numsSize, sizeof(int), cmp);
    (*returnSize) = 0;
    (*returnColumnSizes) = (int*) malloc(sizeof(int) * numsSize * numsSize);
    int **ret = (int**) malloc(sizeof(int*) * numsSize * numsSize);
    for (int i = 0; i < numsSize - 2; i++) {
        if (i == 0 || nums[i] != nums[i-1]) {
            int l = i + 1;
            int r = numsSize - 1;
            while (l < r) {
                if (nums[i] + nums[l] + nums[r] < 0) {
                    l++;
                } else if (nums[i] + nums[l] + nums[r] > 0) {
                    r--;
                } else {
                    ret[(*returnSize)] = (int*) malloc(sizeof(int) * 3);
                    (*returnColumnSizes)[(*returnSize)] = 3;
                    ret[(*returnSize)][0] = nums[i];
                    ret[(*returnSize)][1] = nums[l];
                    ret[(*returnSize)][2] = nums[r];
                    (*returnSize)++;
                    l++;
                    while (l < r && nums[l] == nums[l-1])
                        l++;
                }
```

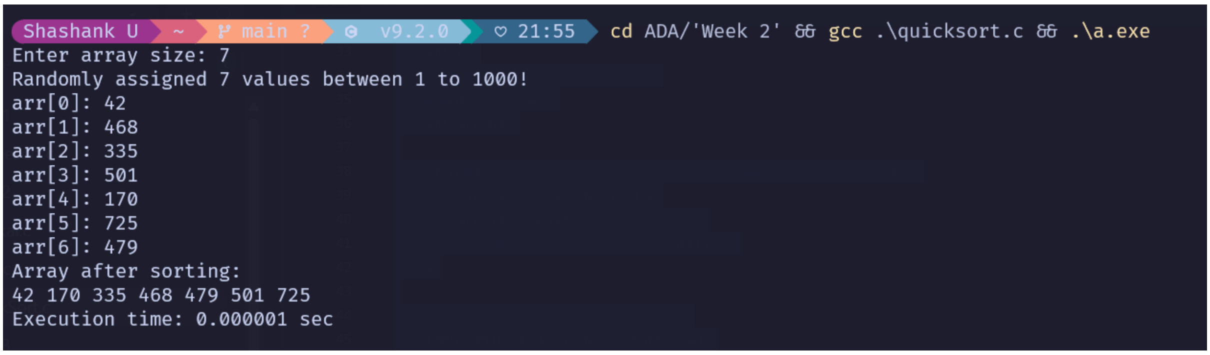
Lab Program 4

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

```
#include <stdio.h>

#include <stdlib.h>

#include <windows.h>


void swap(int *a, int *b) {

    int temp = *a;

    *a = *b;

    *b = temp;

}


int partition(int *arr, int low, int high) {

    int i = low + 1, j = high, pivot = arr[low];


    while (i <= j) {

        while (arr[i] <= pivot && i <= high) i++;

        while (arr[j] > pivot && j >= low) j--;

        if (i < j) swap(&arr[i], &arr[j]);

    }


    swap(&arr[j], &arr[low]);

    return j;

}


void quicksort(int *arr, int low, int high) {

    if (low < high) {

        int pivot_pos = partition(arr, low, high);

        quicksort(arr, low, pivot_pos - 1);
        quicksort(arr, pivot_pos + 1, high);
    }
}

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

    printf("Randomly assigned %d values between 1 to 1000!\n", n);
    for (int i = 0; i < n; i++) {
        arr[i] = (rand() % 1000) + 1;
        printf("arr[%d]: %d\n", i, arr[i]);
    }

    // Windows high-resolution timing
    LARGE_INTEGER frequency, start, end;
    QueryPerformanceFrequency(&frequency);
    QueryPerformanceCounter(&start);
    quicksort(arr, 0, n - 1);
   QueryPerformanceCounter(&end);
    double time_taken = (double)(end.QuadPart - start.QuadPart) / frequency.QuadPart;

    printf("Array after sorting:\n");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\nExecution time: %f sec\n", time_taken);
    return 0;
```

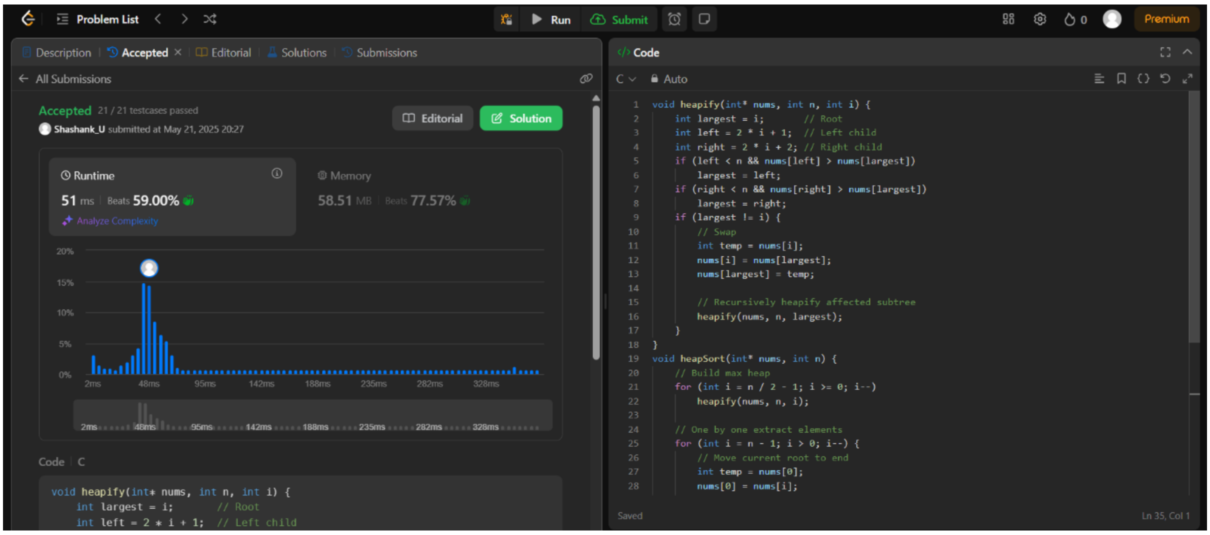
}

OUTPUT:



```
Shashank U  ~  main ?  v9.2.0  ♡ 21:55  cd ADA/'Week 2' && gcc .\quicksort.c && .\a.exe
Enter array size: 7
Randomly assigned 7 values between 1 to 1000!
arr[0]: 42
arr[1]: 468
arr[2]: 335
arr[3]: 501
arr[4]: 170
arr[5]: 725
arr[6]: 479
Array after sorting:
42 170 335 468 479 501 725
Execution time: 0.000001 sec
```

LeetCode Program related to sorting.



Accepted 21 / 21 testcases passed

Shashank_U submitted at May 21, 2025 20:27

Runtime: 51 ms | Beats 59.00%

Memory: 58.51 MB | Beats 77.57%

Code | C

```c
void heapify(int* nums, int n, int i) {
    int largest = i;       // Root
    int left = 2 * i + 1;  // Left child
```

```c
void heapify(int* nums, int n, int i) {
    int largest = i;       // Root
    int left = 2 * i + 1;  // Left child
    int right = 2 * i + 2; // Right child
    if (left < n && nums[left] > nums[largest])
        largest = left;
    if (right < n && nums[right] > nums[largest])
        largest = right;
    if (largest != i) {
        // Swap
        int temp = nums[i];
        nums[i] = nums[largest];
        nums[largest] = temp;

        // Recursively heapify affected subtree
        heapify(nums, n, largest);
    }
}
void heapSort(int* nums, int n) {
    // Build max heap
    for (int i = n / 2 - 1; i >= 0; i--)
        heapify(nums, n, i);

    // One by one extract elements
    for (int i = n - 1; i > 0; i--) {
        // Move current root to end
        int temp = nums[0];
        nums[0] = nums[i];
```

Lab Program 5:

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

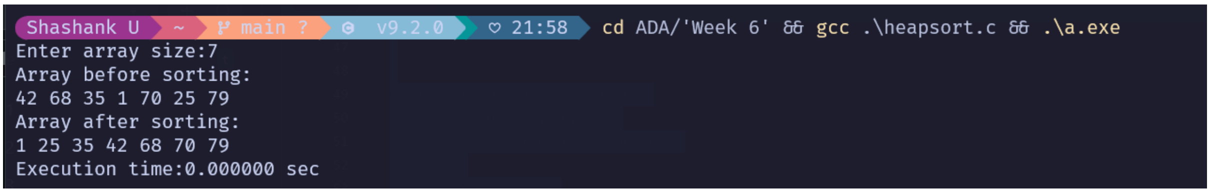
```c
#include<stdio.h>

#include<stdlib.h>

#include<time.h>



void swap(int *a,int *b){

        int temp = *a;

        *a = *b;

        *b = temp;

}
```

```
//build a max-heap
void heapify(int* arr,int n,int i){
        int l = 2*i + 1, r = 2*i + 2;
        int largest = i;

        if(l<n && arr[l]>arr[largest]){
                largest = l;
        }

        if(r<n && arr[r]>arr[largest]){
                largest = r;
        }

        if(largest!=i){
                swap(&arr[largest],&arr[i]);
                //heapify the affected portion
                heapify(arr,n,largest);
        }
}


void heapsort(int* arr,int n){

        //swap the root with last element of heap
        //heapify the new heap

        for(int i=n/2-1;i>=0;i--){
                heapify(arr,n,i);
        }
```

```
  for(int i=n-1;i>0;i--){
        swap(&arr[0],&arr[i]);
        heapify(arr,i,0);
    }

}


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


int main(){
    int n;
    printf("Enter array size:");
    scanf("%d",&n);
    int arr[n];
    printf("Array before sorting:\n");
    for(int i=0;i<n;i++){
        arr[i]=(rand()%100) + 1;
    }
    print_array(arr,n);
    printf("Array after sorting:\n");
    clock_t start,end;
    start = clock();
```

```
        heapsort(arr,n);
        end = clock();
        print_array(arr,n);
        printf("Execution time:%f sec\n",((float)end-start)/CLOCKS_PER_SEC);
        return 0;
}
```

OUTPUT:



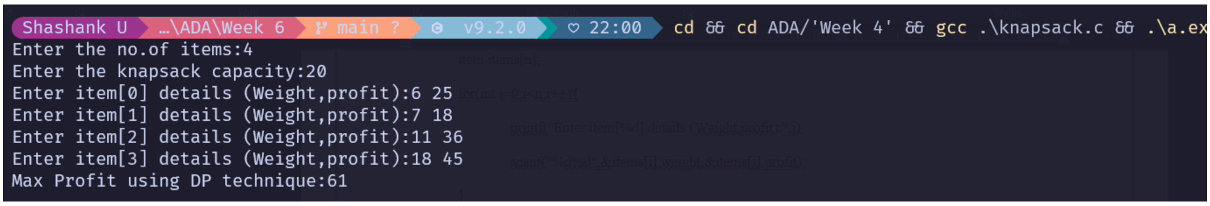
```
Shashank U  ~  main ?  v9.2.0  21:58  cd ADA/'Week 6' && gcc .\heapsort.c && .\a.exe
Enter array size:7
Array before sorting:
42 68 35 1 70 25 79
Array after sorting:
1 25 35 42 68 70 79
Execution time:0.000000 sec
```

Lab Program 6:

Implement 0/1 Knapsack problem using dynamic programming.

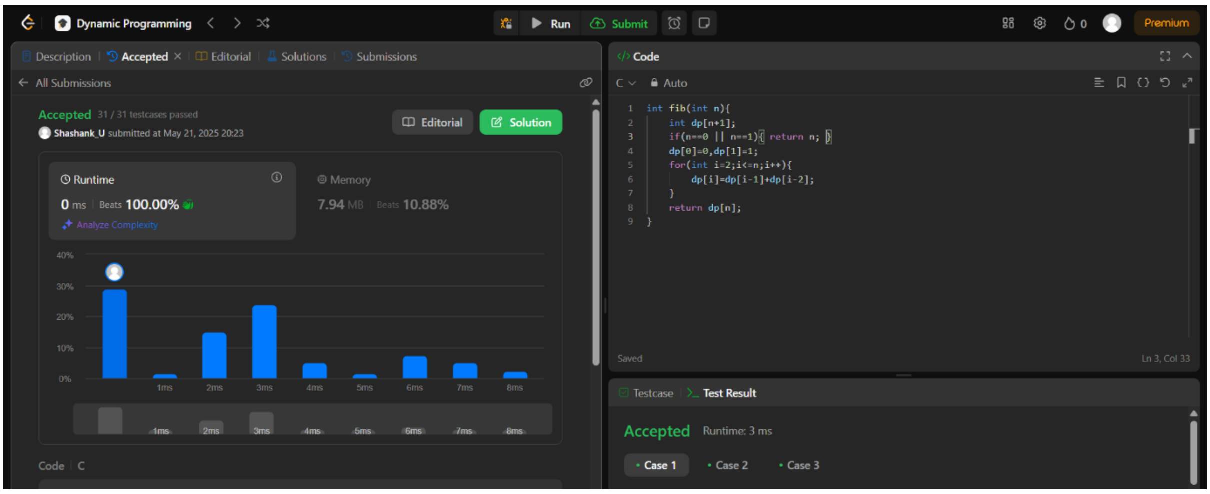
```
#include<stdio.h>
#include<stdlib.h>

typedef struct item{
        int weight,profit;
}item;

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

int knapsack(item* items,int capacity,int n){
        int dp[n+1][capacity+1];
        for(int i=0;i<=n;i++){
                for(int j=0;j<=capacity;j++){
                        if(i==0 || j==0){
                                dp[i][j]=0;
                        }else if(items[i-1].weight>j){
                                dp[i][j]=dp[i-1][j];
                        }
                else{
dp[i][j]=max(items[i-1].profit + dp[i-1][j-items[i-1].weight],dp[i-1][j]);
                  }
                }
        }
    return dp[n][capacity];
}



int main(){
        int n,w;
        printf("Enter the no.of items:");
        scanf("%d",&n);
        printf("Enter the knapsack capacity:");
        scanf("%d",&w);
        item items[n];
        for(int i=0;i<n;i++){
                printf("Enter item[%d] details (Weight,profit):",i);
                scanf("%d%d",&items[i].weight,&items[i].profit);
        }
        int max_profit = knapsack(items,w,n);
        printf("Max Profit using DP technique:%d\n",max_profit);
        return 0;
}
```

OUTPUT:



```
Shashank U  …\ADA\Week 6   main ?   v9.2.0   22:00  cd && cd ADA/'Week 4' && gcc .\knapsack.c && .\a.ex
Enter the no.of items:4
Enter the knapsack capacity:20
Enter item[0] details (Weight,profit):6 25
Enter item[1] details (Weight,profit):7 18
Enter item[2] details (Weight,profit):11 36
Enter item[3] details (Weight,profit):18 45
Max Profit using DP technique:61
```

LeetCode Program related to Knapsack problem or Dynamic Programming.



Accepted 31 / 31 testcases passed

Shashank_U submitted at May 21, 2025 20:23

Runtime: 0 ms | Beats 100.00%

Memory: 7.94 MB | Beats 10.88%

Code | C

```c
int fib(int n){
    int dp[n+1];
    if(n==0 || n==1){ return n; }
    dp[0]=0,dp[1]=1;
    for(int i=2;i<=n;i++){
        dp[i]=dp[i-1]+dp[i-2];
    }
    return dp[n];
}
```

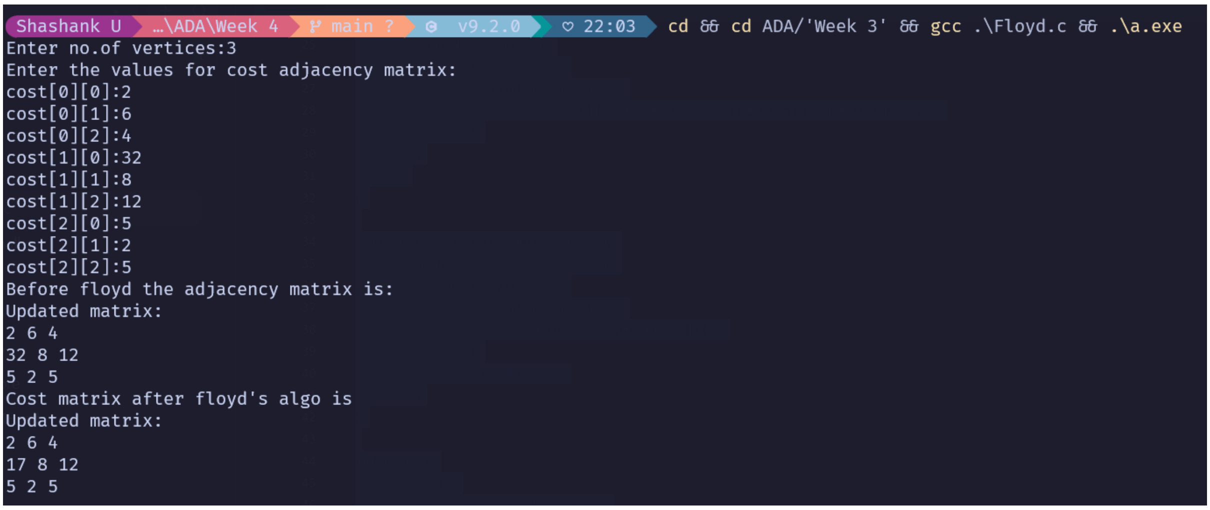
Accepted Runtime: 3 ms

Lab Program 7:

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

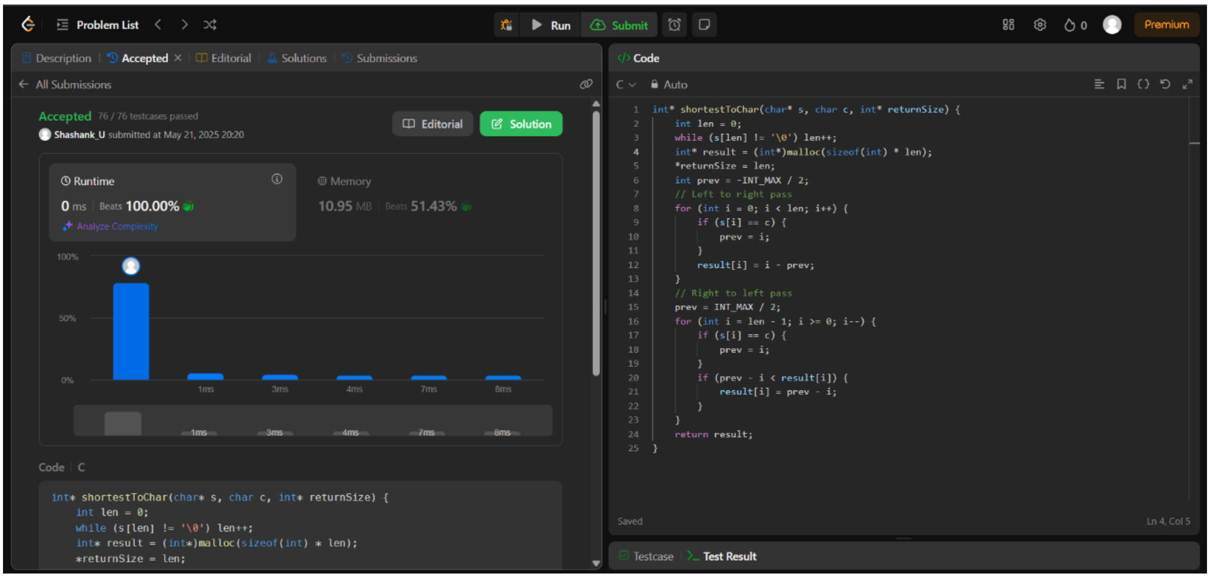
```c
#include<stdio.h>

#include<stdlib.h>

int min(int a,int b){

        return a<b?a:b;

}


int** read_data(int* n){

        printf("Enter no.of vertices:");

        scanf("%d",n);

        int **matrix = (int**)calloc(*n,sizeof(int*));

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

                matrix[i]=(int*)calloc(*n,sizeof(int));

        }

        printf("Enter the values for cost adjacency matrix:\n");

        for(int i=0;i<*n;i++){
                for(int j=0;j<*n;j++){
                        printf("cost[%d][%d]:",i,j);
                        scanf("%d",&matrix[i][j]);
                }
        }
        return matrix;
}

void floyd(int **matrix,int n){
   for(int k=0;k<n;k++){
        for(int i=0;i<n;i++){
                for(int j=0;j<n;j++){
                        matrix[i][j]=min(matrix[i][j],matrix[i][k]+matrix[k][j]);
                }
        }
   }
}

void print_data(int** matrix,int n){
        printf("Updated matrix:\n");
        for(int i=0;i<n;i++){
                for(int j=0;j<n;j++){
                        printf("%d ",matrix[i][j]);
                }
                printf("\n");
        }
}

int main(){
        int n;
        int** cost = read_data(&n);

        printf("Before floyd the adjacency matrix is:\n");

        print_data(cost,n);

        printf("Cost matrix after floyd's algo is\n");

        floyd(cost,n);

        print_data(cost,n);

        return 0;

}
```

OUTPUT:



```
Shashank U  …\ADA\Week 4  main ?  v9.2.0  22:03  cd && cd ADA/'Week 3' && gcc .\Floyd.c && .\a.exe
Enter no.of vertices:3
Enter the values for cost adjacency matrix:
cost[0][0]:2
cost[0][1]:6
cost[0][2]:4
cost[1][0]:32
cost[1][1]:8
cost[1][2]:12
cost[2][0]:5
cost[2][1]:2
cost[2][2]:5
Before floyd the adjacency matrix is:
Updated matrix:
2 6 4
32 8 12
5 2 5
Cost matrix after floyd's algo is
Updated matrix:
2 6 4
17 8 12
5 2 5
```

LeetCode Program related to shortest distance calculation



Accepted 76 / 76 testcases passed

Shashank_U submitted at May 21, 2025 20:20

Runtime 0 ms | Beats 100.00%

Memory 10.95 MB | Beats 51.43%

Code | C

```
int* shortestToChar(char* s, char c, int* returnSize) {
    int len = 0;
    while (s[len] != '\0') len++;
    int* result = (int*)malloc(sizeof(int) * len);
    *returnSize = len;
```

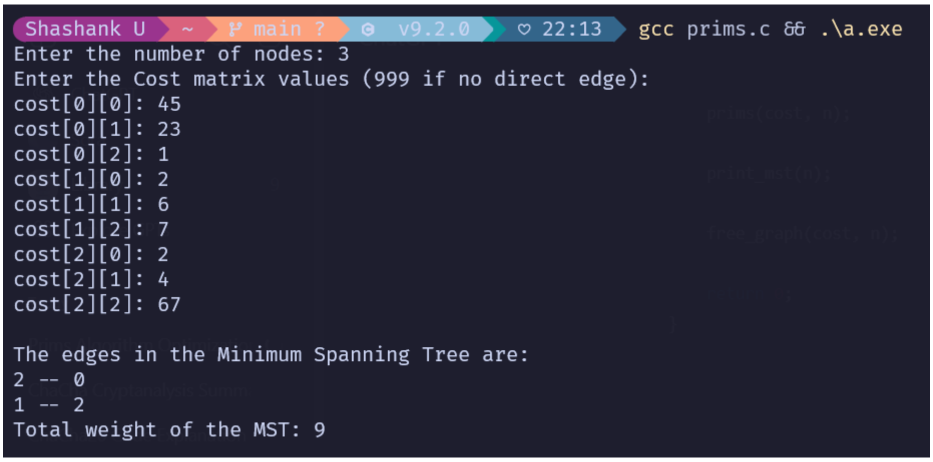
```
int* shortestToChar(char* s, char c, int* returnSize) {
    int len = 0;
    while (s[len] != '\0') len++;
    int* result = (int*)malloc(sizeof(int) * len);
    *returnSize = len;
    int prev = -INT_MAX / 2;
    // Left to right pass
    for (int i = 0; i < len; i++) {
        if (s[i] == c) {
            prev = i;
        }
        result[i] = i - prev;
    }
    // Right to left pass
    prev = INT_MAX / 2;
    for (int i = len - 1; i >= 0; i--) {
        if (s[i] == c) {
            prev = i;
        }
        if (prev - i < result[i]) {
            result[i] = prev - i;
        }
    }
    return result;
}
```

Lab Program 7

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

```
#include<stdio.h>

#include<stdlib.h>


int sum = 0;  // To store the total weight of MST

int t[10][2]; // To store the edges of MST


void prims(int cost[10][10], int n) {

    int i, j, u, v;

    int min, source;

    int p[10], d[10], s[10];  // Arrays to store parent, distance, and status

    min = 999;  // Initialize min to a large number (infinity)

    source = 0;  // Starting vertex (vertex 0)


    // Initialize arrays

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

        d[i] = cost[source][i];  // Initialize distance to the source vertex

        s[i] = 0;  // Initially, no vertex is in the MST

        p[i] = source;  // Set the parent of each vertex to the source

    }

    s[source] = 1;  // Mark the source vertex as included in MST

    sum = 0;  // Total weight of the MST

    int k = 0;  // Counter for the number of edges in MST


    // Main loop to find MST

    for (i = 0; i < n - 1; i++) {  // Repeat for n-1 iterations

        min = 999;  // Reset min value for each iteration

        u = -1;  // Reset u (the vertex to be added to the MST)


        // Find the vertex with the minimum distance to the MST

        for (j = 0; j < n; j++) {
            if (s[j] == 0 && d[j] < min) {  // If vertex j is not in MST and has a smaller distance
                min = d[j];
                u = j;  // Select vertex u with the smallest edge
            }
        }

        if (u != -1) {
            // Add edge (u, p[u]) to MST
            t[k][0] = u;  // Store the edge
            t[k][1] = p[u];
            k++;  // Increment edge counter
            sum += cost[u][p[u]];  // Add the edge weight to the total sum
            s[u] = 1;  // Mark u as added to the MST

            // Update distances for adjacent vertices
            for (v = 0; v < n; v++) {
                if (s[v] == 0 && cost[u][v] < d[v]) {  // If v is not in MST and a shorter edge exists
                    d[v] = cost[u][v];  // Update the distance to v
                    p[v] = u;  // Set parent of v as u
                }
            }
        }
    }
}

int** create_graph(int n) {
    int** cost = (int**)calloc(n, sizeof(int*));
    for (int i = 0; i < n; i++) {
        cost[i] = (int*)calloc(n, sizeof(int));
    }
    printf("Enter the Cost matrix values (999 if no direct edge)\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            printf("cost[%d][%d]:", i, j);
            scanf("%d", &cost[i][j]);
        }
    }
    return cost;
}

void print_mst(int n) {
    printf("The edges in the Minimum Spanning Tree are:\n");
    for (int i = 0; i < n - 1; i++) {
        printf("%d -- %d\n", t[i][0], t[i][1]);
    }
    printf("Total weight of the MST: %d\n", sum);
}

void free_graph(int** cost, int n) {
    for (int i = 0; i < n; i++) {
        free(cost[i]);
    }
    free(cost);
}

int main() {
    int n;
    printf("Enter the number of nodes: ");
    scanf("%d", &n);
    int** cost = create_graph(n);


    prims(cost, n);


    print_mst(n);


    free_graph(cost, n);

    return 0;

}
```

OUTPUT:



```
Shashank U  ~  main ?  v9.2.0  ♡ 22:13  gcc prims.c && .\a.exe
Enter the number of nodes: 3
Enter the Cost matrix values (999 if no direct edge):
cost[0][0]: 45
cost[0][1]: 23
cost[0][2]: 1
cost[1][0]: 2
cost[1][1]: 6
cost[1][2]: 7
cost[2][0]: 2
cost[2][1]: 4
cost[2][2]: 67

The edges in the Minimum Spanning Tree are:
2 -- 0
1 -- 2
Total weight of the MST: 9
```

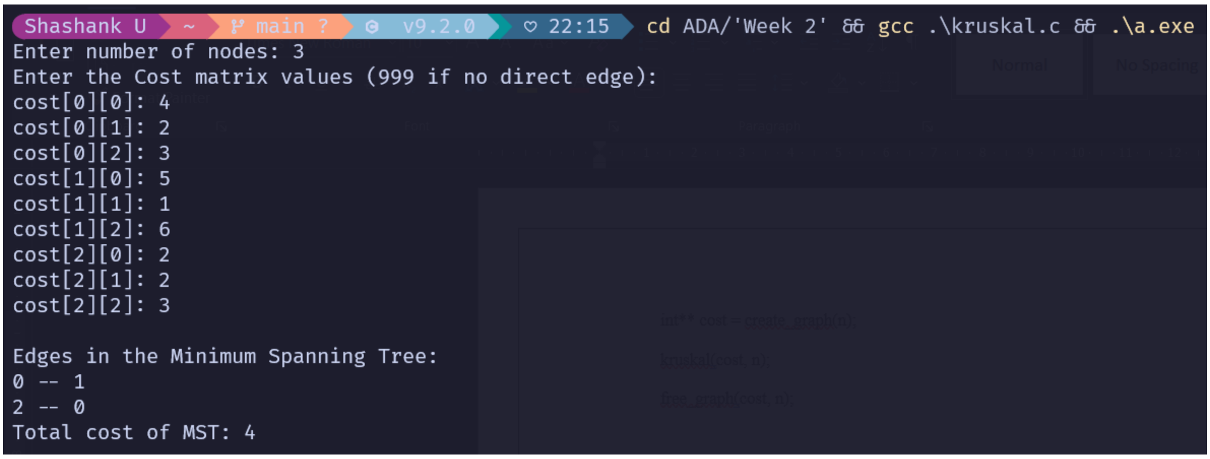
Lab Program 8

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

```
#include <stdio.h>

#include <stdlib.h>


#define INF 999


int find(int parent[], int i){

        if(parent[i] != i)

                parent[i] = find(parent, parent[i]);

        return parent[i];

}
```

```
int** create_graph(int n){
        int** cost = (int**)calloc(n, sizeof(int*));
        for(int i = 0; i < n; i++){
                cost[i] = (int*)calloc(n, sizeof(int));
        }

        printf("Enter the Cost matrix values (999 if no direct edge):\n");
        for(int i = 0; i < n; i++){
                for(int j = 0; j < n; j++){
                        printf("cost[%d][%d]: ", i, j);
                        scanf("%d", &cost[i][j]);
                }
        }
        return cost;
}

void kruskal(int** cost, int n){
        int parent[10];
        int t[10][2]; // MST edge list
        int count = 0, sum = 0, k = 0;
        int min, u, v;

        // Initialize each node as its own parent
        for(int i = 0; i < n; i++){
                parent[i] = i;
        }

        while(count < n - 1){
                min = INF;
                u = -1;
                v = -1;

                // Find the minimum weight edge that doesn't form a cycle
                for(int i = 0; i < n; i++){
                        for(int j = 0; j < n; j++){
                                if(find(parent, i) != find(parent, j) && cost[i][j] < min){
                                        min = cost[i][j];
                                        u = i;
                                        v = j;
                                }
                        }
                }

                // Union the sets
                if(u != -1 && v != -1){
                        int root_u = find(parent, u);
                        int root_v = find(parent, v);
                        parent[root_u] = root_v;

                        t[k][0] = u;
                        t[k][1] = v;
                        sum += cost[u][v];
                        k++;
                        count++;
                }
        }

        // Print the MST
        printf("\nEdges in the Minimum Spanning Tree:\n");
        for(int i = 0; i < k; i++){
                printf("%d -- %d\n", t[i][0], t[i][1]);
        }
        printf("Total cost of MST: %d\n", sum);
}

void free_graph(int** cost, int n){
        for(int i = 0; i < n; i++){
                free(cost[i]);
        }
        free(cost);
}

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

        int** cost = create_graph(n);
        kruskal(cost, n);
        free_graph(cost, n);

        return 0;
}
```

OUTPUT:



```
Shashank U  ~  main ?  v9.2.0  ♡ 22:15  cd ADA/'Week 2' && gcc .\kruskal.c && .\a.exe
Enter number of nodes: 3
Enter the Cost matrix values (999 if no direct edge):
cost[0][0]: 4
cost[0][1]: 2
cost[0][2]: 3
cost[1][0]: 5
cost[1][1]: 1
cost[1][2]: 6
cost[2][0]: 2
cost[2][1]: 2
cost[2][2]: 3

Edges in the Minimum Spanning Tree:
0 -- 1
2 -- 0
Total cost of MST: 4
```

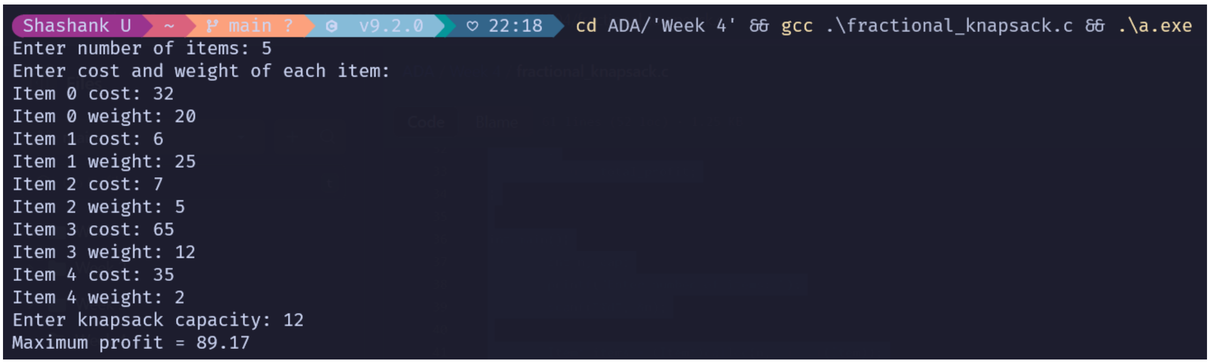
Lab Program 9

Implement fractional knapsack problem using Greedy technique.

```
#include <stdio.h>

#include <stdlib.h>


typedef struct {

        int cost;

        int weight;

        float ratio;

} Item;


void sort_items(Item* items, int n){

        for(int i=0;i<n-1;i++){

                for(int j=0;j<n-i-1;j++){

                        if(items[j].ratio < items[j+1].ratio){

                                Item temp = items[j];

                                items[j] = items[j+1];

                                items[j+1] = temp;

                        }

                }

        }
```

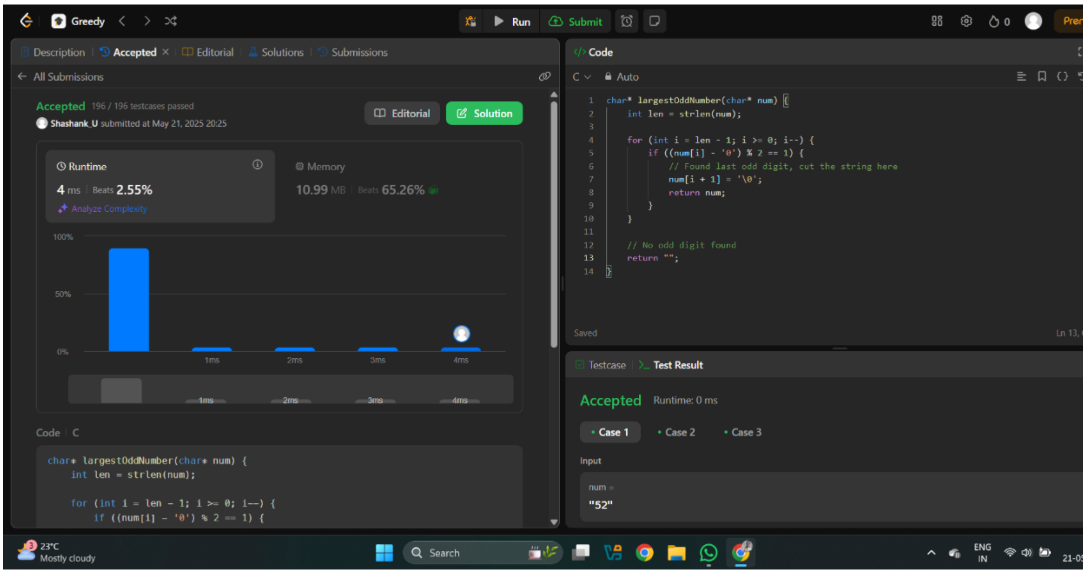
```
}


float fractional_knapsack(Item* items, int n, int capacity){
        float total_profit = 0.0;
        for(int i=0;i<n;i++){
                if(capacity >= items[i].weight){
                        total_profit += items[i].cost;
                        capacity -= items[i].weight;
                }else{
                        total_profit += (float)items[i].cost * ((float)capacity / items[i].weight);
                        break;
                }
        }
        return total_profit;
}


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


        Item* items = (Item*)calloc(n, sizeof(Item));


        printf("Enter cost and weight of each item:\n");
        for(int i=0;i<n;i++){
                printf("Item %d cost: ", i);
                scanf("%d", &items[i].cost);
                printf("Item %d weight: ", i);
                scanf("%d", &items[i].weight);
                items[i].ratio = (float)items[i].cost / items[i].weight;
        }

        printf("Enter knapsack capacity: ");
        scanf("%d", &cap);

        sort_items(items, n);
        float max_profit = fractional_knapsack(items, n, cap);
        printf("Maximum profit = %.2f\n", max_profit);

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

OUTPUT:



```
Shashank U  ~  main ?  v9.2.0  22:18  cd ADA/'Week 4' && gcc .\fractional_knapsack.c && .\a.exe
Enter number of items: 5
Enter cost and weight of each item:
Item 0 cost: 32
Item 0 weight: 20
Item 1 cost: 6
Item 1 weight: 25
Item 2 cost: 7
Item 2 weight: 5
Item 3 cost: 65
Item 3 weight: 12
Item 4 cost: 35
Item 4 weight: 2
Enter knapsack capacity: 12
Maximum profit = 89.17
```

LeetCode Program related to Greedy Technique algorithms.



Accepted 196 / 196 testcases passed
Shashank_U submitted at May 21, 2025 20:25

Runtime 4 ms | Beats 2.55%

Memory 10.99 MB | Beats 65.26%

```
char* largestOddNumber(char* num) {
    int len = strlen(num);

    for (int i = len - 1; i >= 0; i--) {
        if ((num[i] - '0') % 2 == 1) {
            // Found last odd digit, cut the string here
            num[i + 1] = '\0';
            return num;
        }
    }

    // No odd digit found
    return "";
}
```

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

Input

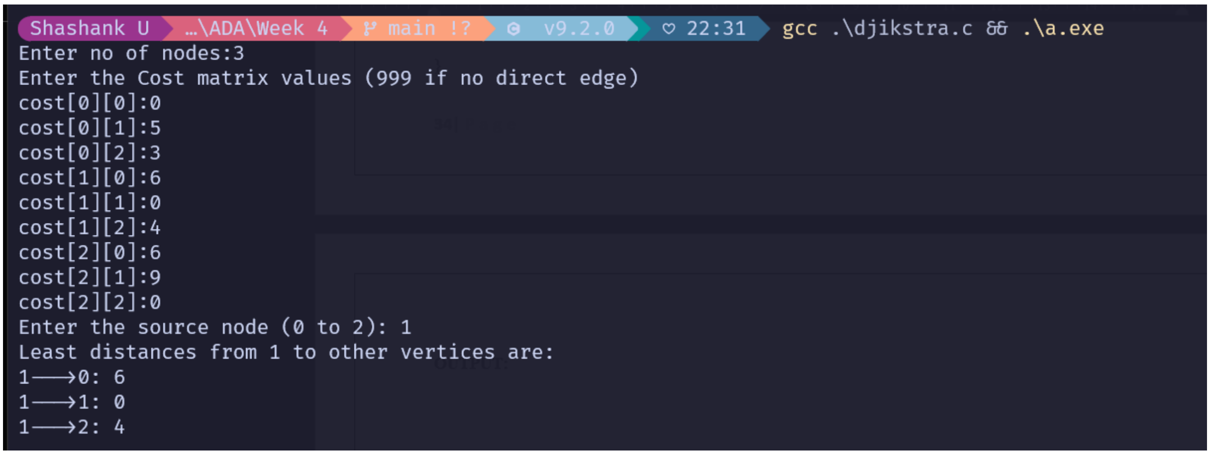
num =
"52"

Lab Program 10

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

```
#include<stdio.h>

#include<stdlib.h>

#include<stdbool.h>

int** create_graph(int n){

        int** cost = (int**)calloc(n,sizeof(int*));

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

                cost[i]=(int*)calloc(n,sizeof(int));

        }

        printf("Enter the Cost matrix values (999 if no direct edge)\n");

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

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

                        printf("cost[%d][%d]:",i,j);

                        scanf("%d",&cost[i][j]);

                }

        }

        return cost;

}


int* djikstra(int** cost,int n,int source){

        int min,u;

        bool visited[n];

        int *sd = (int*)calloc(n,sizeof(int));

        //initialise visited and shortest distance array

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

                if(j==source){

                        visited[j]=true;

                        sd[j]=0;
                }else{

                        visited[j]=false;

                        sd[j]=cost[source][j];

                }

        }

        //Execute the algorithm

        for(int k=1;k<n;k++){

                min=999;

                //find out the unvisited edge with least updated distance from source

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

                        if(!visited[i] && sd[i]<min){

                                min=sd[i];

                                u=i;

                        }

                }

                visited[u]=true;

                //perform relaxation

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

                        if(!visited[j]){

                                if(sd[u] + cost[u][j] < sd[j]){

                                    sd[j] = sd[u] + cost[u][j];

                                }

                        }

                }

        }


        return sd;

}


void print_distance(int *d,int n,int source){
        printf("Least distances from %d to other vertices are:\n",source);
        for(int i=0;i<n;i++){
                printf("%d--->%d: %d\n",source,i,d[i]);
        }
}

void free_graph(int** cost,int n){
        for(int i=0;i<n;i++){
            free(cost[i]);
        }
        free(cost);
}

int main(){
        int n;
        printf("Enter no of nodes:");
        scanf("%d",&n);
        int** cost = create_graph(n);
start:
        int source;
        printf("Enter the source node:");
        scanf("%d",&source);
        if(source>=n || source<0) goto start;

        int* shortest_distance = djikstra(cost,n,source);
        print_distance(shortest_distance,n,source);
        free_graph(cost,n);
        free(shortest_distance);
        return 0;
}
```

OUTPUT:



```
Shashank U  …\ADA\Week 4  main !?  v9.2.0  ♡ 22:31  gcc .\djikstra.c && .\a.exe
Enter no of nodes:3
Enter the Cost matrix values (999 if no direct edge)
cost[0][0]:0
cost[0][1]:5
cost[0][2]:3
cost[1][0]:6
cost[1][1]:0
cost[1][2]:4
cost[2][0]:6
cost[2][1]:9
cost[2][2]:0
Enter the source node (0 to 2): 1
Least distances from 1 to other vertices are:
1⟶0: 6
1⟶1: 0
1⟶2: 4
```

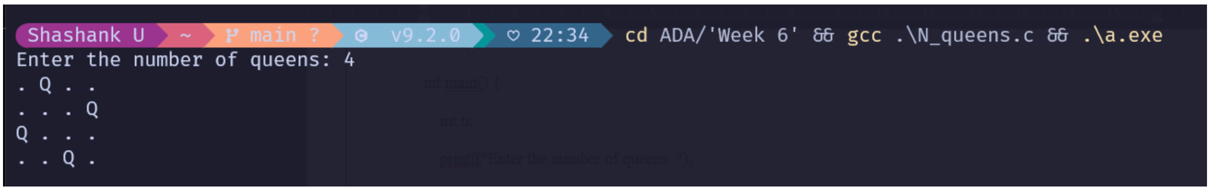
Lab Program 11

Implement “N-Queens Problem” using Backtracking

```
#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>


// Function to print the chessboard

void print_board(int** board, int n) {

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

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

            if(board[i][j] == 1)

                printf("Q ");  // Queen is placed

            else

                printf(". ");  // Empty space

        }

        printf("\n");

    }

}


// Check if it's safe to place a queen at board[row][col]
```

```
bool is_safe(int** board, int row, int col, int n) {
    // Check column
    for(int i = 0; i < row; i++) {
        if(board[i][col] == 1)
            return false;
    }

    // Check upper-left diagonal
    for(int i = row-1, j = col-1; i >= 0 && j >= 0; i--, j--) {
        if(board[i][j] == 1)
            return false;
    }

    // Check upper-right diagonal
    for(int i = row-1, j = col+1; i >= 0 && j < n; i--, j++) {
        if(board[i][j] == 1)
            return false;
    }

    return true;
}

// Solve the N-Queens problem using backtracking
bool solve_n_queens(int** board, int row, int n) {
    // If all queens are placed, return true
    if(row >= n)
        return true;

    // Try all columns in the current row
    for(int col = 0; col < n; col++) {
        // Check if it's safe to place the queen
        if(is_safe(board, row, col, n)) {
            // Place queen
            board[row][col] = 1;

            // Recur to place the queen in the next row
            if(solve_n_queens(board, row + 1, n))
                return true;

            // If placing queen in this position doesn't lead to a solution, backtrack
            board[row][col] = 0;
        }
    }

    return false;
}

// Function to solve the N-Queens problem
void n_queens(int n) {
    // Create an empty board
    int** board = (int**)calloc(n, sizeof(int*));
    for(int i = 0; i < n; i++) {
        board[i] = (int*)calloc(n, sizeof(int));
    }

    // Try to solve the problem starting from the first row
    if(solve_n_queens(board, 0, n)) {
        print_board(board, n);
    } else {
        printf("Solution does not exist.\n");
    }

    // Free the memory for the board
    for(int i = 0; i < n; i++) {
        free(board[i]);
    }
    free(board);
}

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

    // Solve the N-Queens problem
    n_queens(n);

    return 0;
}
```

OUTPUT:



```
Shashank U  ~  main ?  v9.2.0  22:34  cd ADA/'Week 6' && gcc .\N_queens.c && .\a.exe
Enter the number of queens: 4
. Q . .
. . . Q
Q . . .
. . Q .
```