# VISVESVARAYA TECHNOLOGICAL UNIVERSITY

**“JnanaSangama”, Belgaum -590014, Karnataka.**

**LAB REPORT**
**On**

**ANALYSIS AND DESIGN OF ALGORITHMS (23CS4PCADA)**

**Submitted by**

**Subramanya J (1BM23CS343)**

**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 Subramanya J **(1BM23CS343)**, 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.

| | |
|---|---|
| **Dr. 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. |

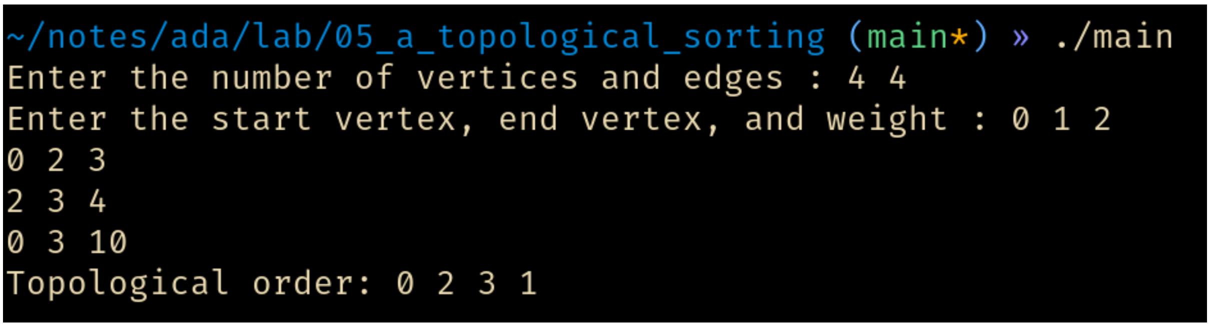
## Lab program 1:

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

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

#define MAX 100

void topological_sort(int **adj, int n) {
        int in_degree[MAX] = {0};
        int *order = malloc(n * sizeof(int));
        int index = 0;

        for (int i = 0; i < n; i++)
                for (int j = 0; j < n; j++)
                        if (adj[i][j] != 0)
                                in_degree[j]++;

        int stack[MAX], top = -1;
        for (int i = 0; i < n; i++)
                if (in_degree[i] == 0)
                        stack[++top] = i;

        while (top != -1) {
                int u = stack[top--];
                order[index++] = u;

                for (int v = 0; v < n; v++) {
                        if (adj[u][v] != 0) {
                                in_degree[v]--;
                                if (in_degree[v] == 0)
```

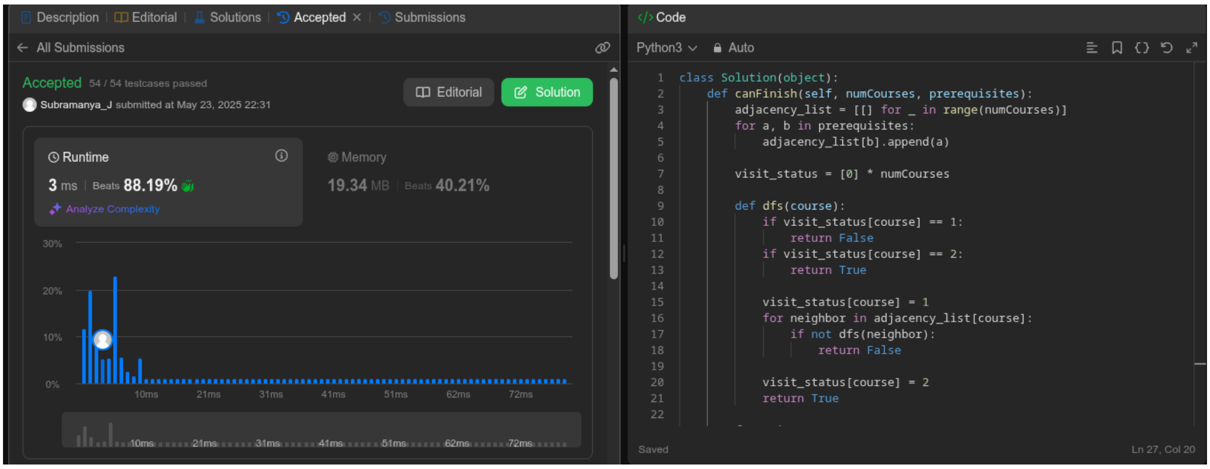
```
                    stack[++top] = v;
                }
            }
        }

        if (index != n) {
            printf("Cycle detected. Topological sorting not possible.\n");
        } else {
            printf("Topological order: ");
            for (int i = 0; i < n; i++)
                printf("%d ", order[i]);
            printf("\n");
        }

        free(order);
}

int main() {
        int n, e;
        int **graph = graph_create(&n, &e);
        topological_sort(graph, n);

        for (int i = 0; i < n; i++)
            free(graph[i]);
        free(graph);

        return 0;
}
```

**Ouput :**



```
~/notes/ada/lab/05_a_topological_sorting (main*) » ./main
Enter the number of vertices and edges : 4 4
Enter the start vertex, end vertex, and weight : 0 1 2
0 2 3
2 3 4
0 3 10
Topological order: 0 2 3 1
```

## LeetCode 207: Course Schedule



Accepted 54 / 54 testcases passed

Subramanya_J submitted at May 23, 2025 22:31

Runtime

3 ms | Beats 88.19%

Memory

19.34 MB | Beats 40.21%

```python
class Solution(object):
    def canFinish(self, numCourses, prerequisites):
        adjacency_list = [[] for _ in range(numCourses)]
        for a, b in prerequisites:
            adjacency_list[b].append(a)

        visit_status = [0] * numCourses

        def dfs(course):
            if visit_status[course] == 1:
                return False
            if visit_status[course] == 2:
                return True

            visit_status[course] = 1
            for neighbor in adjacency_list[course]:
                if not dfs(neighbor):
                    return False

            visit_status[course] = 2
            return True

```

Saved

Ln 27, Col 20

## Lab program 2:

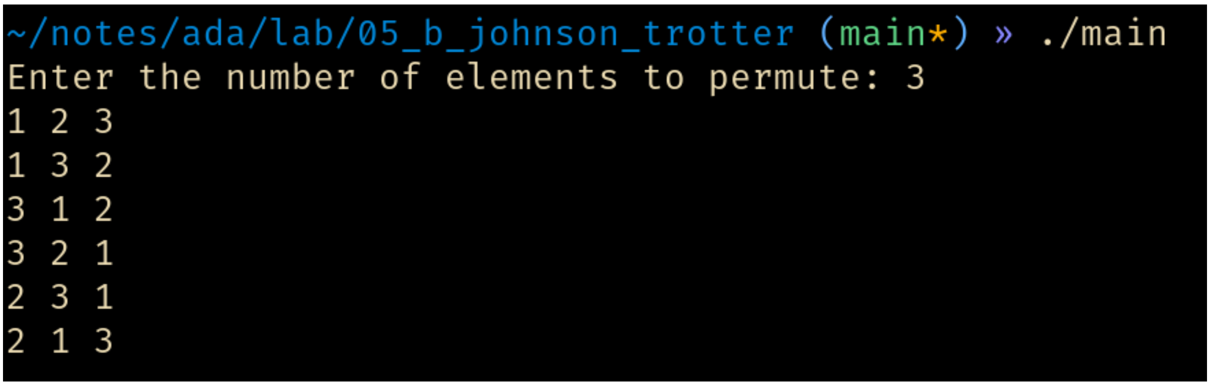
Implement Johnson Trotter algorithm to generate permutations.

```
#include <stdio.h>

#include <stdlib.h>


#define L -1

#define R 1


typedef struct { int v, d; } E;


void p(E *a, int n) {

    for (int i = 0; i < n; i++) printf("%d ", a[i].v);

    printf("\n");

}


int lm(E *a, int n) {

    int m = -1, lv = -1;

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

        int adj = i + a[i].d;

        if (adj >= 0 && adj < n && a[i].v > a[adj].v && a[i].v > lv) {

            lv = a[i].v;

            m = i;

        }

    }

    return m;

}
```

```
void rd(E *a, int n, int v) {

        for (int i = 0; i < n; i++) if (a[i].v > v) a[i].d *= -1;

}


void jt(int n) {

        E *a = malloc(n * sizeof(E));

        for (int i = 0; i < n; i++) { a[i].v = i + 1; a[i].d = L; }

        p(a, n);


        while (1) {

                int idx = lm(a, n);

                if (idx == -1) break;

                int d = a[idx].d, si = idx + d;

                E t = a[idx]; a[idx] = a[si]; a[si] = t;

                rd(a, n, a[si].v);

                p(a, n);

        }


        free(a);

}


int main() {

        int n;

        printf("Enter the number of elements to permute: ");

        scanf("%d", &n);

        jt(n);

        return 0;
```

}

Ouput :



```
~/notes/ada/lab/05_b_johnson_trotter (main*) » ./main
Enter the number of elements to permute: 3
1 2 3
1 3 2
3 1 2
3 2 1
2 3 1
2 1 3
```

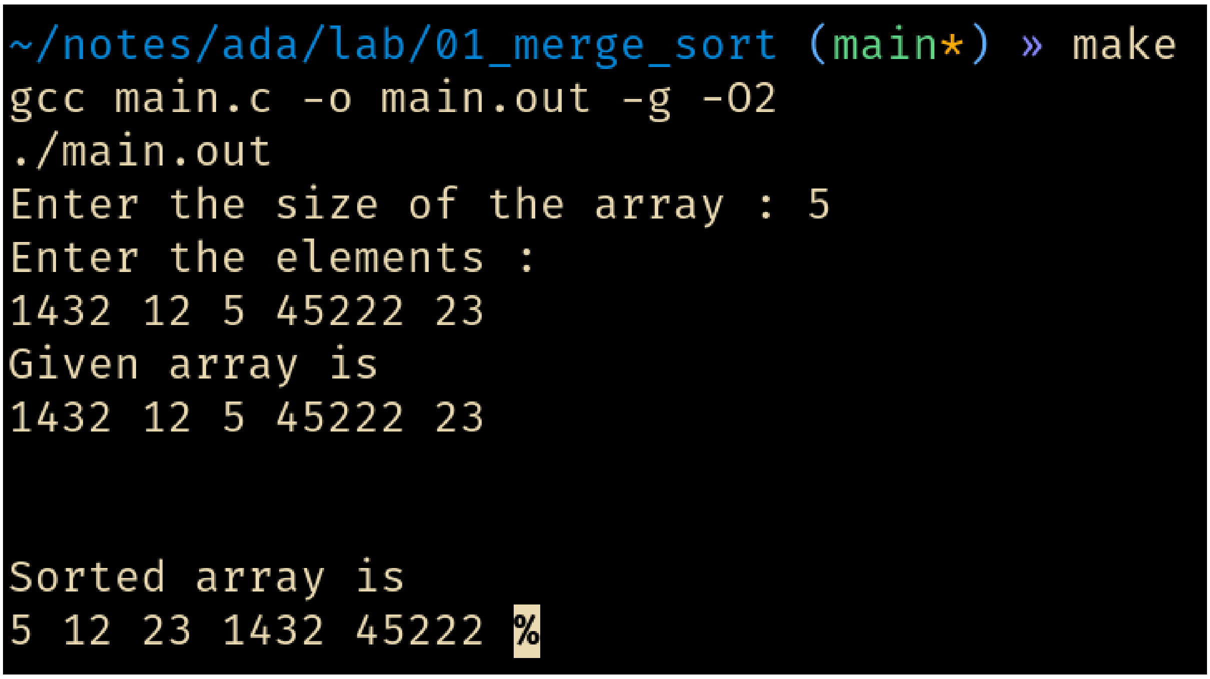
**Lab program 3:**

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 <time.h>

#include <stdlib.h>


#define MAX 1000


void merge(int *arr, int left, int mid, int right) {

        int i, j, k;

        int n1 = mid - left + 1;

        int n2 = right - mid;


        int *L = (int *) malloc(n1 * sizeof(int));

        int *R = (int *) malloc(n2 * sizeof(int));


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

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

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

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


        i = 0;

        j = 0;

        k = left;

        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++;
        }

        free(L);
        free(R);
}

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);
```

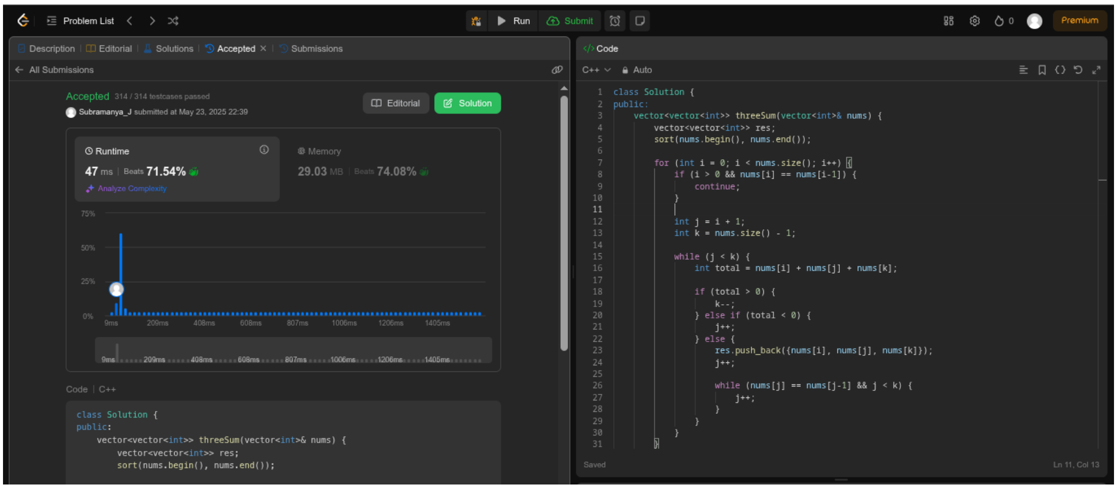
```
                merge(arr, left, mid, right);
        }
}

void sortArray(int *arr, int size) {
        mergeSort(arr, 0, size - 1);
        printf("\n\n");
}

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

int main() {
        int max = MAX;
                int *arr, size;
                printf("Enter the size of the array : ");
                scanf("%d", &size);
                arr = malloc(size*sizeof(int));

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

                printf("Given array is\n");
                printArray(arr, size);
                sortArray(arr, size);


                printf("\nSorted array is \n");
                printArray(arr, size);
        return 0;
}
```

Output :



```
~/notes/ada/lab/01_merge_sort (main*) » make
gcc main.c -o main.out -g -O2
./main.out
Enter the size of the array : 5
Enter the elements :
1432 12 5 45222 23
Given array is
1432 12 5 45222 23


Sorted array is
5 12 23 1432 45222 %
```

**LeetCode 15: 3Sum**



Accepted 314 / 314 testcases passed

Subramanya_J submitted at May 23, 2025 22:39

Runtime: 47 ms | Beats 71.54%

Memory: 29.03 MB | Beats 74.08%

Code | C++

```cpp
class Solution {
public:
    vector<vector<int>> threeSum(vector<int>& nums) {
        vector<vector<int>> res;
        sort(nums.begin(), nums.end());
```

```cpp
class Solution {
public:
    vector<vector<int>> threeSum(vector<int>& nums) {
        vector<vector<int>> res;
        sort(nums.begin(), nums.end());

        for (int i = 0; i < nums.size(); i++) {
            if (i > 0 && nums[i] == nums[i-1]) {
                continue;
            }

            int j = i + 1;
            int k = nums.size() - 1;

            while (j < k) {
                int total = nums[i] + nums[j] + nums[k];

                if (total > 0) {
                    k--;
                } else if (total < 0) {
                    j++;
                } else {
                    res.push_back({nums[i], nums[j], nums[k]});
                    j++;

                    while (nums[j] == nums[j-1] && j < k) {
                        j++;
                    }
                }
            }
        }
```

**Lab program 4:**

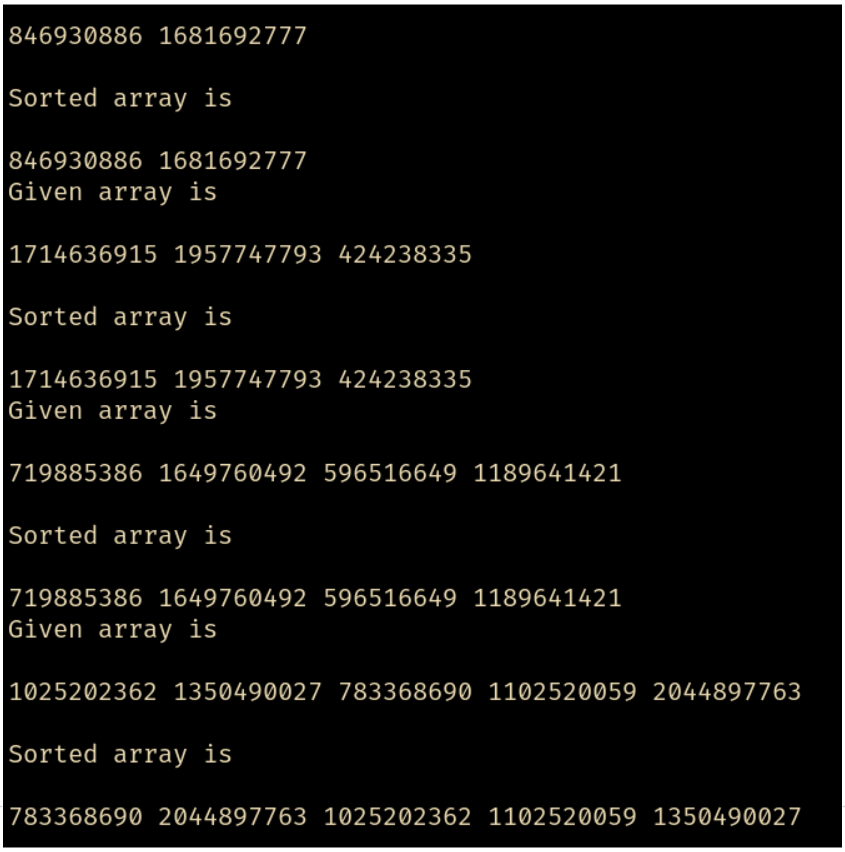
**Sort a given set of N integer elements using Quick Sort technique and compute its time taken.**

```
// quick_sort.c

#include <stdio.h>

#include <stdlib.h>

#include <time.h>


#define MAX 5


void quick_sort(int *arr, int low, int high);

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

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

int temp;


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

        if(low >= high) {

                return;

        }

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

        quick_sort(arr, low, p_index - 1);

        quick_sort(arr, p_index + 1, high);

}


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

        int pivot = arr[(low + (high - low) / 2)];

        int i = low - 1;

        for(int j = low; j < high; j++) {

                if(arr[j] <= pivot) {

                        swap(&arr[++i], &arr[j]);

                }

        }
```

```
	swap(&arr[i+1], &arr[high]);
	return i + 1;
}

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

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

int main() {
	int max = MAX;
	clock_t start, end;
	FILE *fp = fopen("data.txt", "w");
	for(int i = 1; i <= max; i++) {
//		printf("%d elements\n", i);
		int *arr, size = i;
//		printf("Enter the size of the array : ");
//		scanf("%d", &size);
		arr = malloc(size*sizeof(int));

//		printf("Enter the elements : \n");
		for(int j = 0; j < size; j++) {
//			scanf("%d", &arr[j]);
```

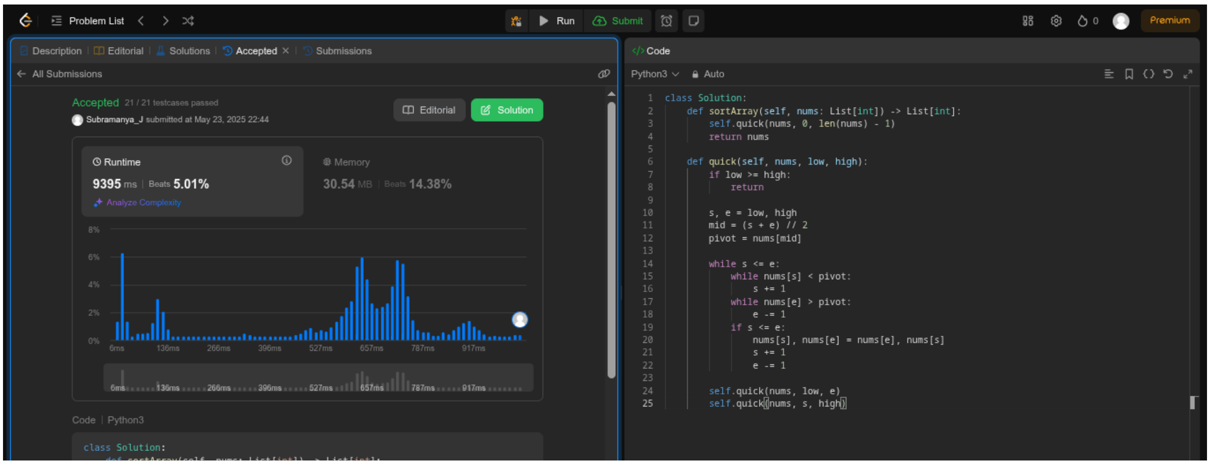
```
                        arr[j] = rand();
                }

//              printf("\n-----------------------------------\n\nSize : %d\n", i);
                printf("Given array is\n");
                display_array(arr, size);

                start = clock();
                quick_sort(arr, 0, size-1);
                end = clock();

                printf("\nSorted array is \n");
                display_array(arr, size);
                fprintf(fp, "%d, %d\n", (int) (end - start), i);
        }
        fclose(fp);
        return 0;
}
```

Output :



```
846930886 1681692777

Sorted array is

846930886 1681692777
Given array is

1714636915 1957747793 424238335

Sorted array is

1714636915 1957747793 424238335
Given array is

719885386 1649760492 596516649 1189641421

Sorted array is

719885386 1649760492 596516649 1189641421
Given array is

1025202362 1350490027 783368690 1102520059 2044897763

Sorted array is

783368690 2044897763 1025202362 1102520059 1350490027
```

## LeetCode 912: Sort an Array



Accepted 21 / 21 testcases passed

Subramanya_J submitted at May 23, 2025 22:44

Runtime 9395 ms | Beats 5.01%

Analyze Complexity

Memory 30.54 MB | Beats 14.38%

Code | Python3

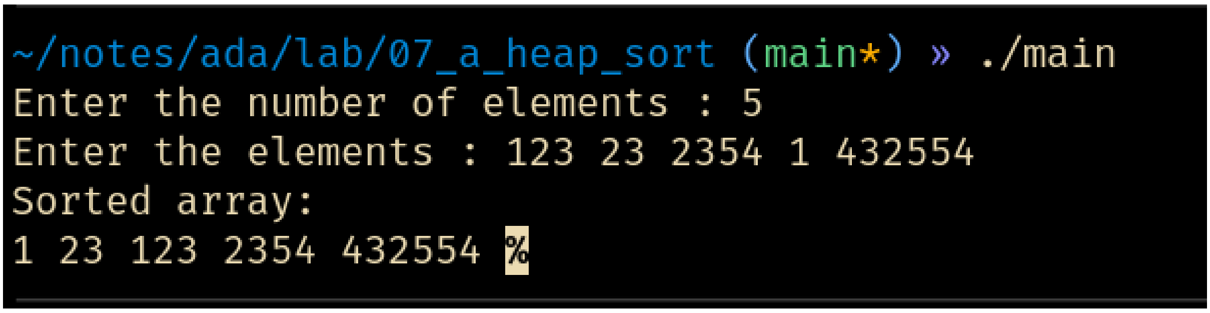
```python
class Solution:
    def sortArray(self, nums: List[int]) -> List[int]:
        self.quick(nums, 0, len(nums) - 1)
        return nums

    def quick(self, nums, low, high):
        if low >= high:
            return

        s, e = low, high
        mid = (s + e) // 2
        pivot = nums[mid]

        while s <= e:
            while nums[s] < pivot:
                s += 1
            while nums[e] > pivot:
                e -= 1
            if s <= e:
                nums[s], nums[e] = nums[e], nums[s]
                s += 1
                e -= 1

        self.quick(nums, low, e)
        self.quick(nums, s, high)
```

**Lab program 5:**

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

```
#include <stdio.h>


void heapify(int arr[], int n, int i) {

        int largest = i;

        int left = 2 * i + 1;

        int right = 2 * i + 2;


        if (left < n && arr[left] > arr[largest]) {

                largest = left;

        }


        if (right < n && arr[right] > arr[largest]) {

                largest = right;

        }


        if (largest != i) {

                int temp = arr[i];

                arr[i] = arr[largest];

                arr[largest] = temp;

                heapify(arr, n, largest);

        }

}


void heapSort(int arr[], int n) {

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

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

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

        heapSort(arr, n);

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

**Output :**

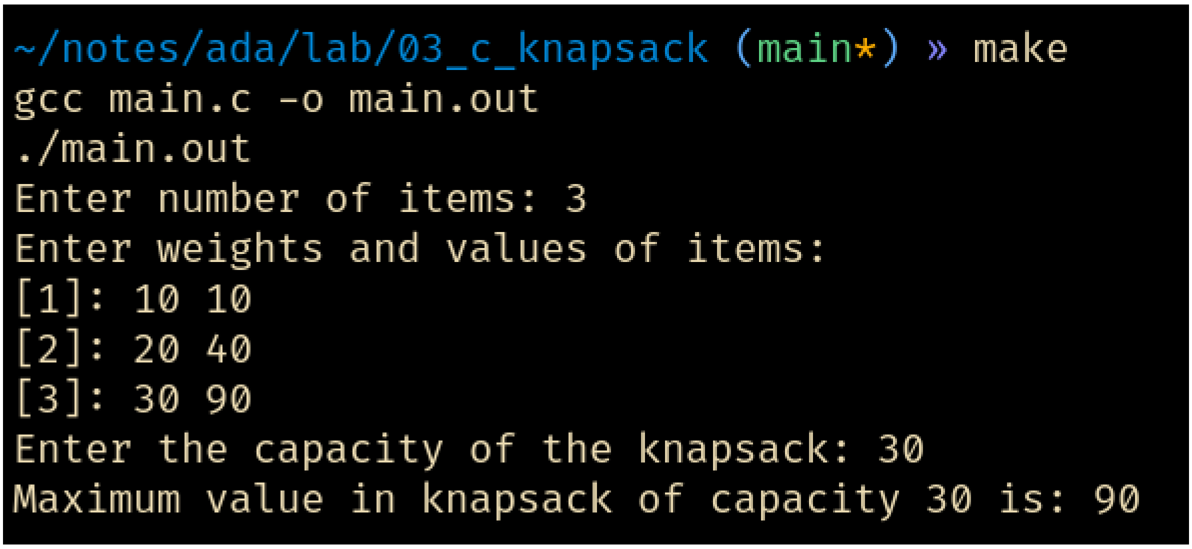


```
~/notes/ada/lab/07_a_heap_sort (main*) » ./main
Enter the number of elements : 5
Enter the elements : 123 23 2354 1 432554
Sorted array:
1 23 123 2354 432554 %
```

**Lab program 6:**

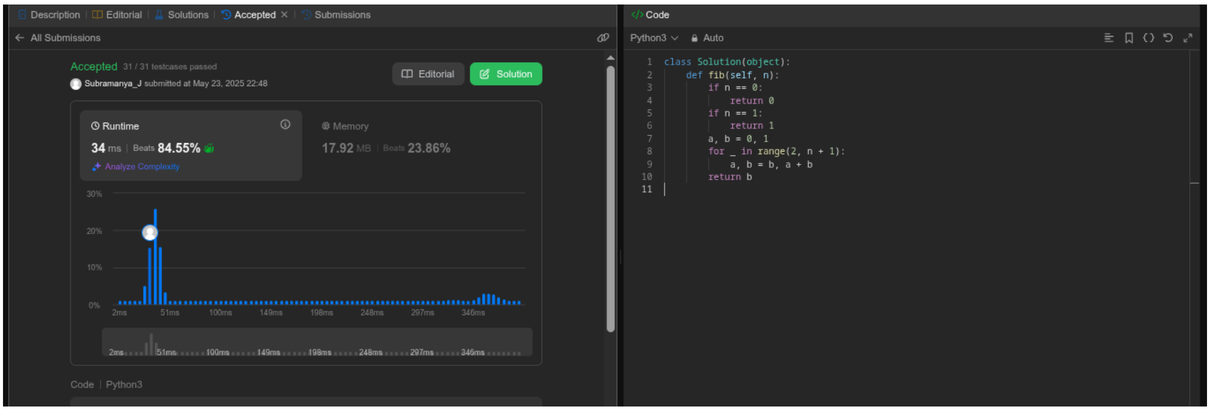
**Implement 0/1 Knapsack problem using dynamic programming.**

```
#include <stdio.h>

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

int knapsack(int weights[], int values[], int n, int capacity) {
    int dp[n + 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 number of items: ");
    scanf("%d", &n);
    int weights[n], values[n];

    printf("Enter weights and values of items:\n");
    for (int i = 0; i < n; i++) {
      printf("[%d]: ", i + 1);
      scanf("%d %d", &weights[i], &values[i]);
    }

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

    int maxValue = knapsack(weights, values, n, capacity);
    printf("Maximum value in knapsack of capacity %d is: %d\n", capacity, maxValue);

    return 0;
}
```



```
~/notes/ada/lab/03_c_knapsack (main*) » make
gcc main.c -o main.out
./main.out
Enter number of items: 3
Enter weights and values of items:
[1]: 10 10
[2]: 20 40
[3]: 30 90
Enter the capacity of the knapsack: 30
Maximum value in knapsack of capacity 30 is: 90
```

## LeetCode 509: Fibonacci Number



Accepted 31 / 31 testcases passed

Subramanya_J submitted at May 23, 2025 22:48

Runtime: 34 ms | Beats 84.55%

Memory: 17.92 MB | Beats 23.86%

Code | Python3

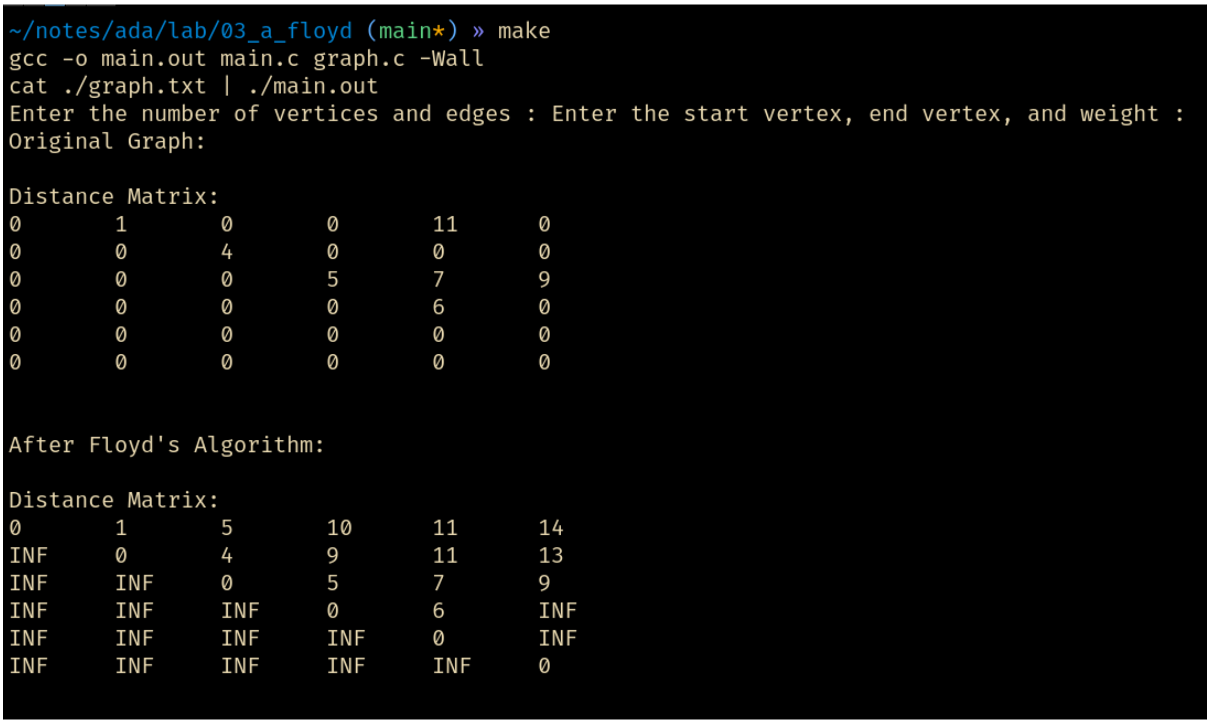
```python
class Solution(object):
    def fib(self, n):
        if n == 0:
            return 0
        if n == 1:
            return 1
        a, b = 0, 1
        for _ in range(2, n + 1):
            a, b = b, a + b
        return b
```

**Lab program 7:**

**Implement All Pair Shortest paths problem using Floyd's algorithm.**

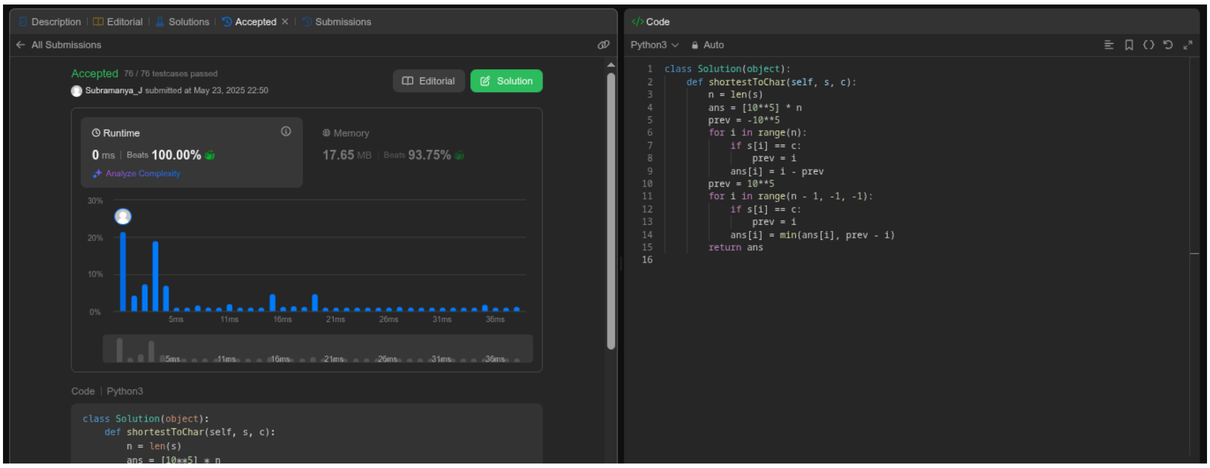
```
#include <stdio.h>

#include <limits.h>

#include "graph.h"


void floyd(int **graph, int vertices) {

    // Initialize: Convert 0s to INT_MAX, but not the diagonal

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

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

            if(i != j && graph[i][j] == 0) {

                graph[i][j] = INT_MAX;

            }

        }

    }


    // Floyd–Warshall core logic

    for(int k = 0; k < vertices; k++) {

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

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

                if (graph[i][k] != INT_MAX && graph[k][j] != INT_MAX
&&

                    graph[i][k] + graph[k][j] < graph[i][j]) {

                    graph[i][j] = graph[i][k] + graph[k][j];

                }

            }

        }
        }
}

void graph_print(int **graph, int vertices) {
        printf("\nDistance Matrix:\n");
        for(int i = 0; i < vertices; i++) {
                for(int j = 0; j < vertices; j++) {
                        if (graph[i][j] == INT_MAX)
                                printf("INF\t");
                        else
                                printf("%d\t", graph[i][j]);
                }
                putchar('\n');
        }
        putchar('\n');
}

int main() {
        int edges, vertices;
        int **graph = graph_create(&vertices, &edges);
        printf("\nOriginal Graph:\n");
        graph_print(graph, vertices);
        floyd(graph, vertices);
        printf("\nAfter Floyd's Algorithm:\n");
        graph_print(graph, vertices);
        return 0;
}
```

## Output :



```
~/notes/ada/lab/03_a_floyd (main*) » make
gcc -o main.out main.c graph.c -Wall
cat ./graph.txt | ./main.out
Enter the number of vertices and edges : Enter the start vertex, end vertex, and weight :
Original Graph:

Distance Matrix:
0       1       0       0       11      0
0       0       4       0       0       0
0       0       0       5       7       9
0       0       0       0       6       0
0       0       0       0       0       0
0       0       0       0       0       0


After Floyd's Algorithm:

Distance Matrix:
0       1       5       10      11      14
INF     0       4       9       11      13
INF     INF     0       5       7       9
INF     INF     INF     0       6       INF
INF     INF     INF     INF     0       INF
INF     INF     INF     INF     INF     0
```

## LeetCode 821: Shortest Distance to a Character



Accepted 76 / 76 testcases passed

Subramanya_J submitted at May 23, 2025 22:50

Runtime 0 ms | Beats 100.00%

Memory 17.65 MB | Beats 93.75%

Code | Python3

```python
class Solution(object):
    def shortestToChar(self, s, c):
        n = len(s)
        ans = [10**5] * n
        prev = -10**5
        for i in range(n):
            if s[i] == c:
                prev = i
            ans[i] = i - prev
        prev = 10**5
        for i in range(n - 1, -1, -1):
            if s[i] == c:
                prev = i
            ans[i] = min(ans[i], prev - i)
        return ans
```

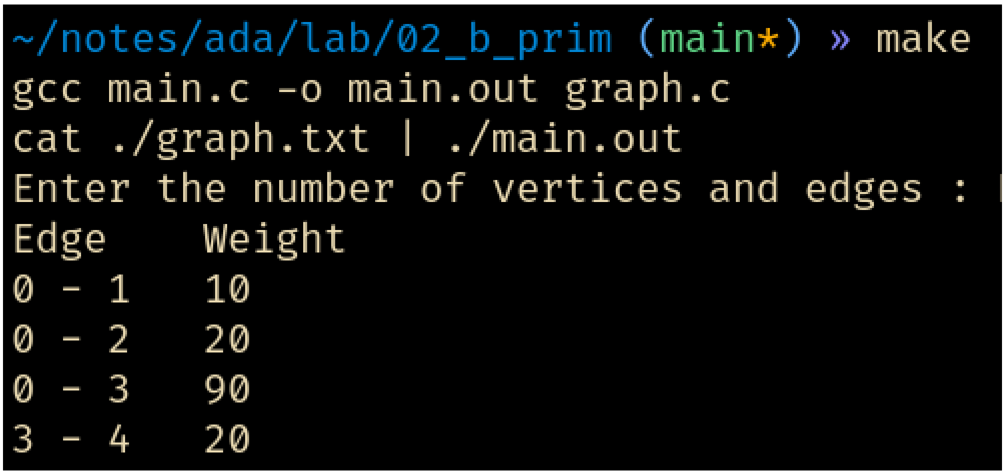
**Lab program 8:**

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

```
// prim.c

#include "graph.h"

#include <stdio.h>

#include <limits.h>


int minKey(int key[], int mstSet[], int vertices) {

        int min = INT_MAX, min_index;

        for (int v = 0; v < vertices; v++) {

                if (!mstSet[v] && key[v] < min) {

                        min = key[v], min_index = v;

                }

        }

        return min_index;

}


void primMST(int **graph, int vertices) {

        int parent[vertices];  // Stores constructed MST

        int key[vertices];     // Used to pick minimum weight edge

        int mstSet[vertices];  // To represent set of vertices included in MST


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

                key[i] = INT_MAX;

                mstSet[i] = 0;

        }


        key[0] = 0;      // First vertex is picked first

        parent[0] = -1;  // First node is always the root


        for (int count = 0; count < vertices - 1; count++) {
```

```
        int u = minKey(key, mstSet, vertices);
        mstSet[u] = 1;

        for (int v = 0; v < vertices; v++) {
            if (graph[u][v] && !mstSet[v] && graph[u][v] < key[v]) {
                parent[v] = u, key[v] = graph[u][v];
            }
        }
    }

    printf("\nEdge \tWeight\n");
    for (int i = 1; i < vertices; i++) {
        printf("%d - %d \t%d\n", parent[i], i, graph[i][parent[i]]);
    }
}

int main() {
    int edges, vertices, **graph = graph_create(&vertices, &edges);
    primMST(graph, vertices);
    return 0;
}
```

**Output :**

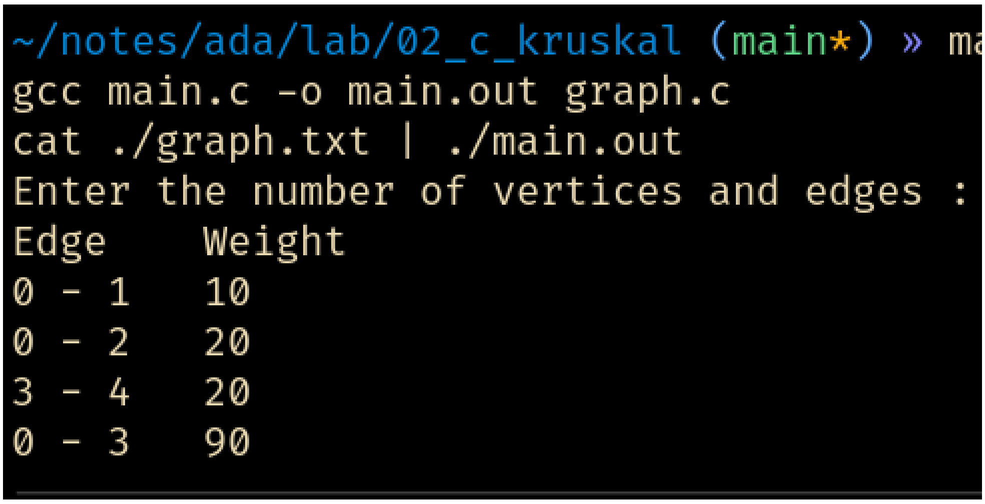


```
~/notes/ada/lab/02_b_prim (main*) » make
gcc main.c -o main.out graph.c
cat ./graph.txt | ./main.out
Enter the number of vertices and edges : 
Edge    Weight
0 - 1   10
0 - 2   20
0 - 3   90
3 - 4   20
```

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

```
// kruskal.c

#include "graph.h"

#include <stdio.h>

#include <stdlib.h>


struct Edge {

        int src, dest, weight;

};


struct subset {

        int parent;

        int rank;

};


int find(struct subset subsets[], int i) {

        if (subsets[i].parent != i) {

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

        }

        return subsets[i].parent;

}


void Union(struct subset subsets[], int x, int y) {

        int xroot = find(subsets, x);

        int yroot = find(subsets, y);


        if (subsets[xroot].rank < subsets[yroot].rank) {

                subsets[xroot].parent = yroot;

        } else if (subsets[xroot].rank > subsets[yroot].rank) {

                subsets[yroot].parent = xroot;
        } else {
                subsets[yroot].parent = xroot;
                subsets[xroot].rank++;
        }
}

int compare(const void *a, const void *b) {
        struct Edge *a1 = (struct Edge *) a;
        struct Edge *b1 = (struct Edge *) b;
        return a1->weight - b1->weight;
}

void kruskalMST(int **graph, int vertices) {
        int edgeCount = 0;
        struct Edge result[vertices];

        struct Edge *edges = malloc(vertices * vertices * sizeof(struct Edge));
        int edgeIndex = 0;
        for (int i = 0; i < vertices; i++) {
                for (int j = i + 1; j < vertices; j++) {
                        if (graph[i][j] != 0) {
                                edges[edgeIndex].src = i;
                                edges[edgeIndex].dest = j;
                                edges[edgeIndex].weight = graph[i][j];
                                edgeIndex++;
                        }
                }
        }

        qsort(edges, edgeIndex, sizeof(edges[0]), compare);

        struct subset *subsets = malloc(vertices * sizeof(struct subset));
        for (int v = 0; v < vertices; v++) {
                subsets[v].parent = v;
                subsets[v].rank = 0;
        }

        for (int i = 0; i < edgeIndex; i++) {
                int x = find(subsets, edges[i].src);
                int y = find(subsets, edges[i].dest);

                if (x != y) {
                        result[edgeCount++] = edges[i];
                        Union(subsets, x, y);
                }
        }

        printf("\nEdge \tWeight\n");
        for (int i = 0; i < edgeCount; i++) {
                printf("%d - %d \t%d\n", result[i].src, result[i].dest, result[i].weight);
        }

        free(edges);
        free(subsets);
}

int main() {
        int edges, vertices, **graph = graph_create(&vertices, &edges);
        kruskalMST(graph, vertices);
        return 0;
}
```

**Ouput :**



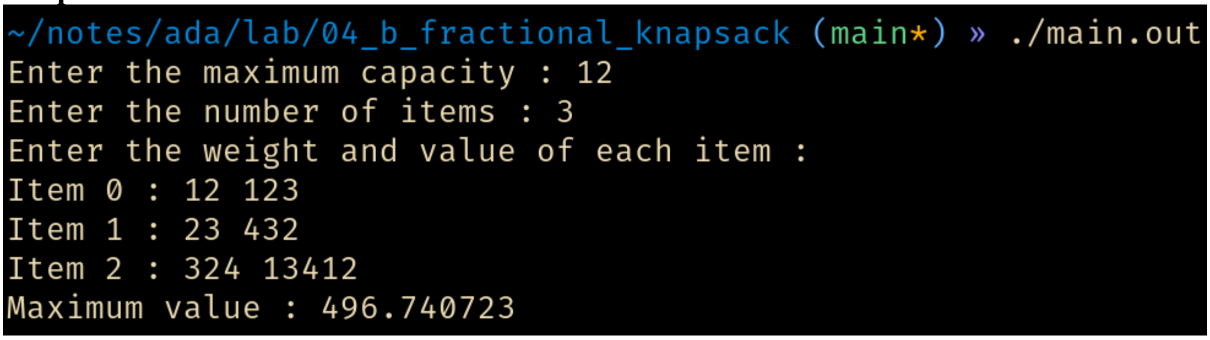
```
~/notes/ada/lab/02_c_kruskal (main*) » ma
gcc main.c -o main.out graph.c
cat ./graph.txt | ./main.out
Enter the number of vertices and edges :
Edge    Weight
0 - 1   10
0 - 2   20
3 - 4   20
0 - 3   90
```

**Lab program 9:**

**Implement Fractional Knapsack using Greedy technique.**

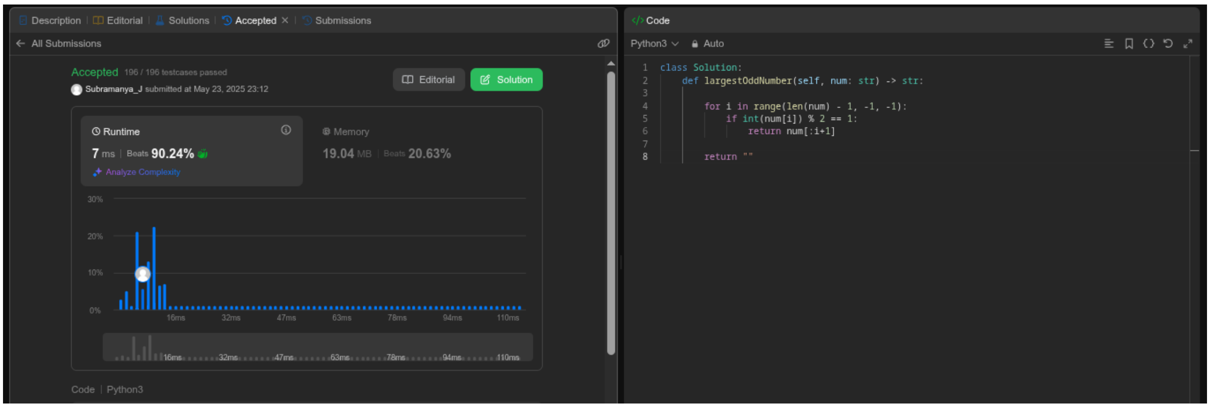
```
/**
 * Algorithm :
 * function fractionalKnapsack(W, value[], weight[], n):
   items = []
   for i from 0 to n-1:
      ratio = value[i] / weight[i]
      items.append((ratio, value[i], weight[i]))

   sort items by ratio descending

   totalValue = 0.0
   for each (ratio, val, wt) in items:
      if W >= wt:
         totalValue += val
         W -= wt
      else:
         totalValue += ratio * W
         break

   return totalValue
*/

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

struct Item {
	int weight;
	int value;
	float ratio;
};

int itemcmp(const void *x, const void *y) {
	float ret = ((struct Item *) x )->ratio - ((struct Item *) y)->ratio;
	if(ret > 0) {return -1;}
	if(ret < 0) {return 1;}
	return 0;
}

struct Item *get_items(int *n) {
	printf("Enter the number of items : ");
	scanf("%d", n);

	struct Item *arr = calloc(*n, sizeof(struct Item));

	printf("Enter the weight and value of each item : \n");
	for(int i = 0; i < *n; i++) {
        printf("Item %d : ", i);
        scanf("%d %d", &arr[i].weight, &arr[i].value);
        arr[i].ratio = (float) arr[i].value / arr[i].weight;
    }
    qsort(arr, *n, sizeof(struct Item), itemcmp);
    return arr;
}

float calculate_max(struct Item *arr, int n, int max) {
    float total = 0;
    for(int i = 0; i < n; i++) {
        if(max == 0) {
            break;
        }
        else if(arr[i].weight < max) {
            total += arr[i].value;
            max -= arr[i].weight;
        }
        else if(arr[i].weight > max) {
            total += arr[i].ratio * max;
            break;
        }
    }
    return total;
}

int main() {
    int n, max;
    printf("Enter the maximum capacity : ");
    scanf("%d", &max);
    struct Item * arr = get_items(&n);
    float max_val = calculate_max(arr, n, max);
    printf("Maximum value : %f\n", max_val);

    free(arr);

    return 0;
}
```

**Output :**



```
~/notes/ada/lab/04_b_fractional_knapsack (main*) » ./main.out
Enter the maximum capacity : 12
Enter the number of items : 3
Enter the weight and value of each item :
Item 0 : 12 123
Item 1 : 23 432
Item 2 : 324 13412
Maximum value : 496.740723
```

## LeetCode 1903: Largest Odd Number in a String



Accepted 196 / 196 testcases passed

Subramanya_J submitted at May 23, 2025 23:12

Runtime

7 ms | Beats 90.24%

Memory

19.04 MB | Beats 20.63%

```python
class Solution:
    def largestOddNumber(self, num: str) -> str:

        for i in range(len(num) - 1, -1, -1):
            if int(num[i]) % 2 == 1:
                return num[:i+1]

        return ""
```

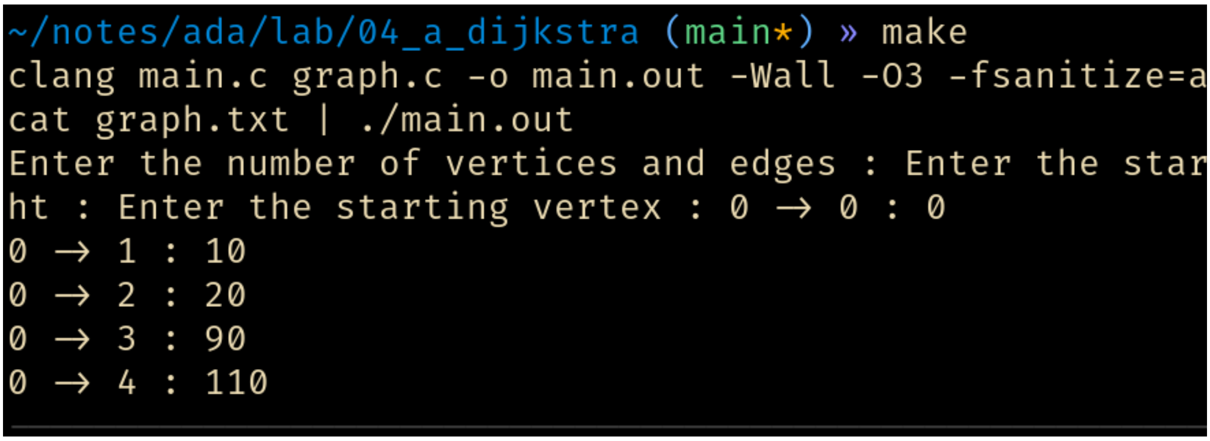
**Lab program 10:**

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

```
#include <stdio.h>

#include <limits.h>

#include <stdlib.h>

#include <stdbool.h>

#include "graph.h"


int min_index(int dist[], bool visited[], int n) {

        int min_in = -1, min = INT_MAX;

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

                if (!visited[i]

                        && dist[i] < min) {

                        min = dist[i];

                        min_in = i;

                }

        }

        return min_in;

}


void dijkstra(int **graph, int vertices, int start) {

        bool visited[vertices];

        int dist[vertices];


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

                visited[i] = false;
        dist[i] = INT_MAX;
    }

    dist[start] = 0;

    int curr;
    for(int i = 0; i < vertices - 1; i++) {
        curr = min_index(dist, visited, vertices);
        visited[curr] = true;

        for(int i = 0; i < vertices; i++) {
            if (
                !visited[i]
                && graph[curr][i]
                && dist[curr] != INT_MAX
                && dist[curr] + graph[i][curr] < dist[i]
            ) {
                dist[i] = dist[curr] + graph[curr][i];
            }
        }
    }

    for(int i = 0; i < vertices; i++) {
        printf("%d -> %d : %d\n", start, i, dist[i]);
    }
}
```

```
int main() {
    int vertices, edges,
    **graph = graph_create(&vertices, &edges);
    printf("Enter the starting vertex : ");
    int start;
    scanf("%d", &start);
    dijkstra(graph, vertices, start);


    graph_free(graph, vertices);


    return 0;
}
```

**Output :**



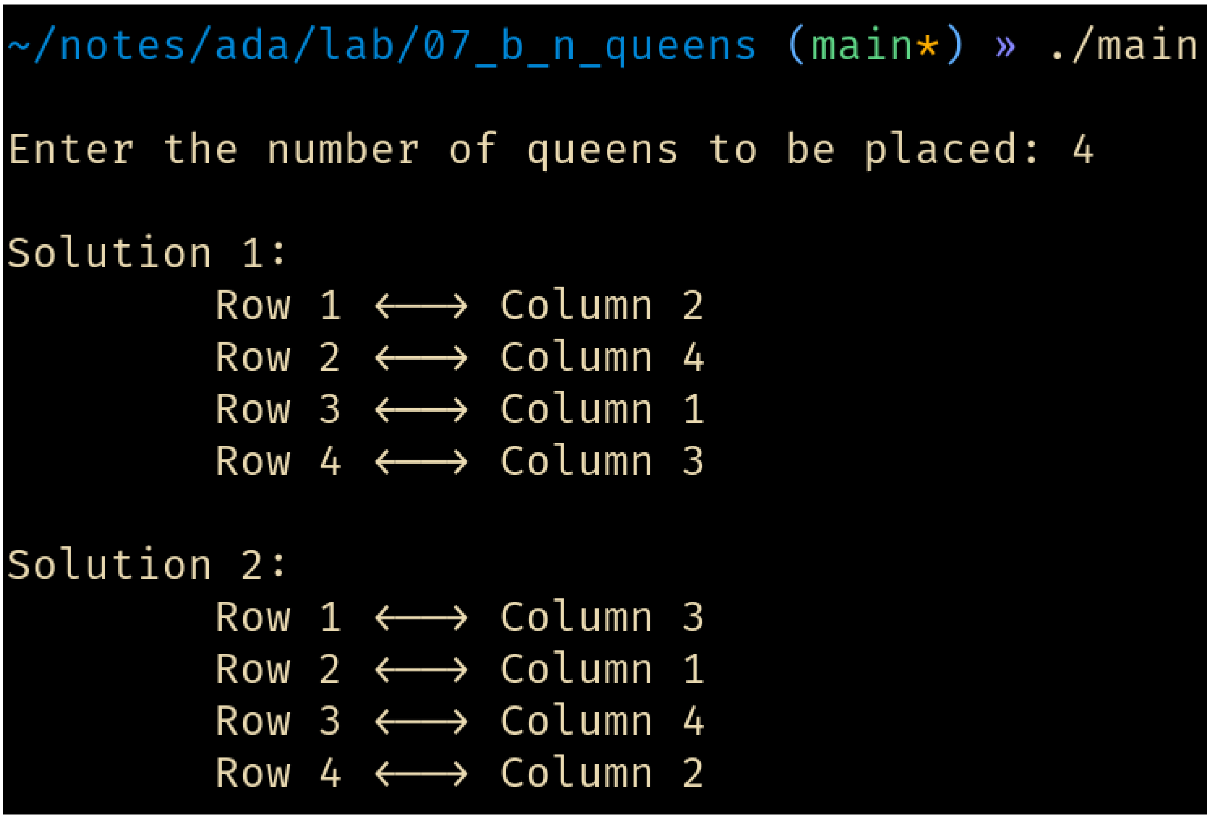
```
~/notes/ada/lab/04_a_dijkstra (main*) » make
clang main.c graph.c -o main.out -Wall -O3 -fsanitize=a
cat graph.txt | ./main.out
Enter the number of vertices and edges : Enter the star
ht : Enter the starting vertex : 0 → 0 : 0
0 → 1 : 10
0 → 2 : 20
0 → 3 : 90
0 → 4 : 110
```

**Lab program 11:**

**Implement “N-Queens Problem” using Backtracking.**

```
#include <stdio.h>

#include <stdlib.h>

#include <math.h>


int x[20], count = 1;


void queens(int k, int n);

int place(int k, int j);


int main() {

   int n;

   printf("\nEnter the number of queens to be placed: ");

   scanf("%d", &n);

   if (n < 4) {

      printf("No solutions for N = %d\n", n);

   } else {

      queens(1, n);

   }

   return 0;

}


void queens(int k, int n) {

   int i, j;

   for (j = 1; j <= n; j++) {

      if (place(k, j)) {
            x[k] = j;
            if (k == n) {
                printf("\nSolution %d:\n", count);
                count++;
                for (i = 1; i <= n; i++) {
                    printf("\tRow %d <--> Column %d\n", i, x[i]);
                }
            } else {
                queens(k + 1, n);
            }
        }
    }
}

int place(int k, int j) {
    int i;
    for (i = 1; i < k; i++) {
        if ((x[i] == j) || (abs(x[i] - j)) == abs(i - k)) {
            return 0;
        }
    }
    return 1;
}
```

**Output :**



```
~/notes/ada/lab/07_b_n_queens (main*) » ./main

Enter the number of queens to be placed: 4

Solution 1:
        Row 1 ⟷ Column 2
        Row 2 ⟷ Column 4
        Row 3 ⟷ Column 1
        Row 4 ⟷ Column 3

Solution 2:
        Row 1 ⟷ Column 3
        Row 2 ⟷ Column 1
        Row 3 ⟷ Column 4
        Row 4 ⟷ Column 2
```

**Auxillary file graph.c**

```
// graph.c

#include "graph.h"

#include <stdlib.h>

#include <stdio.h>


int **graph_create(int *vertices, int *edges) {

        printf("Enter the number of vertices and edges : ");

        scanf("%d %d", vertices, edges);

        int **arr = calloc(*vertices, sizeof(int *));

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

                arr[i] = calloc(*vertices, sizeof(int));

        }

        int start, end, weight;

        printf("Enter the start vertex, end vertex, and weight : ");

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

                scanf("%d %d %d", &start, &end, &weight);

                arr[start][end] = arr[end][start] = weight;

        }



        return arr;

}
```