# VISVESVARAYA TECHNOLOGICAL UNIVERSITY

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

**LAB REPORT**
**on**

# ANALYSIS AND DESIGN OF ALGORITHMS

***Submitted by***

**CREVAN NEIL FERNANDES (1BM23CS082)**

***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**
**Feb 2025 to Jun 2025**

# B. M. S. College of Engineering,

**Bull Temple Road, Bangalore 560019**
(Affiliated To Visvesvaraya Technological University, Belgaum)

## Department of Computer Science and Engineering

## CERTIFICATE

This is to certify that the Lab work entitled “**ANALYSIS AND DESIGN OF ALGORITHMS(23CS4PCADA)**” carried out by **CREVAN NEIL FERNANDES (1BM23CS082) ,** 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 year 2025. The Lab report has been approved as it satisfies the academic requirements in respect of **ANALYSIS AND DESIGN OF ALGORITHMS(23CS4PCADA)** work prescribed for the said degree.

| | |
|---|---|
| **Prof Sowmya T** | **Dr. Kavitha Sooda** |
| 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 algorithm 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-1

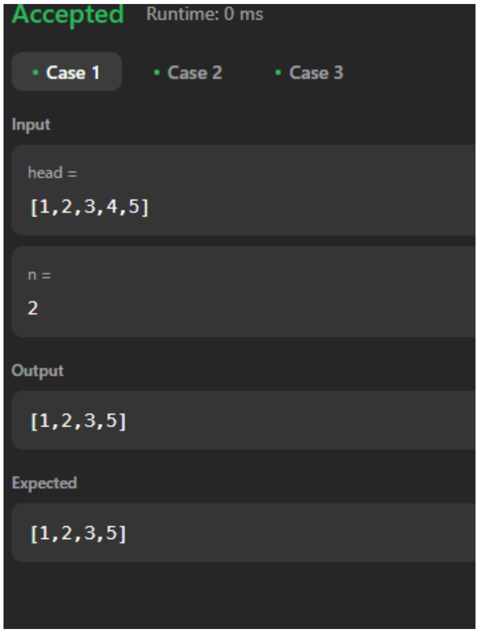
## Question- 1:

LeetCode Problem - Remove Nth Node from end of a List

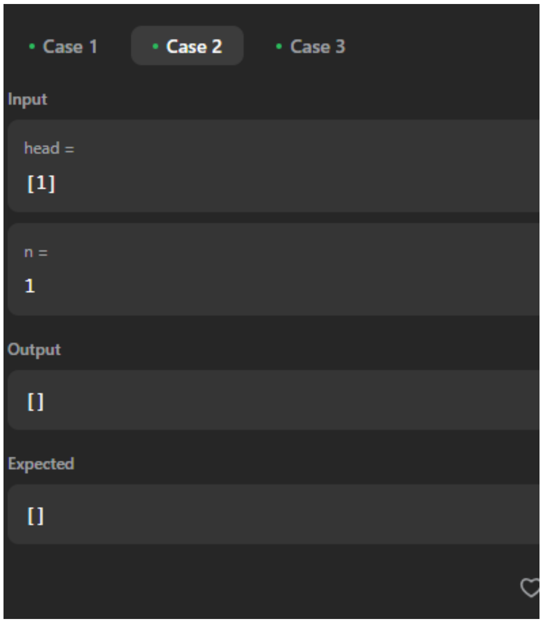
## Code:

```
    struct ListNode* removeNthFromEnd(struct ListNode* head,
int n) {

    struct ListNode* temp1=(head);

    struct ListNode* temp2=(head);


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

        temp1=temp1->next;

    }

    if (head->next==NULL){

        return NULL;

    }

    if(temp1==NULL){

        head=head->next;

        return head;


    }




    while(temp1->next!=NULL){

        temp1=temp1->next;

        temp2=temp2->next;

    }

    temp2->next=temp2->next->next;
    return head;

}
```

## Result:



```
Accepted   Runtime: 0 ms
• Case 1   • Case 2   • Case 3
Input
head =
[1,2,3,4,5]
n =
2
Output
[1,2,3,5]
Expected
[1,2,3,5]
```



```
• Case 1   • Case 2   • Case 3
Input
head =
[1]
n =
1
Output
[]
Expected
[]
```

## Question -2:

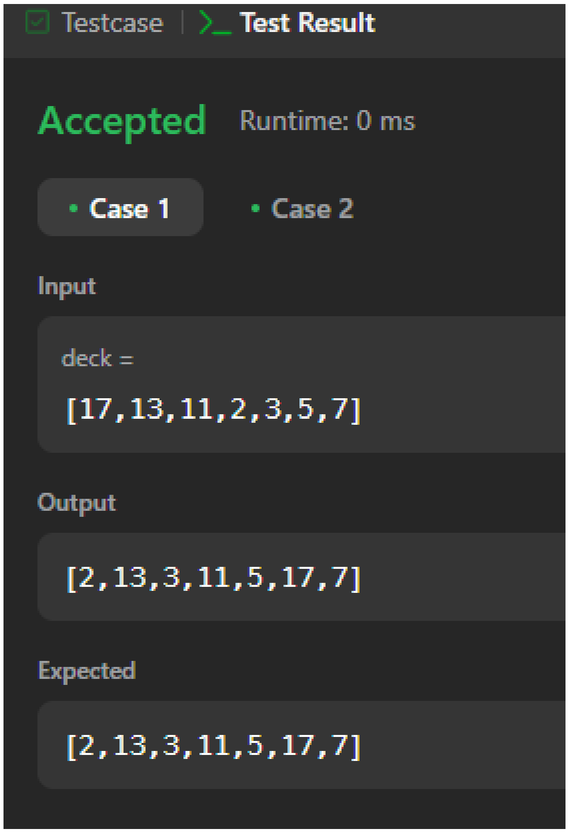
LeetCode Problem- Reveal Cards in Increasing Order

## CODE:

```
int cmp(const void *a , const void *b )
```

```
{
    return *(int*)a - *(int*)b;
}



void PushBack(int deck[static 1] , int size , int currFull)
{
    //Swap
    int end = deck[size-1];
    deck[size-currFull-1] = end;

    //PushBack
    for(int i= size-1 ; i>=(size-currFull); i--)
    {
        deck[i] = deck[i-1];
    }
}



int*
deckRevealedIncreasing(
        int deck[static 1],
        int deckSize,
        int* returnSize // Reference
    )
{
    int *res = malloc(sizeof(int) * deckSize);
    *returnSize = deckSize;

    qsort(deck,deckSize , sizeof(int) , cmp);

    res[deckSize-1] = deck[deckSize-1];
    int currFull = 1;

    for(int i = deckSize - 2 ; i>= 0 ; --i )
    {
        PushBack(res,deckSize,currFull);
        currFull++;
        res[deckSize-currFull] = deck[i];
    }
    return res;

}
```

**OUTPUT:**



Accepted Runtime: 0 ms

Input

deck =

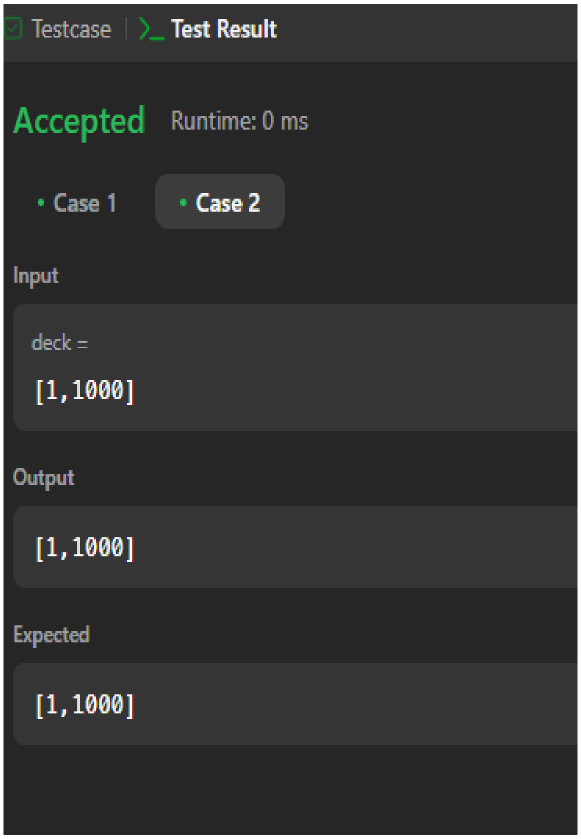
```
[17,13,11,2,3,5,7]
```

Output

```
[2,13,3,11,5,17,7]
```

Expected

```
[2,13,3,11,5,17,7]
```



Accepted Runtime: 0 ms

Input

deck =

```
[1,1000]
```

Output

```
[1,1000]
```

Expected

```
[1,1000]
```

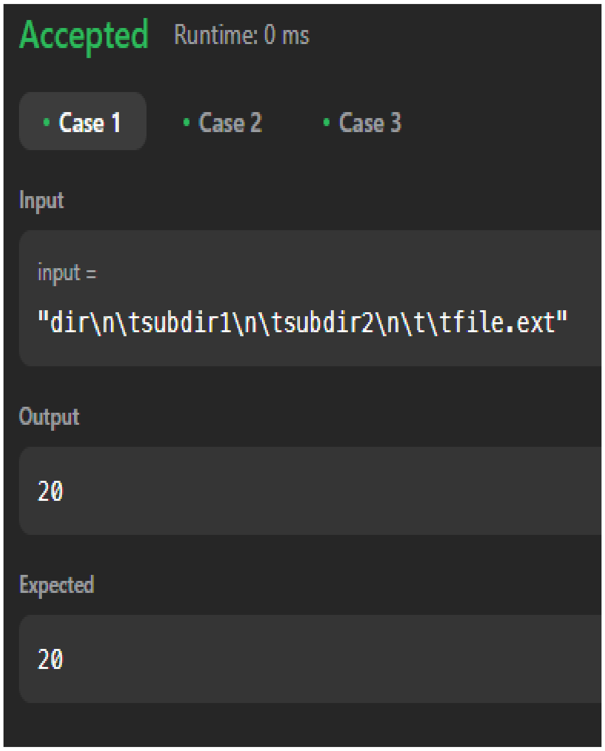
## Question 3:

LeetCode Problem: Longest Absolute File Path.

### Code:

```
int lengthLongestPath(char *input) {
    char *token;
    int len[100] = {0};
    int ans = 0;
    token = strtok(input, "\n");
    while (token != NULL) {
        int l = 0;
        while (token[l] == '\t') {
            l++;
        }

        len[l] = strlen(token) - l;

        if (strchr(token, '.')) {
            int totalLength = 0;
            for (int i = 0; i <= l; i++) {
                totalLength += len[i];
            }
            ans = (ans > totalLength + l) ? ans : totalLength + l;
        }

        token = strtok(NULL, "\n");
    }

    return ans;
}
```

## Output:


Accepted Runtime: 0 ms

Input

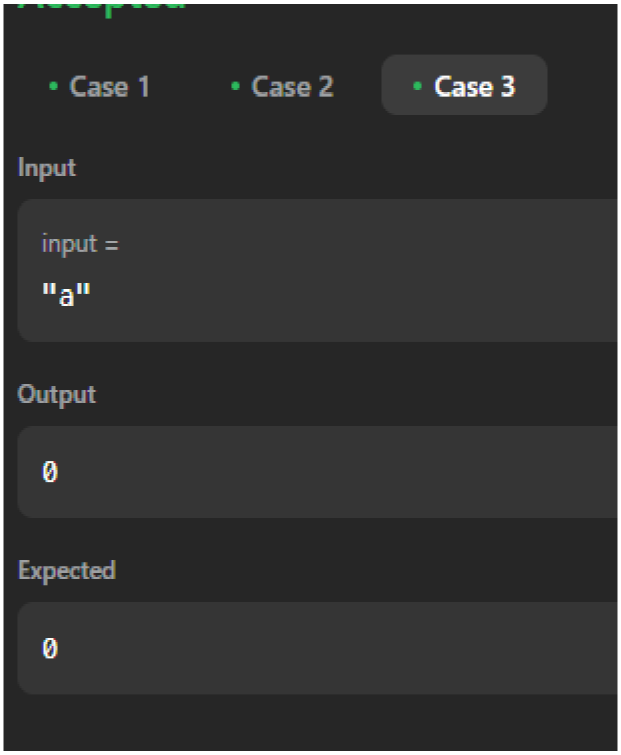
```
input =
"dir\n\tsubdir1\n\tsubdir2\n\t\tfile.ext"
```

Output

```
20
```

Expected

```
20
```



Input

```
input =
"a"
```

Output

```
0
```

Expected

```
0
```

## LAB 2

### QUESTION 1:

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

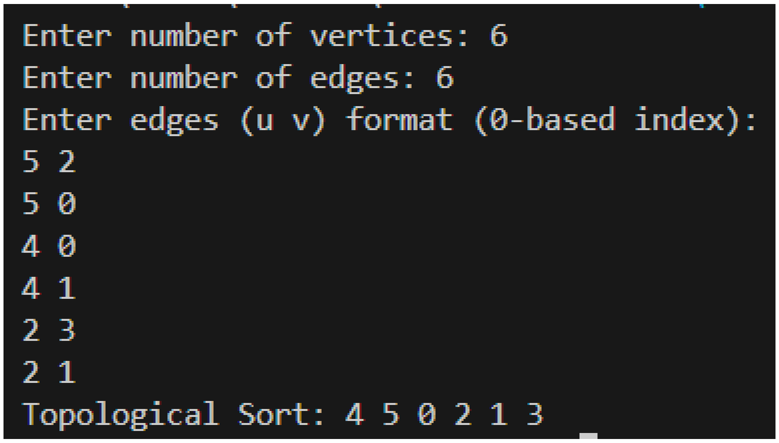
### CODE:

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

#define MAX_VERTICES 100
```

```
typedef struct Graph {
    int vertices;
    int adj[MAX_VERTICES][MAX_VERTICES];
    int in_degree[MAX_VERTICES];
} Graph;

void initGraph(Graph *g, int vertices) {
    g->vertices = vertices;
    for (int i = 0; i < vertices; i++) {
        g->in_degree[i] = 0;
        for (int j = 0; j < vertices; j++) {
            g->adj[i][j] = 0;
        }
    }
}

void addEdge(Graph *g, int u, int v) {
    g->adj[u][v] = 1;
    g->in_degree[v]++;
}

void topologicalSort(Graph *g) {
    int queue[MAX_VERTICES], front = 0, rear = 0;
    int topOrder[MAX_VERTICES];
    int index = 0;
```

```
for (int i = 0; i < g->vertices; i++) {
    if (g->in_degree[i] == 0) {
        queue[rear++] = i;
    }
}

while (front < rear) {
    int u = queue[front++];
    topOrder[index++] = u;

    for (int v = 0; v < g->vertices; v++) {
        if (g->adj[u][v] == 1) {
            g->in_degree[v]--;
            if (g->in_degree[v] == 0) {
                queue[rear++] = v;
            }
        }
    }
}

if (index != g->vertices) {
    printf("Graph contains a cycle, topological sorting is not possible.\n");
    return;
}

printf("Topological Sort: ");
for (int i = 0; i < g->vertices; i++) {
        printf("%d ", topOrder[i]);
    }
    printf("\n");
}

int main() {
    Graph g;
    int vertices, edges, u, v;

    printf("Enter number of vertices: ");
    scanf("%d", &vertices);
    initGraph(&g, vertices);

    printf("Enter number of edges: ");
    scanf("%d", &edges);

    printf("Enter edges (u v) format (0-based index):\n");
    for (int i = 0; i < edges; i++) {
        scanf("%d %d", &u, &v);
        addEdge(&g, u, v);
    }

    topologicalSort(&g);

    return 0;
}
```

## OUTPUT:



```
Enter number of vertices: 6
Enter number of edges: 6
Enter edges (u v) format (0-based index):
5 2
5 0
4 0
4 1
2 3
2 1
Topological Sort: 4 5 0 2 1 3
```

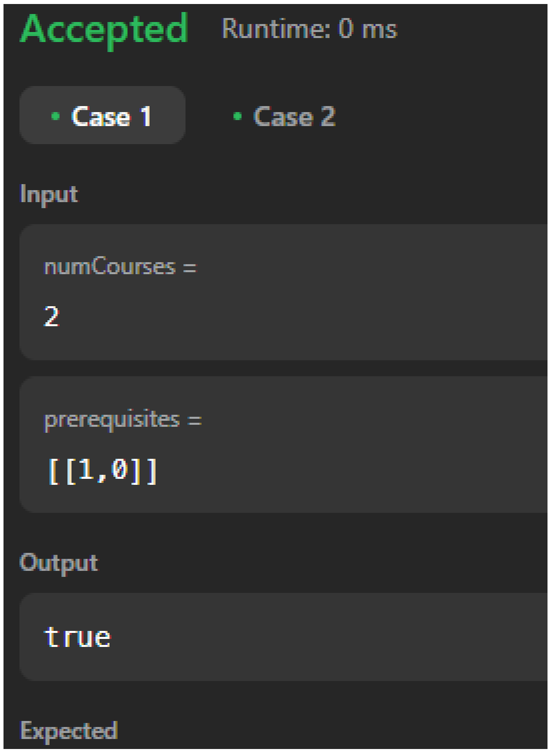
## QUESTION 2:

LeetCode Problem: Course Scheduling

## CODE:

```
#define MAX (13)


bool Cycle( int id, int coursemap[][MAX], int cpr[], int *visited ) {


    if ( visited[id] > 0 )
        return true;



    if ( visited[id] == 0 ) {
        visited[id] = 1;
        for ( int i=0; i<cpr[id]; i++ ) {
            if ( Cycle( coursemap[id][i], coursemap, cpr, visited ) ) {
                return true;
            }
        }
        visited[id] = -1;
    }

    return false;
}

bool canFinish(int numCourses, int** prerequisites, int prerequisitesSize, int*
prerequisitesColSize){
    int coursemap[2001][MAX];
    int cpr[2001]     = { 0 };
    int visited[2001] = { 0 };


    for ( int i=0; i<prerequisitesSize; i++ ) {
        int course = prerequisites[i][0];
        int prereq = prerequisites[i][1];

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

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

        if ( Cycle( i, coursemap, cpr, visited ) ) {
            return false;
        }
    }

    return true;
}
```

## OUTPUT:



Accepted Runtime: 0 ms

Input

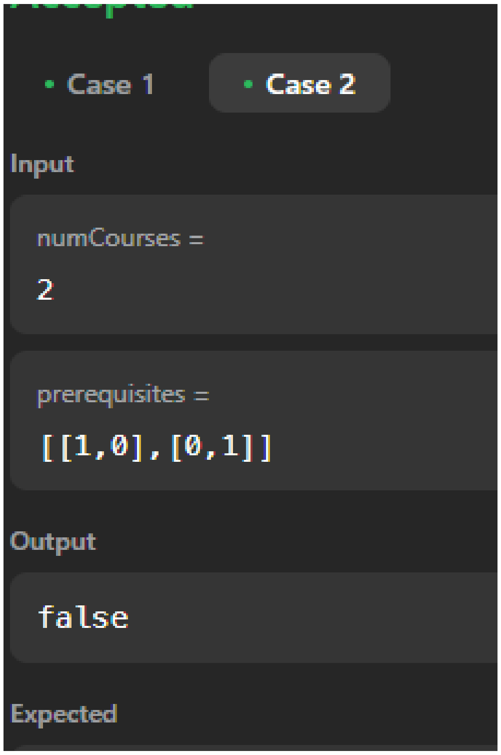
numCourses =
2

prerequisites =
[[1,0]]

Output

true

Expected



Input

numCourses =
2

prerequisites =
[[1,0],[0,1]]

Output

false

Expected

## LAB 3

## QUESTION 1:

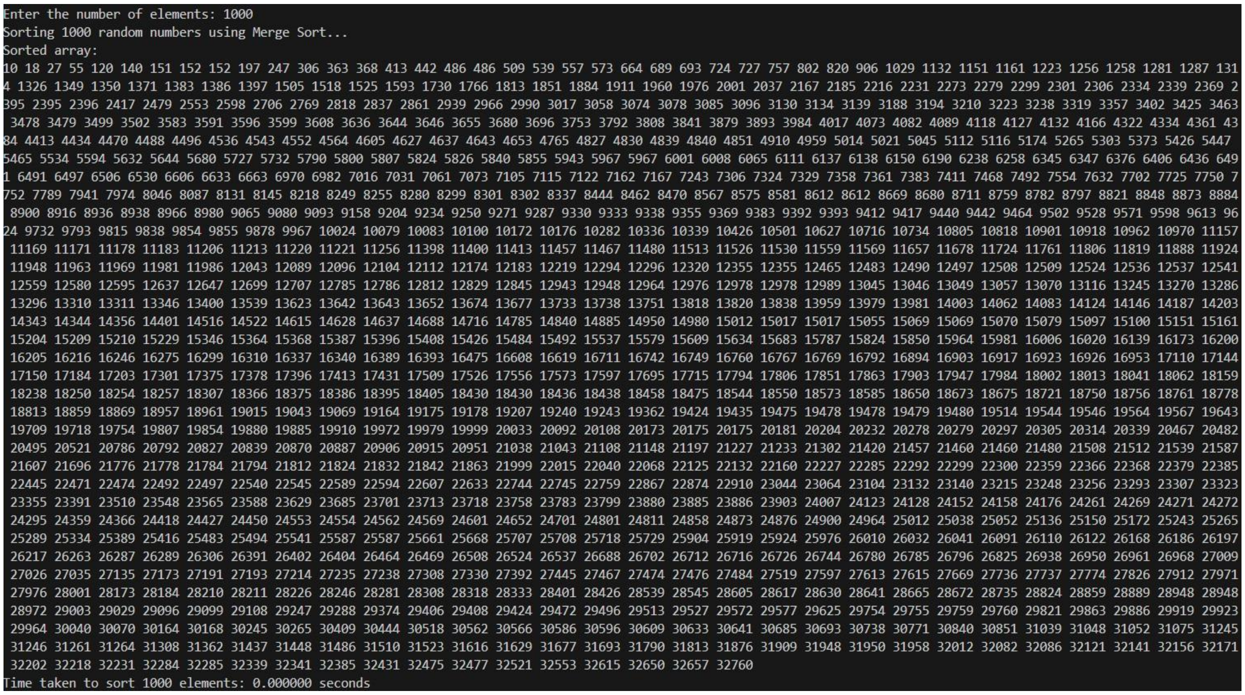
Sort a given set of N integer elements using Merge Sort technique and compute its time taken. Run the program for different values of N and record the time taken to sort.

## CODE:

```
#include <stdio.h>

#include <stdlib.h>
```

```
#include <time.h>

void merge(int arr[], int left, int mid, int right) {
    int n1 = mid - left + 1;
    int n2 = right - mid;
    int L[n1], R[n2];

    for (int i = 0; i < n1; i++)
        L[i] = arr[left + i];
    for (int j = 0; j < n2; j++)
        R[j] = arr[mid + 1 + j];

    int 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++;
    }
}

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

int main() {
    int n;

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

    int arr[n];
    srand(time(0));
    for (int i = 0; i < n; i++) {
        arr[i] = rand() % 100000;  // Random integers between 0 and 99999
    }

    printf("Running Merge Sort for N = %d...\n", n);

    clock_t start = clock();
    mergeSort(arr, 0, n - 1);
    clock_t end = clock();

    double time_taken = ((double)(end - start)) / CLOCKS_PER_SEC;
    printf("Time taken to sort N = %d elements: %f seconds\n", n, time_taken);

    return 0;
}
```

## OUTPUT:



```
Enter the number of elements: 1000
Sorting 1000 random numbers using Merge Sort...
Sorted array:
10 18 27 55 120 140 151 152 152 197 247 306 363 368 413 442 486 486 509 539 557 573 664 689 693 724 727 757 802 820 906 1029 1132 1151 1161 1223 1256 1258 1281 1287 131
4 1326 1349 1350 1371 1383 1386 1397 1505 1518 1525 1593 1730 1766 1813 1851 1884 1911 1960 1976 2001 2037 2167 2185 2216 2231 2273 2279 2299 2301 2306 2334 2339 2369 2
395 2395 2396 2417 2479 2553 2598 2706 2769 2818 2837 2861 2939 2966 2990 3017 3058 3074 3078 3085 3096 3130 3134 3139 3188 3194 3210 3223 3238 3319 3357 3402 3425 3463
 3478 3479 3499 3502 3583 3591 3596 3599 3608 3636 3644 3646 3655 3680 3696 3753 3792 3808 3841 3879 3893 3984 4017 4073 4082 4089 4118 4127 4132 4166 4322 4334 4361 43
84 4413 4434 4470 4488 4496 4536 4543 4552 4564 4605 4627 4637 4643 4653 4765 4827 4830 4839 4840 4851 4910 4959 5014 5021 5045 5112 5116 5174 5265 5303 5373 5426 5447
5465 5534 5594 5632 5644 5680 5727 5732 5790 5800 5807 5824 5826 5840 5855 5943 5967 5967 6001 6008 6065 6111 6137 6138 6150 6190 6238 6258 6345 6347 6376 6406 6436 649
1 6491 6497 6506 6530 6606 6633 6663 6970 6982 7016 7031 7061 7073 7105 7115 7122 7162 7167 7243 7306 7324 7329 7358 7361 7383 7411 7468 7492 7554 7632 7702 7725 7750 7
752 7789 7941 7974 8046 8087 8131 8145 8218 8249 8255 8280 8299 8301 8302 8337 8444 8462 8470 8567 8575 8581 8612 8612 8669 8680 8711 8759 8782 8797 8821 8848 8873 8884
 8900 8916 8936 8938 8966 8980 9065 9080 9093 9158 9204 9234 9250 9271 9287 9330 9333 9338 9355 9369 9383 9392 9393 9412 9417 9440 9442 9464 9502 9528 9571 9598 9613 96
24 9732 9793 9815 9838 9854 9855 9878 9967 10024 10079 10083 10100 10172 10176 10282 10336 10339 10426 10501 10627 10716 10734 10805 10818 10901 10918 10962 10970 11157
 11169 11171 11178 11183 11206 11213 11220 11221 11256 11398 11400 11413 11457 11467 11480 11513 11526 11530 11559 11569 11657 11678 11724 11761 11806 11819 11888 11924
 11948 11963 11969 11981 11986 12043 12089 12096 12104 12112 12174 12183 12219 12294 12296 12320 12355 12355 12465 12483 12490 12497 12508 12509 12524 12536 12537 12541
 12559 12580 12595 12637 12647 12699 12707 12785 12786 12812 12829 12845 12943 12948 12964 12976 12978 12978 12989 13045 13046 13049 13057 13070 13116 13245 13270 13286
 13296 13310 13311 13346 13400 13539 13623 13642 13643 13652 13674 13677 13733 13738 13751 13818 13820 13838 13959 13979 13981 14003 14062 14083 14124 14146 14187 14203
 14343 14344 14356 14401 14516 14522 14615 14628 14637 14688 14716 14785 14840 14885 14950 14980 15012 15017 15017 15055 15069 15069 15070 15079 15097 15100 15151 15161
 15204 15209 15210 15229 15346 15364 15368 15387 15396 15408 15426 15484 15492 15537 15579 15609 15634 15683 15787 15824 15850 15964 15981 16006 16020 16139 16173 16200
 16205 16216 16246 16275 16299 16310 16337 16340 16389 16393 16475 16608 16619 16711 16742 16749 16760 16767 16769 16792 16894 16903 16917 16923 16926 16953 17110 17144
 17150 17184 17203 17301 17375 17378 17396 17413 17431 17509 17526 17556 17573 17597 17695 17715 17794 17806 17851 17863 17903 17947 17984 18002 18013 18041 18062 18159
 18238 18250 18254 18257 18307 18366 18375 18386 18395 18405 18430 18430 18436 18438 18458 18475 18544 18550 18573 18585 18650 18673 18675 18721 18750 18756 18761 18778
 18813 18859 18869 18957 18961 19015 19043 19069 19164 19175 19178 19207 19240 19243 19362 19424 19435 19475 19478 19478 19479 19480 19514 19544 19546 19564 19567 19643
 19709 19718 19754 19807 19854 19880 19885 19910 19972 19979 19999 20033 20092 20108 20173 20175 20175 20181 20204 20232 20278 20279 20297 20305 20314 20339 20467 20482
 20495 20521 20786 20792 20827 20839 20870 20887 20906 20915 20951 21038 21043 21108 21148 21197 21227 21233 21302 21420 21457 21460 21460 21480 21508 21512 21539 21587
 21607 21696 21776 21778 21784 21794 21812 21824 21832 21842 21863 21999 22015 22040 22068 22125 22132 22160 22227 22285 22292 22299 22300 22359 22366 22368 22379 22385
 22445 22471 22474 22492 22497 22540 22545 22589 22594 22607 22633 22744 22745 22759 22867 22874 22910 23044 23064 23104 23132 23140 23215 23248 23256 23293 23307 23323
 23355 23391 23510 23548 23565 23588 23629 23685 23701 23713 23718 23758 23783 23799 23880 23885 23886 23903 24007 24123 24128 24152 24158 24176 24261 24269 24271 24272
 24295 24359 24366 24418 24427 24450 24553 24554 24562 24569 24601 24652 24701 24801 24811 24858 24873 24876 24900 24964 25012 25038 25052 25136 25150 25172 25243 25265
 25289 25334 25389 25416 25483 25494 25541 25587 25587 25661 25668 25707 25708 25718 25729 25904 25919 25924 25976 26010 26032 26041 26091 26110 26122 26168 26186 26197
 26217 26263 26287 26289 26306 26391 26402 26404 26464 26469 26508 26524 26537 26688 26702 26712 26716 26726 26744 26780 26785 26796 26825 26938 26950 26961 26968 27009
 27026 27035 27135 27173 27191 27193 27214 27235 27238 27308 27330 27392 27445 27467 27474 27476 27484 27519 27597 27613 27615 27669 27736 27737 27774 27826 27912 27971
 27976 28001 28173 28184 28210 28211 28226 28246 28281 28308 28318 28333 28401 28426 28539 28545 28605 28617 28630 28641 28665 28672 28735 28824 28859 28889 28948 28948
 28972 29003 29029 29096 29099 29108 29247 29288 29374 29406 29408 29424 29472 29496 29513 29527 29572 29577 29625 29754 29755 29759 29760 29821 29863 29886 29919 29923
 29964 30040 30070 30164 30168 30245 30265 30409 30444 30518 30562 30566 30586 30596 30609 30633 30641 30685 30693 30738 30771 30840 30851 31039 31048 31052 31075 31245
 31246 31261 31264 31308 31362 31437 31448 31486 31510 31523 31616 31629 31677 31693 31790 31813 31876 31909 31948 31950 31958 32012 32082 32086 32121 32141 32156 32171
 32202 32218 32231 32284 32285 32339 32341 32385 32431 32475 32477 32521 32553 32615 32650 32657 32760
Time taken to sort 1000 elements: 0.000000 seconds
```

## QUESTION 2:

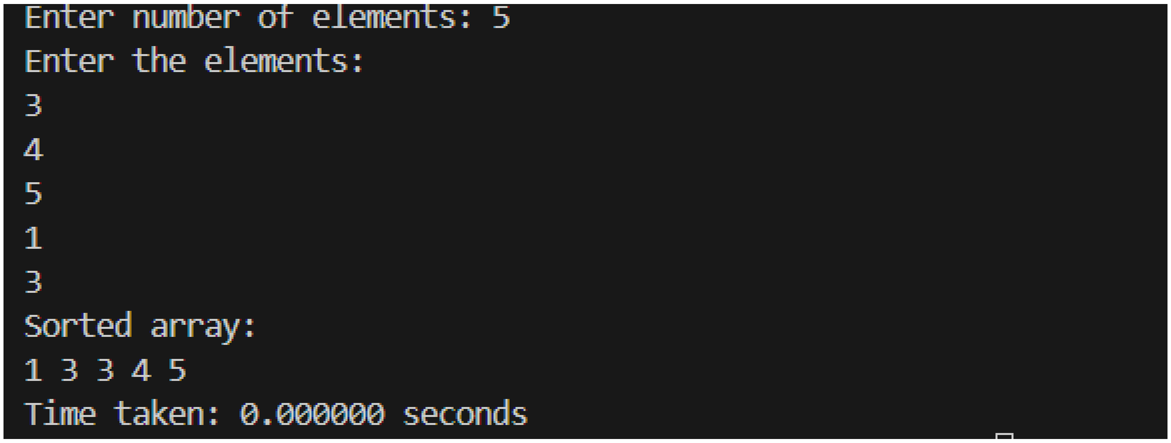
Sort a given set of N integer elements using Quick Sort technique and compute its time taken.

## CODE:

```
#include  <stdio.h>

#include <stdlib.h>

#include <time.h>


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

    int t = *a;

    *a = *b;

    *b = t;

}
```

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

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

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

    clock_t start = clock();
    quickSort(arr, 0, n - 1);
    clock_t end = clock();

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

    printf("Sorted array:\n");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\nTime taken: %f seconds\n", time_taken);

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

## OUTPUT:



```
Enter number of elements: 5
Enter the elements:
3
4
5
1
3
Sorted array:
1 3 3 4 5
Time taken: 0.000000 seconds
```

## QUESTION 3:

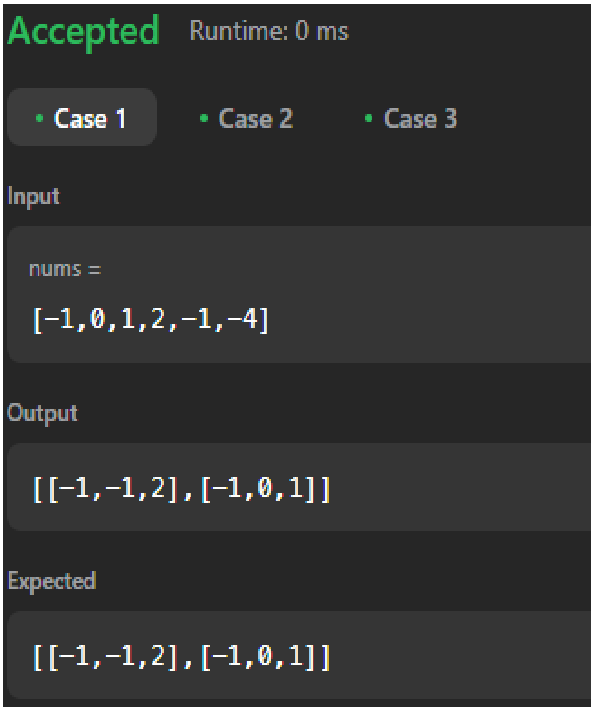
LeetCode Problem : 3Sum

Given an integer array nums, return all the triplets [nums[i], nums[j], nums[k]] such that i != j, i != k, and j != k, and nums[i] + nums[j] + nums[k] == 0.

## CODE:

```
int compare(const void *a, const void *b) {
    return (*(int*)a - *(int*)b);
}


int** threeSum(int* nums, int numsSize, int* returnSize, int** returnColumnSizes)
{
    if (numsSize < 3) {
        *returnSize = 0;
        return NULL;
    }
    qsort(nums, numsSize, sizeof(int), compare);
    int** result = (int**)malloc(sizeof(int*) * numsSize * numsSize);
    *returnColumnSizes = (int*)malloc(sizeof(int) * numsSize * numsSize);
    *returnSize = 0;

    for (int i = 0; i < numsSize - 2; i++) {
        if (i > 0 && nums[i] == nums[i - 1]) continue;
        int left = i + 1, right = numsSize - 1;
        while (left < right) {
            int sum = nums[i] + nums[left] + nums[right];
            if (sum == 0) {
                result[*returnSize] = (int*)malloc(sizeof(int) * 3);
                result[*returnSize][0] = nums[i];
                result[*returnSize][1] = nums[left];
                result[*returnSize][2] = nums[right];
                (*returnColumnSizes)[*returnSize] = 3;
                (*returnSize)++;
                while (left < right && nums[left] == nums[left + 1]) left++;
                while (left < right && nums[right] == nums[right - 1]) right--;
                left++; right--;
            } else if (sum < 0) left++;
            else right--;
        }
    }
    return result;
}
```

## OUTPUT:



Accepted Runtime: 0 ms

Input

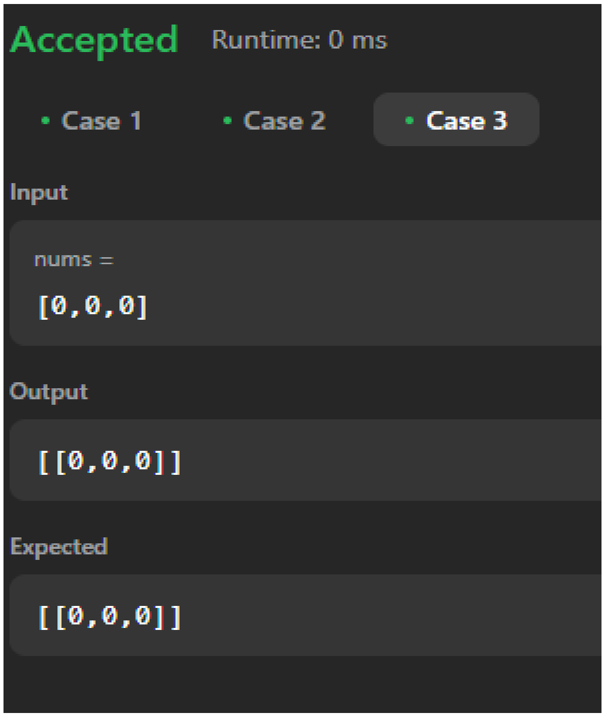
```
nums =
[-1,0,1,2,-1,-4]
```

Output

```
[[-1,-1,2],[-1,0,1]]
```

Expected

```
[[-1,-1,2],[-1,0,1]]
```



Accepted Runtime: 0 ms

Input

```
nums =
[0,0,0]
```

Output

```
[[0,0,0]]
```

Expected

```
[[0,0,0]]
```

## LAB4:

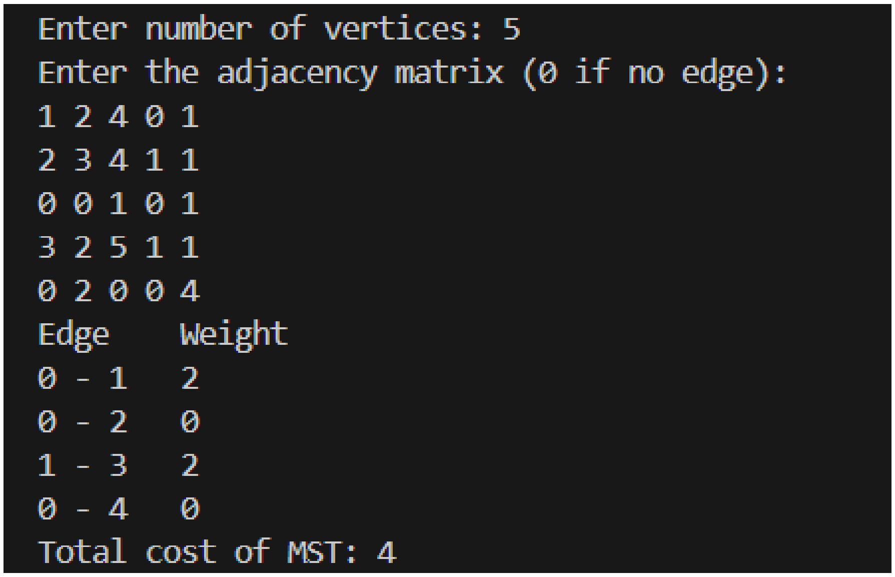
## QUESTION 1:

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

## CODE:

```
#include <stdio.h>

#include <limits.h>


#define MAX 100


int minKey(int key[], int mstSet[], int n) {
    int min = INT_MAX, min_index;
    for (int v = 0; v < n; v++) {
        if (mstSet[v] == 0 && key[v] < min) {
            min = key[v];
            min_index = v;
        }
    }
    return min_index;
}

void primMST(int graph[MAX][MAX], int n) {
    int parent[MAX];
    int key[MAX];
    int mstSet[MAX];

    for (int i = 0; i < n; i++) {
        key[i] = INT_MAX;
        mstSet[i] = 0;
    }

    key[0] = 0;
    parent[0] = -1;

    for (int count = 0; count < n - 1; count++) {
        int u = minKey(key, mstSet, n);
        mstSet[u] = 1;
        for (int v = 0; v < n; v++) {
            if (graph[u][v] && mstSet[v] == 0 && graph[u][v] < key[v]) {
                parent[v] = u;
                key[v] = graph[u][v];
            }
        }
    }

    int totalCost = 0;
    printf("Edge \tWeight\n");
    for (int i = 1; i < n; i++) {
        printf("%d - %d \t%d\n", parent[i], i, graph[i][parent[i]]);
        totalCost += graph[i][parent[i]];
    }
    printf("Total cost of MST: %d\n", totalCost);
}

int main() {
    int n;
    int graph[MAX][MAX];

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

    printf("Enter the adjacency matrix (0 if no edge):\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            scanf("%d", &graph[i][j]);
        }
    }

    primMST(graph, n);
    return 0;
}
```

## OUTPUT:



```
Enter number of vertices: 5
Enter the adjacency matrix (0 if no edge):
1 2 4 0 1
2 3 4 1 1
0 0 1 0 1
3 2 5 1 1
0 2 0 0 4
Edge    Weight
0 - 1   2
0 - 2   0
1 - 3   2
0 - 4   0
Total cost of MST: 4
```

## QUESTION 2:

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

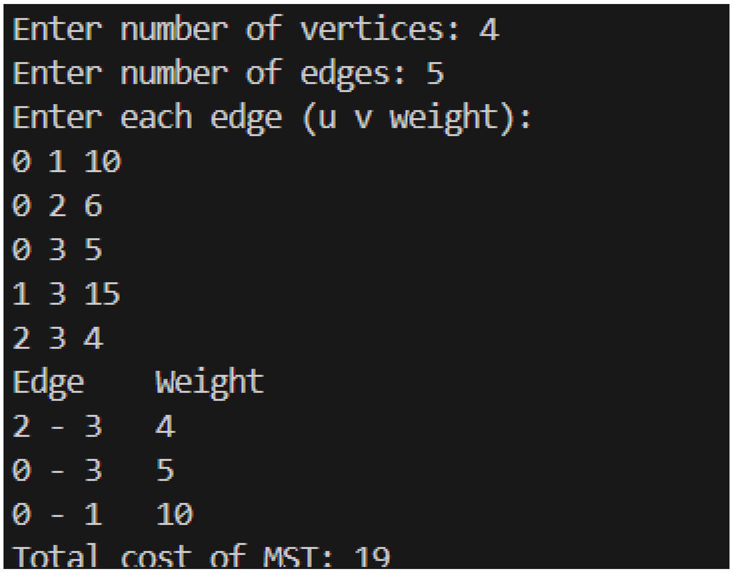
## CODE:

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

```
#define MAX 100

typedef struct {
    int u, v, weight;
} Edge;

int parent[MAX];

int find(int i) {
    while (parent[i] != i)
        i = parent[i];
    return i;
}

void unionSet(int i, int j) {
    int a = find(i);
    int b = find(j);
    parent[a] = b;
}

void kruskal(Edge edges[], int n, int e) {
    Edge result[MAX];
    int i = 0, j = 0, totalCost = 0;

    for (int k = 0; k < n; k++)
        parent[k] = k;
    while (j < n - 1 && i < e) {
        int u = edges[i].u;
        int v = edges[i].v;
        int set_u = find(u);
        int set_v = find(v);

        if (set_u != set_v) {
            result[j++] = edges[i];
            totalCost += edges[i].weight;
            unionSet(set_u, set_v);
        }
        i++;
    }

    printf("Edge \tWeight\n");
    for (int k = 0; k < j; k++)
        printf("%d - %d \t%d\n", result[k].u, result[k].v, result[k].weight);
    printf("Total cost of MST: %d\n", totalCost);
}

int compare(const void *a, const void *b) {
    Edge *e1 = (Edge *)a;
    Edge *e2 = (Edge *)b;
    return e1->weight - e2->weight;
}
```

```
int main() {
    int n, e;
    Edge edges[MAX];

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

    printf("Enter number of edges: ");
    scanf("%d", &e);

    printf("Enter each edge (u v weight):\n");
    for (int i = 0; i < e; i++)
        scanf("%d %d %d", &edges[i].u, &edges[i].v, &edges[i].weight);

    qsort(edges, e, sizeof(Edge), compare);
    kruskal(edges, n, e);

    return 0;
}
```

**OUTPUT:**



```
Enter number of vertices: 4
Enter number of edges: 5
Enter each edge (u v weight):
0 1 10
0 2 6
0 3 5
1 3 15
2 3 4
Edge    Weight
2 - 3   4
0 - 3   5
0 - 1   10
Total cost of MST: 19
```

## LAB 5:

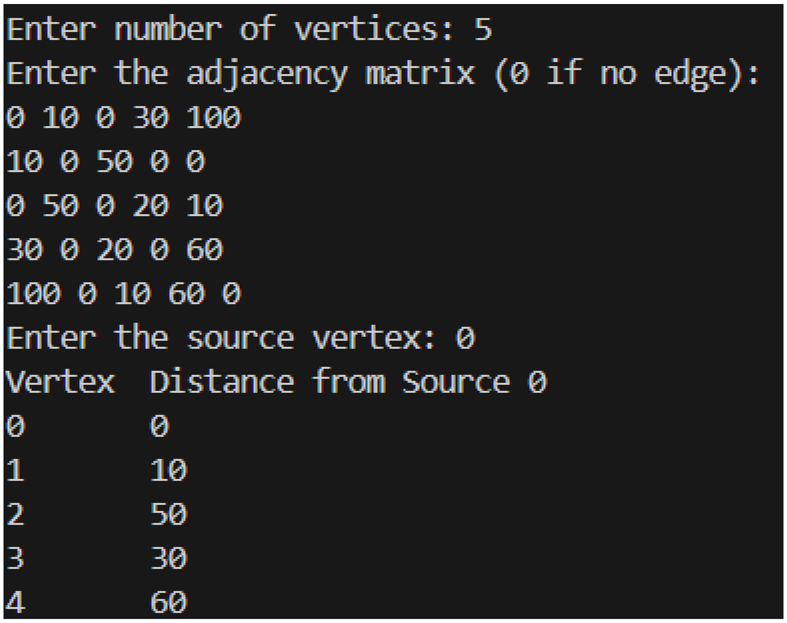
## QUESTION 1:

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

## CODE:

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


#define MAX 100
#define INF 99999


int minDistance(int dist[], int visited[], int n) {
    int min = INF, min_index;
    for (int v = 0; v < n; v++) {
        if (!visited[v] && dist[v] <= min) {
            min = dist[v];
            min_index = v;
        }
    }
    return min_index;
}

void dijkstra(int graph[MAX][MAX], int n, int src) {
    int dist[MAX], visited[MAX];

    for (int i = 0; i < n; i++) {
        dist[i] = INF;
        visited[i] = 0;
    }

    dist[src] = 0;

    for (int count = 0; count < n - 1; count++) {
        int u = minDistance(dist, visited, n);
        visited[u] = 1;

        for (int v = 0; v < n; v++) {
            if (!visited[v] && graph[u][v] && dist[u] != INF &&
                dist[u] + graph[u][v] < dist[v]) {
                dist[v] = dist[u] + graph[u][v];
            }
        }
    }
    printf("Vertex\tDistance from Source %d\n", src);
    for (int i = 0; i < n; i++)
        printf("%d\t%d\n", i, dist[i]);
}

int main() {
    int n, src;
    int graph[MAX][MAX];

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

    printf("Enter the adjacency matrix (0 if no edge):\n");
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            scanf("%d", &graph[i][j]);

    printf("Enter the source vertex: ");
    scanf("%d", &src);

    dijkstra(graph, n, src);

    return 0;
}
```

**OUTPUT:**



```
Enter number of vertices: 5
Enter the adjacency matrix (0 if no edge):
0 10 0 30 100
10 0 50 0 0
0 50 0 20 10
30 0 20 0 60
100 0 10 60 0
Enter the source vertex: 0
Vertex  Distance from Source 0
0       0
1       10
2       50
3       30
4       60
```

## QUESTION 2:

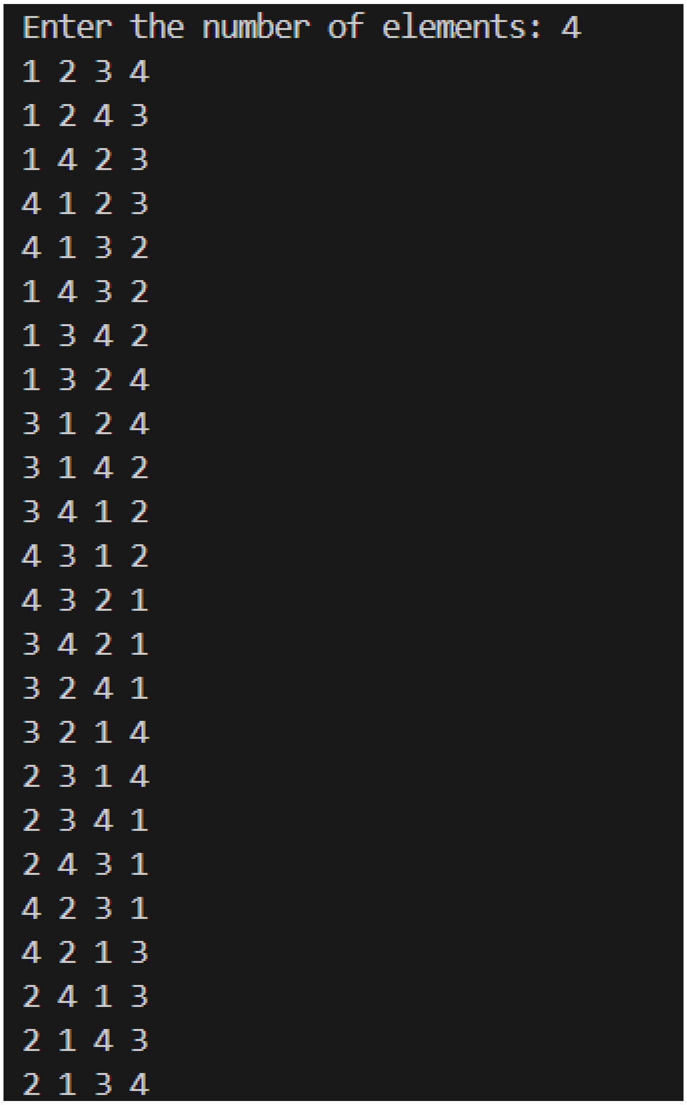
Implement Johnson Trotter algorithm to generate permutations.

## CODE:

```
#include <stdio.h>


#define LEFT -1
#define RIGHT 1


typedef struct {
    int value;
    int dir;
} Element;
```

```
int getMobile(Element perm[], int n) {
    int mobile = 0, mobileIndex = -1;
    for (int i = 0; i < n; i++) {
        int next = i + perm[i].dir;
        if (next >= 0 && next < n && perm[i].value > perm[next].value) {
            if (perm[i].value > mobile) {
                mobile = perm[i].value;
                mobileIndex = i;
            }
        }
    }
    return mobileIndex;
}

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

void generatePermutations(int n) {
    Element perm[n];
    for (int i = 0; i < n; i++) {
        perm[i].value = i + 1;
        perm[i].dir = LEFT;
    }
    printPermutation(perm, n);

    while (1) {
        int mobileIndex = getMobile(perm, n);
        if (mobileIndex == -1)
            break;

        int swapIndex = mobileIndex + perm[mobileIndex].dir;
        Element temp = perm[mobileIndex];
        perm[mobileIndex] = perm[swapIndex];
        perm[swapIndex] = temp;

        mobileIndex = swapIndex;

        for (int i = 0; i < n; i++) {
            if (perm[i].value > perm[mobileIndex].value)
                perm[i].dir *= -1;
        }

        printPermutation(perm, n);
    }
}

int main() {
    int n;
    printf("Enter the number of elements: ");
    scanf("%d", &n);
    generatePermutations(n);
    return 0;
}
```

## OUTPUT:



```
Enter the number of elements: 4
1 2 3 4
1 2 4 3
1 4 2 3
4 1 2 3
4 1 3 2
1 4 3 2
1 3 4 2
1 3 2 4
3 1 2 4
3 1 4 2
3 4 1 2
4 3 1 2
4 3 2 1
3 4 2 1
3 2 4 1
3 2 1 4
2 3 1 4
2 3 4 1
2 4 3 1
4 2 3 1
4 2 1 3
2 4 1 3
2 1 4 3
2 1 3 4
```

## LAB 6:

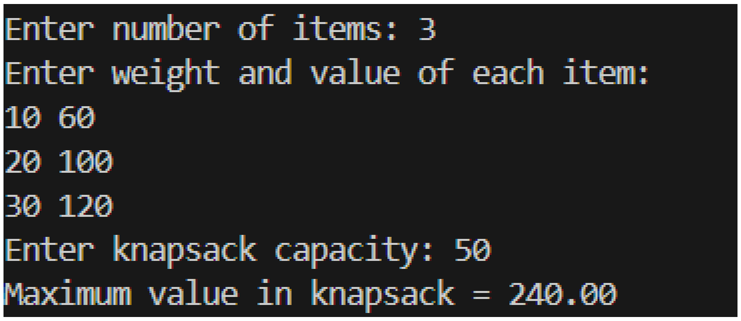
## QUESTION 1:

Implement fractional knapsack problem using Greedy technique.

## CODE:

```
#include <stdio.h>

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

void sortItems(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;
            }
        }
    }
}

void fractionalKnapsack(Item items[], int n, int capacity) {
    sortItems(items, n);

    float totalValue = 0.0;
    int currWeight = 0;

    for (int i = 0; i < n; i++) {
        if (currWeight + items[i].weight <= capacity) {
            currWeight += items[i].weight;
            totalValue += items[i].value;
        } else {
            int remain = capacity - currWeight;
            totalValue += items[i].ratio * remain;
            break;
        }
    }

    printf("Maximum value in knapsack = %.2f\n", totalValue);
}

int main() {
    int n, capacity;

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

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

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

    fractionalKnapsack(items, n, capacity);

    return 0;
}
```

## OUTPUT:



```
Enter number of items: 3
Enter weight and value of each item:
10 60
20 100
30 120
Enter knapsack capacity: 50
Maximum value in knapsack = 240.00
```

## QUESTION 2:

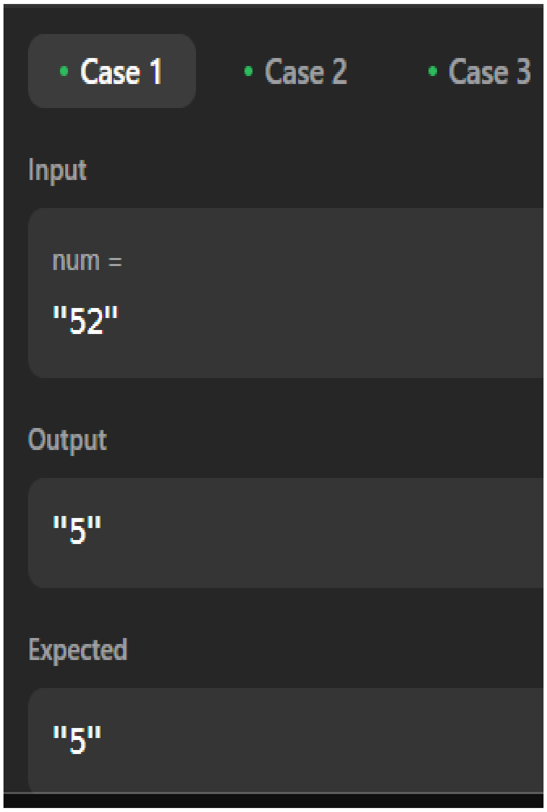
LeetCode Problem- Largest Odd Number in a String

You are given a string num, representing a large integer. Return *the* ***largest-valued odd*** *integer (as a string) that is a* ***non-empty substring*** *of* num*, or an empty string* "" *if no odd integer exists*.

## Code:

```
class Solution {
    public:
        string largestOddNumber(string num) {
            return num.substr(0, num.find_last_not_of("0|2|4|6|8") + 1);
        }
    };
```

## OUTPUT:



Input

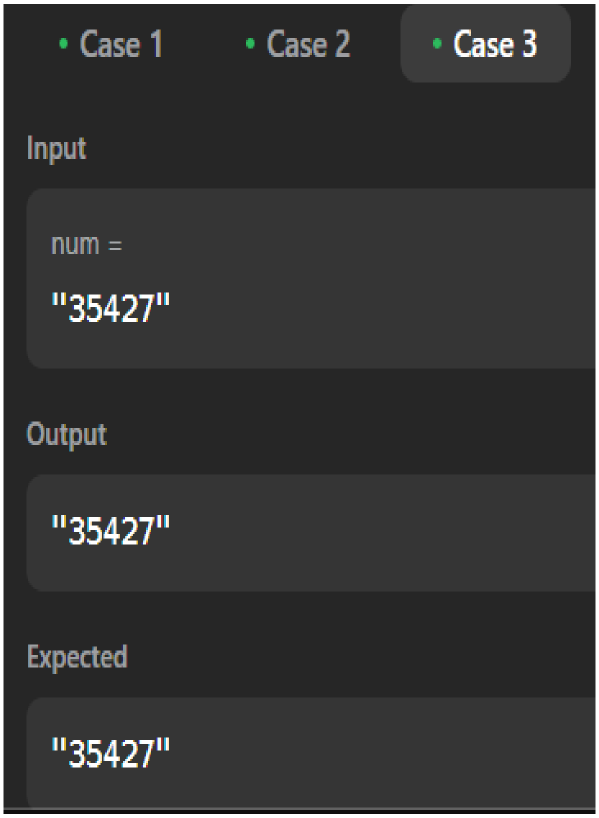
num =
"52"

Output

"5"

Expected

"5"



Input

num =
"35427"

Output

"35427"

Expected

"35427"

## LAB 7:

## QUESTION 1:

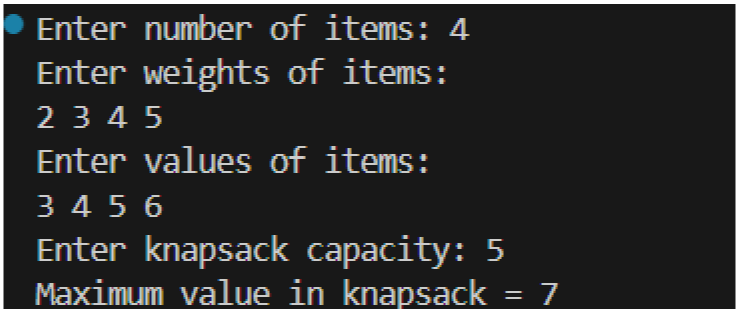
Implement 0/1 Knapsack problem using dynamic programming.

## CODE:

```
#include <stdio.h>

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

int knapsack(int weight[], int value[], 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 (weight[i - 1] <= w)
                dp[i][w] = max(value[i - 1] + dp[i - 1][w - weight[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 weight[n], value[n];

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

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

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

    int maxValue = knapsack(weight, value, n, capacity);
    printf("Maximum value in knapsack = %d\n", maxValue);

    return 0;
}
```

**OUPUT:**



```
Enter number of items: 4
Enter weights of items:
2 3 4 5
Enter values of items:
3 4 5 6
Enter knapsack capacity: 5
Maximum value in knapsack = 7
```

## QUESTION 2:

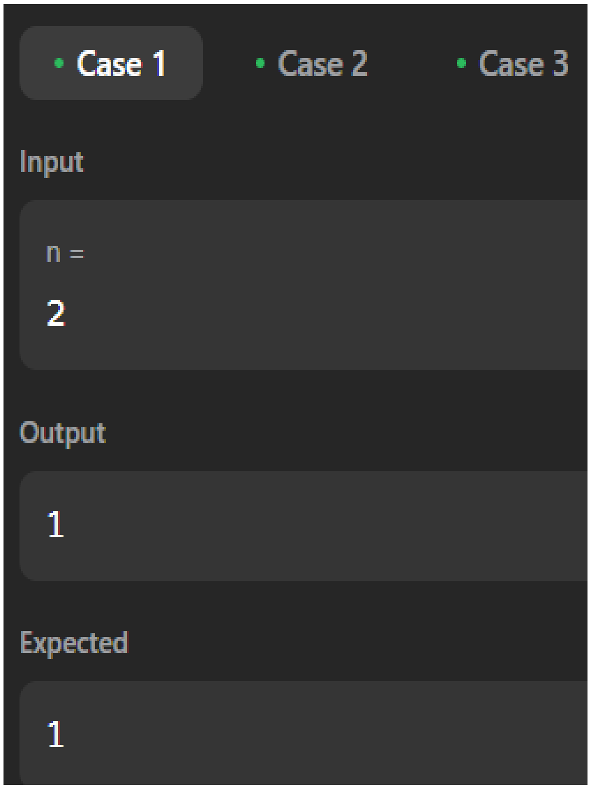
LeetCode Problemm : Fibonacci Number

## CODE:

```
int fib(int n){
 if (n <= 1)
    return n;

  // Array to store Fibonacci numbers
  int f[n + 1]; // 1 extra to handle case n = 0

  // First two Fibonacci numbers
  f[0] = 1;
  f[1] = 1;

  // Build the Fibonacci sequence
  for (int i = 2; i <= n; i++) {
    f[i] = f[i - 1] + f[i - 2];
  }
```

```
    return f[n - 1];
}
```

## OUTPUT:



Input

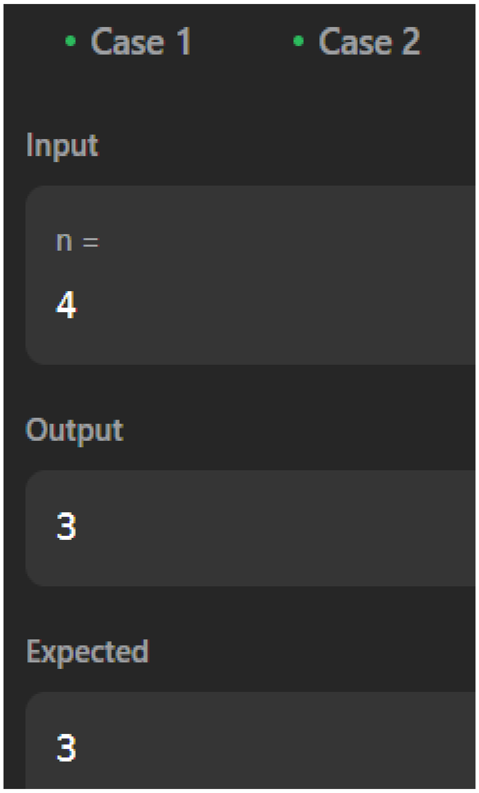
n =
2

Output

1

Expected

1



Input

n =
4

Output

3

Expected

3

## LAB 8:

## QUESTION 1:

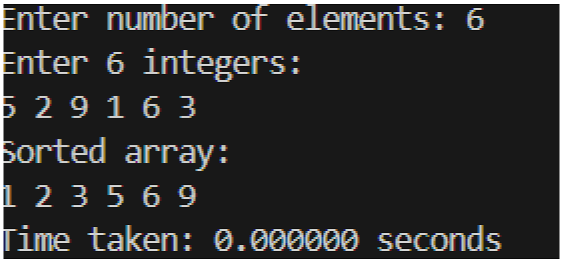
Sort a given set of N integer elements using Heap Sort technique and compute its time taken.

## CODE:

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

```
void heapify(int arr[], int n, int i) {
    int largest = i;
    int l = 2 * i + 1;
    int r = 2 * i + 2;

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

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

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

    int arr[n];
    printf("Enter %d integers:\n", n);
    for (int i = 0; i < n; i++)
        scanf("%d", &arr[i]);

    clock_t start, end;
    start = clock();
    heapSort(arr, n);
    end = clock();

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

    double time_taken = ((double)(end - start)) / CLOCKS_PER_SEC;
    printf("Time taken: %.6f seconds\n", time_taken);
    return 0;
}
```

## OUTPUT:



```
Enter number of elements: 6
Enter 6 integers:
5 2 9 1 6 3
Sorted array:
1 2 3 5 6 9
Time taken: 0.000000 seconds
```

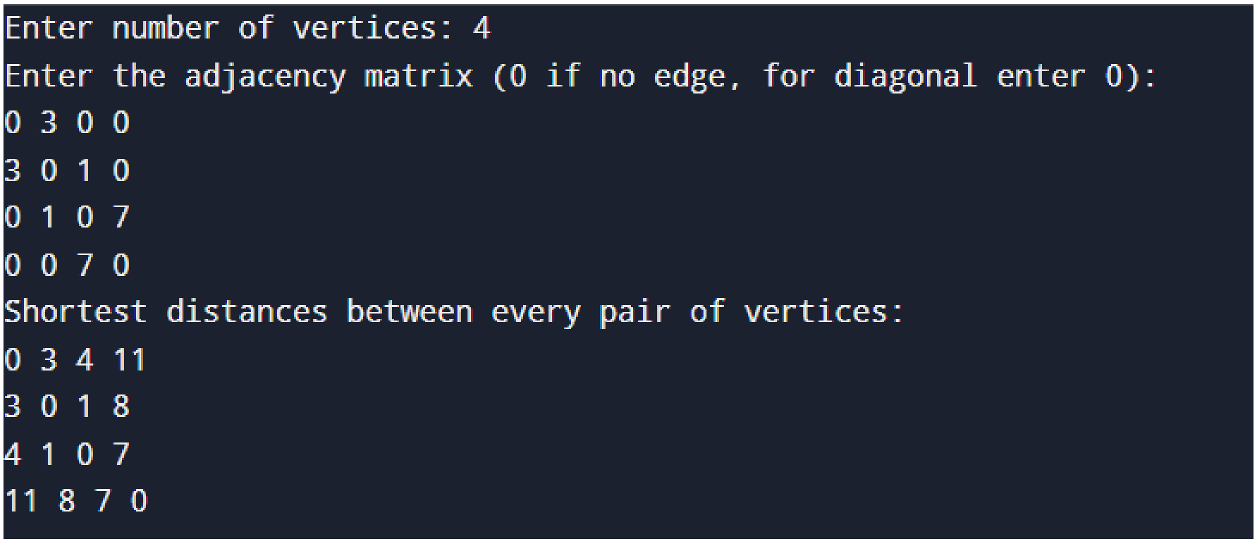
## QUESTION 2:

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

## CODE:

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

#define INF 99999
#define MAX 100

void floydWarshall(int graph[MAX][MAX], int n) {
    int dist[MAX][MAX];

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (graph[i][j] == 0 && i != j) {
                dist[i][j] = INF; // No edge between i and j
            } else {
                dist[i][j] = graph[i][j]; // Set initial distances
            }
        }
    }


    for (int k = 0; k < n; k++) {
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; 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]; // Update the distance if shorter path is found
                }
            }
        }
    }

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

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

    int graph[MAX][MAX];

    printf("Enter the adjacency matrix (0 if no edge, for diagonal enter 0):\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            scanf("%d", &graph[i][j]);
        }
    }

    floydWarshall(graph, n);

    return 0;
}
```

**OUTPUT:**



```
Enter number of vertices: 4
Enter the adjacency matrix (0 if no edge, for diagonal enter 0):
0 3 0 0
3 0 1 0
0 1 0 7
0 0 7 0
Shortest distances between every pair of vertices:
0 3 4 11
3 0 1 8
4 1 0 7
11 8 7 0
```

## QUESTION 3:

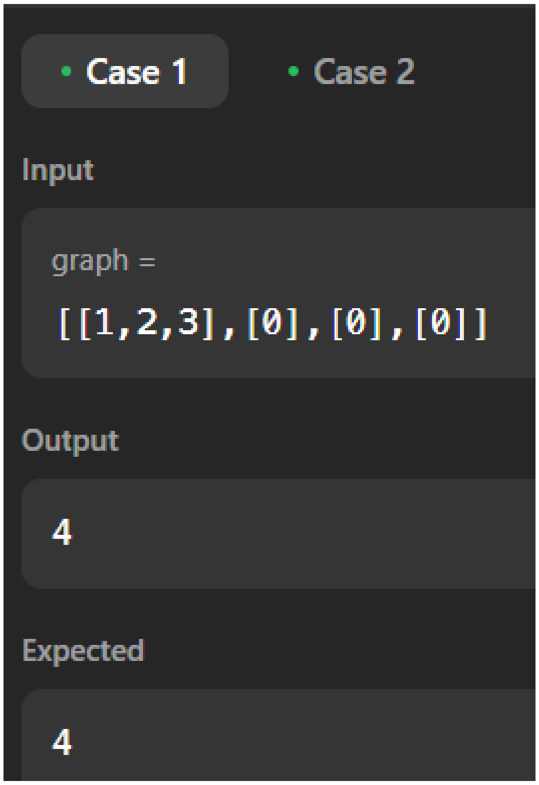
LeetCode Problem :Shortest Path Visiting All Nodes

## CODE:

```
struct Node {
    int id;
    int mask;
    int level;
};

struct Queue {
    struct Node data[10000];
    int i;
    int j;
    int size;
};
```

```
void qpush(struct Queue* q, struct Node n) {
    q->size ++;
    q->data[q->i++] = n;
    q->i = q->i % 10000;
}

struct Node qpop(struct Queue* q) {
    struct Node r = q->data[q->j++];
    q->size --;
    q->j = q->j % 10000;
    return r;
}

int qempty(struct Queue* q) {
    return q->size == 0;
}

int shortestPathLength(int** graph, int graphSize, int* graphColSize) {
    struct Queue q;
    char V[(1<<(13+4)) + 100] = {0}; // visited
    memset(&q, 0, sizeof(q));
    for (int i = 0; i < graphSize; i++) {
        struct Node n = {i, 1 << i, 1};
        qpush(&q, n);
    }
    while (!qempty(&q)) {
        struct Node n = qpop(&q);
        if (n.mask == (1<<graphSize)-1) {
            return n.level-1;
        }

        for (int i = 0; i < graphColSize[n.id]; i++) {
            int mask = n.mask;
            int jd = graph[n.id][i];
            mask |= 1<<jd;

            struct Node m = {jd, mask, n.level+1};

            if (!V[(jd << 13) | mask]) {
                V[(jd << 13) | mask] = 1;
                qpush(&q, m);
            }
        }
    }
    return -1;
}
```

**OUTPUT:**



Input

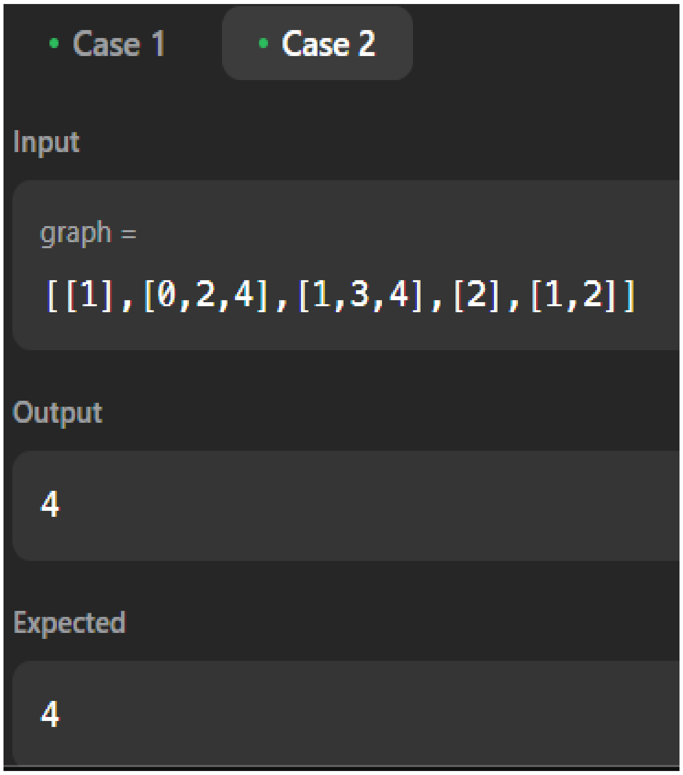
graph =
[[1,2,3],[0],[0],[0]]

Output

4

Expected

4



Input

graph =
[[1],[0,2,4],[1,3,4],[2],[1,2]]

Output

4

Expected

4

## LAB 9:

## QUESTION 1:

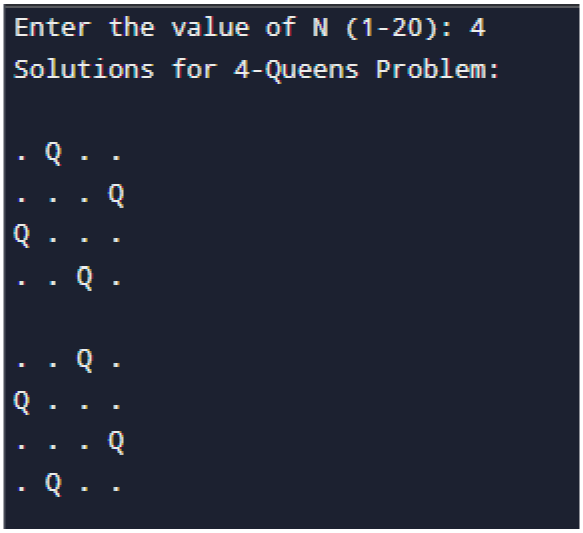
Implement “N-Queens Problem” using Backtracking.

## CODE:

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

#define MAX 20

int board[MAX];
int N;
```

```
bool isSafe(int row, int col) {
    for (int i = 0; i < row; i++) {
        if (board[i] == col ||
            board[i] - i == col - row ||
            board[i] + i == col + row)
            return false;
    }
    return true;
}

void solveNQueens(int row) {
    if (row == N) {
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < N; j++) {
                if (board[i] == j)
                    printf("Q ");
                else
                    printf(". ");
            }
            printf("\n");
        }
        printf("\n");
        return;
    }

    for (int col = 0; col < N; col++) {
        if (isSafe(row, col)) {
            board[row] = col;
            solveNQueens(row + 1);
        }
    }
}

int main() {
    printf("Enter the value of N (1-%d): ", MAX);
    scanf("%d", &N);

    if (N < 1 || N > MAX) {
        printf("Invalid input. N should be between 1 and %d.\n", MAX);
        return 1;
    }

    printf("Solutions for %d-Queens Problem:\n\n", N);
    solveNQueens(0);

    return 0;
}
```

**OUTPUT:**



```
Enter the value of N (1-20): 4
Solutions for 4-Queens Problem:

. Q . .
. . . Q
Q . . .
. . Q .

. . Q .
Q . . .
. . . Q
. Q . .
```