

# VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.



## LAB REPORT on

## Analysis and Design of Algorithms

*Submitted by*

**ROHIT RAMCHANDRA GANDHI (1BM23CS417)**

*in partial fulfillment for the award of the degree of*

**BACHELOR OF ENGINEERING**

*in*

**COMPUTER SCIENCE AND ENGINEERING**



**B.M.S. COLLEGE OF ENGINEERING**

(Autonomous Institution under VTU)

**BENGALURU-560019**

**April-2024 to August-2024**

**B. M. S. College of Engineering,**

**Bull Temple Road, Bangalore 560019**

(Affiliated to Visvesvaraya Technological University, Belgaum)

**Department of Computer Science and Engineering**



**CERTIFICATE**

This is to certify that the Lab work entitled “**Analysis and Design of Algorithms**” carried out by **ROHIT RAMCHNADRA GANDHI(1BM22CS098)**, 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 academic semester April-2024 to August-2024. The Lab report has been approved as it satisfies the academic requirements in respect of an **Analysis and Design of Algorithms (23CS4PCADA)** work prescribed for the said degree.

**Madhavi R.P**

Associate Professor  
Department of CSE  
BMSCE, Bengaluru

**Dr. Jyothi S Nayak**

Professor and Head  
Department of CSE  
BMSCE, Bengaluru

## Index Sheet

Lab Program No.	Program Details	Page No.
1	➤ Leetcode exercise on Numbers Disappeared in an array.	1
2	➤ Leetcode exercises on binary search tree	2-3
3	➤ Leetcode exercises on Increasing order search tree	4-5
4	<ul style="list-style-type: none"> <li>➤ Write program to obtain the Topological ordering of vertices in a given digraph using DFS.</li> <li>➤ Write program to obtain the Topological ordering of vertices in a given digraph using source removal method.</li> </ul>	5-12
5	<ul style="list-style-type: none"> <li>➤ Sort a given set of N integer elements using selection sort technique and compute its time taken. Run the program for different values of N and record the time taken to sort.</li> <li>➤ 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.</li> </ul>	13-22
6	➤ Sort a given set of N integer elements using Quick Sort technique and compute its time taken. Run the program for different values of N and record the time taken to sort.	22-25
7	<ul style="list-style-type: none"> <li>➤ Implement Johnson Trotter algorithm to generate permutations.</li> <li>➤ Substring Matching Program.</li> <li>➤ Leetcode : exercise on Find Kth Largest Integer</li> </ul>	26-31
8	<ul style="list-style-type: none"> <li>➤ Sort a given set of N integer elements using Heap Sort technique and compute its time taken.</li> <li>➤ Floyds algorithm.</li> </ul>	32-36

9	<ul style="list-style-type: none"> <li>➤ Implement 0/1 Knapsack problem using dynamic programming.</li> <li>➤ Find Minimum Cost Spanning Tree of a given undirected graph using Prim's algorithm.</li> </ul>	36-40
10	<ul style="list-style-type: none"> <li>➤ From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm.</li> <li>➤ Find Minimum Cost Spanning Tree of a given undirected graph using Kruskal's algorithm.</li> </ul>	40-46

## Course Outcome

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

## 1. Leetcode :exercise on Numbers Disappeared in an array.

### 448. Find All Numbers Disappeared in an Array:

Given an array `nums` of `n` integers where `nums[i]` is in the range `[1, n]`, return an array of all the integers in the range `[1, n]` that do not appear in `nums`.

```
int* findDisappearedNumbers(int* nums, int numsSize, int* returnSize) {  
    int temp = 0;  
    for (int index = 0; index < numsSize; ++index) {  
        temp = abs(nums[index]) - 1;  
        nums[temp] = abs(nums[temp]) * -1; }  
    int insert_index = 0;  
    *returnSize = 0;  
    for (int index = 0; index < numsSize; ++index) {  
        if (nums[index] > 0) {  
            ++*returnSize;  
            nums[insert_index++] = index + 1; }  
    }
```

Accepted

Rohitgandhi02599 submitted at May 02, 2024 09:58

Editorial

Solution

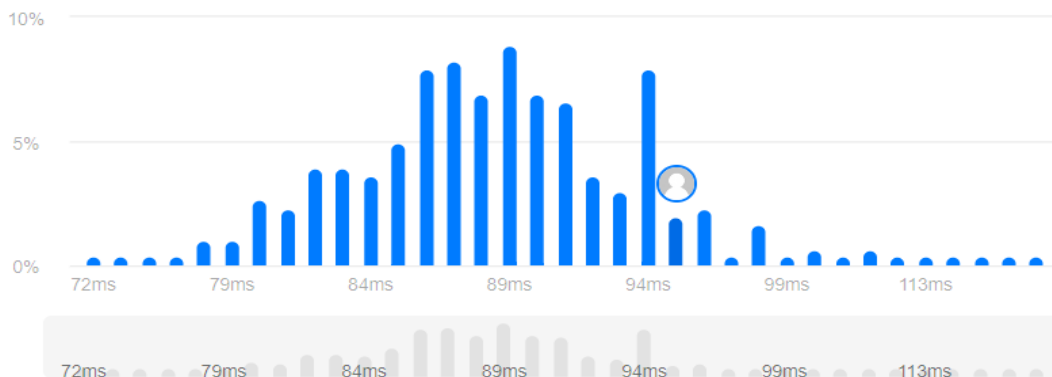
Runtime

95 ms | Beats 15.46%

Analyze Complexity

Memory

16.90 MB | Beats 89.47%



## 2. Leetcode :exercises on Binary Tree Zigzag Level Order Traversal


### 103. Binary Tree Zigzag Level Order Traversal


**Given the root of a binary tree, return the zigzag level order traversal of its nodes' values. (i.e., from left to right, then right to left for the next level and alternate between).**


```
int** zigzagLevelOrder(struct TreeNode* root, int* returnSize, int** returnColumnSizes) {
    int **ans = malloc(2000*sizeof(int*));
    *returnColumnSizes = malloc(2000*sizeof(int));
    *returnSize = 0;
    struct TreeNode *tmp[2000] = {0};
    int top = -1, start = 0;
    tmp[++top] = root;
    while(tmp[start])
    {
        int tmp_top = top;
        ans[(*returnSize)] = malloc((top-start+1)*sizeof(int));
        (*returnColumnSizes)[(*returnSize)] = (top-start+1);
        int idx = (*returnSize)%2 ? (top-start+1)-1:0;
        int step = (*returnSize)%2 ? -1:1;
        while(start <= tmp_top)
        {
            ans[(*returnSize)][idx] = tmp[start]->val;
            if(tmp[start]->left)
                tmp[++top] = tmp[start]->left;
            if(tmp[start]->right)
                tmp[++top] = tmp[start]->right;
            start++;
            idx += step;
        }
        *returnSize++;
    }
}
```


```
    idx += step;
}
(*returnSize)++;
}
return ans;
```

Accepted


 Rohitgandhi02599 submitted at May 09, 2024 10:03

 Editorial

 Solution

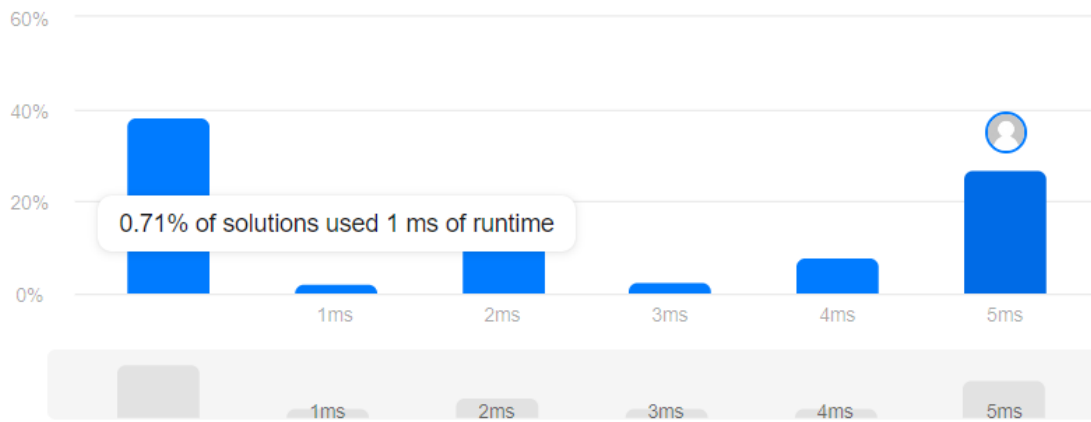
 Runtime

5 ms | Beats 36.17%

 Analyze Complexity

 Memory

7.01 MB | Beats 51.06% 



### 3. Leetcode :exercises on Increasing Order Search Tree

**Given the root of a binary search tree, rearrange the tree in in-order so that the leftmost node in the tree is now the root of the tree, and every node has no left child and only one right child.**

```
// struct TreeNode* rotRight(struct TreeNode* root) {  
//   struct TreeNode* temp = root->left;  
//   struct TreeNode* ptr = root->left;  
//   while (ptr->right) {  
//     ptr = ptr->right;  
//   }  
//   ptr->right = root;  
//   root->left = NULL;  
//   return temp;  
// }
```

```
struct TreeNode* increasingBST(struct TreeNode* root) {  
    if (!root)  
        return root;  
  
    struct TreeNode* lft = increasingBST(root->left);  
    if (lft){  
        struct TreeNode* temp = lft;  
        while (temp->right)  
            temp = temp->right;  
        root->left = NULL;  
        temp->right = root;  
        root->right = increasingBST(root->right);  
        root = lft;  
    }  
    else  
        root->right = increasingBST(root->right);  
  
    return root;  
}
```

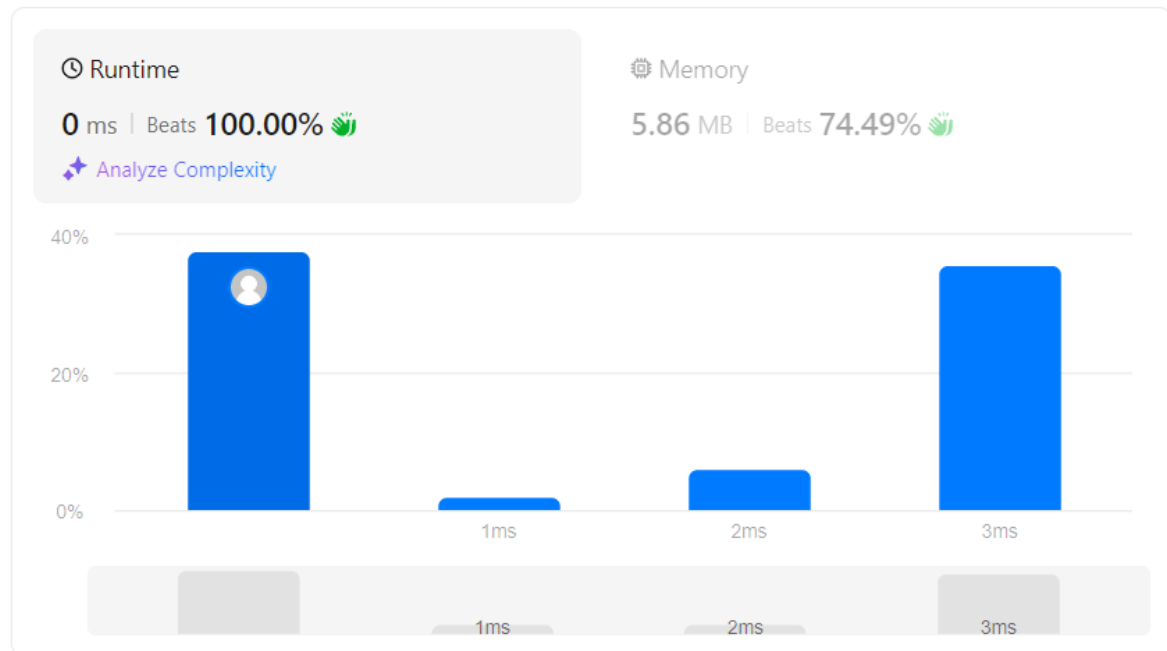


Accepted

Rohitgandhi02599 submitted at May 16, 2024 09:33

Editorial

Solution



Code | C

4. (a) Write a program to obtain the Topological ordering of vertices in a given digraph using DFS.

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

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

```
#define MAX_VERTICES 100
```

```
typedef struct {  
    int vertices[MAX_VERTICES];  
    int top;  
} Stack;
```

```
void push(Stack *s, int value) {  
    s->vertices[++s->top] = value;  
}
```

```
int pop(Stack *s) {
```

```

    return s->vertices[s->top--];
}

int isEmpty(Stack *s) {
    return s->top == -1;
}

void topologicalSortUtil(int v, int visited[], Stack *stack, int
graph[][MAX_VERTICES], int n) {
    visited[v] = 1;

    for (int i = 0; i < n; i++) {
        if (graph[v][i] && !visited[i]) {
            topologicalSortUtil(i, visited, stack, graph, n);
        }
    }
    push(stack, v);
}

void topologicalSort(int n, int graph[][MAX_VERTICES]) {
    Stack stack;
    stack.top = -1;
    int visited[MAX_VERTICES] = {0};

    for (int i = 0; i < n; i++) {
        if (!visited[i]) {
            topologicalSortUtil(i, visited, &stack, graph, n);
        }
    }
    while (!isEmpty(&stack)) {
        printf("%d ", pop(&stack));
    }
    printf("\n");
}

int main() {
    int n;

```

```

printf("Enter the number of vertices in the graph: ");
scanf("%d", &n);
int graph[MAX_VERTICES][MAX_VERTICES] = {0};

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

printf("\nTopological sorting of vertices:\n");
topologicalSort(n, graph);

return 0;
}

```

#### OUTPUT:

Sample Input:

```

mathematica Copy code

Enter the number of vertices in the graph: 6
Enter the adjacency matrix of the graph:
0 1 1 0 0 0
0 0 0 1 0 0
0 0 0 1 1 0
0 0 0 0 0 1
0 0 0 0 0 1
0 0 0 0 0 0

```

Sample Output:

```

yaml Copy code

Topological sorting of vertices:
0 2 4 1 3 5

```

**(b) Write a program to obtain the Topological ordering of vertices in a given digraph using Source Removal method.**

```
#include <stdio.h>

#include <stdlib.h>

#define MAX_VERTICES 100

typedef struct Queue
{
    int items[MAX_VERTICES];
    int front;
    int rear;
} Queue;

void enqueue(Queue *q, int value);
int dequeue(Queue *q);
int isEmpty(Queue *q);
void topologicalSort(int n, int graph[][MAX_VERTICES]);
int main()
{
    int n;
    printf("Enter the number of vertices in the graph: ");
    scanf("%d", &n);
    int graph[MAX_VERTICES][MAX_VERTICES] = {0};
    printf("Enter the adjacency matrix of the graph:\n");
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
```

```

        scanf("%d", &graph[i][j]);
    }
}
printf("\nTopological sorting of vertices:\n");
topologicalSort(n, graph);
return 0;
}

void topologicalSort(int n, int graph[][MAX_VERTICES])
{
    int indegree[MAX_VERTICES] = {0};
    Queue q;
    q.front = -1;
    q.rear = -1;

    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            if (graph[i][j] == 1)
            {
                indegree[j]++;
            }
        }
    }

    for (int i = 0; i < n; i++)
    {
        if (indegree[i] == 0)

```

```

        {
            enqueue(&q, i);
        }
    }
    while (!isEmpty(&q))
    {
        int vertex = dequeue(&q);
        printf("%d ", vertex);

        for (int i = 0; i < n; i++)
        {
            if (graph[vertex][i] == 1)
            {
                if (--indegree[i] == 0)
                {
                    enqueue(&q, i);
                }
            }
        }
    }
    printf("\n");
}

void enqueue(Queue *q, int value)
{
    if (q->rear == MAX_VERTICES - 1)
    {
        printf("Queue is full\n");
    }
}

```

```

    }
    else
    {
        if (q->front == -1)
        {
            q->front = 0;
        }
        q->rear++;
        q->items[q->rear] = value;
    }
}

```

```

int dequeue(Queue *q)
{
    int item;
    if (isEmpty(q))
    {
        printf("Queue is empty\n");
        item = -1;
    }
    else
    {
        item = q->items[q->front];
        q->front++;
        if (q->front > q->rear)
        {
            q->front = q->rear = -1;
        }
    }
}

```

```

    }
}
return item;
}
int isEmpty(Queue *q)
{
    if (q->rear == -1)
    {
        return 1;
    }
    else
    {
        return 0;
    }
}

```

```

Enter the number of vertices in the graph: 5
Enter the adjacency matrix of the graph:
0 1 1 0 0
0 0 1 0 0
0 0 0 1 1
0 0 0 0 1
0 0 0 0 0

Topological sorting of vertices:
0 1 2 3 4

```



- 5. Sort a given set of N integer elements using selection 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>

void selsort(int n, int a[]);

int main() {
    int a[15000], n, i, j, ch, temp;
    clock_t start, end;
    while (1) {
        printf("\n1: For manual entry of N value and array elements");
        printf("\n2: To display time taken for sorting number of elements N in the
range 500 to 14500");
        printf("\n3: To exit");
        printf("\nEnter your choice: ");
        scanf("%d", &ch);
        switch (ch) {
            case 1:
                printf("\nEnter the number of elements: ");
                scanf("%d", &n);
                printf("\nEnter array elements: ");
                for (i = 0; i < n; i++) {
```

```

        scanf("%d", &a[i]);
    }
    start = clock();
    selsort(n, a);
    end = clock();
    printf("\nSorted array is: ");
    for (i = 0; i < n; i++)
        printf("%d\t", a[i]);

    printf("\nTime taken to sort %d numbers is %f Secs", n, (((double)(end -
start)) / CLOCKS_PER_SEC));
    break;

```

case 2:

```

n = 500;
while (n <= 14500) {
    for (i = 0; i < n; i++) {
        //a[i] = rand() % 1000;
        a[i] = n - i;
    }
    start = clock();
    selsort(n, a);
    //Dummy loop to create delay
    for (j = 0; j < 500000; j++) {
        temp = 38 / 600;
    }
}

```

```

        end = clock();

        printf("\nTime taken to sort %d numbers is %f Secs", n, (((double)(end
- start)) / CLOCKS_PER_SEC));

        n = n + 1000;
    }

    break;

case 3:

    exit(0);

}

getchar();

}

return 0;

}

```

```

void selsort(int n, int a[]) {
    int i, j, t, small, pos;
    for (i = 0; i < n - 1; i++) {
        pos = i;
        small = a[i];
        for (j = i + 1; j < n; j++) {
            if (a[j] < small) {
                small = a[j];
                pos = j;
            }
        }
    }
}

```

```

        t = a[i];
        a[i] = a[pos];
        a[pos] = t;
    }
}

```

OUTPUT:

```

Enter array elements: 44 33 22 11

Sorted array is: 11      22      33      44
Time taken to sort 4 numbers is 0.000000 Secs
1: For manual entry of N value and array elements
2: To display time taken for sorting number of elements N in the range 500 to 14500
3: To exit
Enter your choice: 2

Time taken to sort 500 numbers is 0.000000 Secs
Time taken to sort 1500 numbers is 0.000000 Secs
Time taken to sort 2500 numbers is 0.000000 Secs
Time taken to sort 3500 numbers is 0.000000 Secs
Time taken to sort 4500 numbers is 0.016000 Secs
Time taken to sort 5500 numbers is 0.016000 Secs
Time taken to sort 6500 numbers is 0.015000 Secs
Time taken to sort 7500 numbers is 0.016000 Secs
Time taken to sort 8500 numbers is 0.015000 Secs
Time taken to sort 9500 numbers is 0.032000 Secs
Time taken to sort 10500 numbers is 0.032000 Secs
Time taken to sort 11500 numbers is 0.046000 Secs
Time taken to sort 12500 numbers is 0.047000 Secs
Time taken to sort 13500 numbers is 0.063000 Secs
Time taken to sort 14500 numbers is 0.078000 Secs
1: For manual entry of N value and array elements
2: To display time taken for sorting number of elements N in the range 500 to 14500
3: To exit
Enter your choice: 3

Process returned 0 (0x0)   execution time : 41.550 s
Press any key to continue.
|

```

**(B)Sort a given set of N integer elements using Merge Sort 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> /* To recognise exit function when compiling with gcc*/

void split(int[],int,int);

void combine(int[],int,int,int);

void main()
{
    int a[15000],n, i,j,ch, temp;

    clock_t start,end;

    while(1)
    {
        printf("\n1:For manual entry of N value and array elements");

        printf("\n2:To display time taken for sorting number of elements N in the range
        500 to 14500");

        printf("\n3:To exit");

        printf("\nEnter your choice:");

        scanf("%d", &ch);

        switch(ch)
        {

            case 1: printf("\nEnter the number of elements: ");

                    scanf("%d",&n);

                    printf("\nEnter array elements: ");

```

```

        for(i=0;i<n;i++)
        {
            scanf("%d",&a[i]);
        }
        start=clock();
        split(a,0,n-1);
        end=clock();
        printf("\nSorted array is: ");
        for(i=0;i<n;i++)
            printf("%d\t",a[i]);

        printf("\n Time taken to sort %d numbers is %f Secs",n, (((double)(end-
start))/CLOCKS_PER_SEC));

        break;
case 2:
    n=500;
    while(n<=14500) {
        for(i=0;i<n;i++)
        {
            //a[i]=random(1000);
            a[i]=n-i;
        }
        start=clock();

```

```

        split(a,0,n-1);

        //Dummy loop to create delay
        for(j=0;j<90000000;j++){ temp=38/600;}

        end=clock();

        printf("\n Time taken to sort %d numbers is %f Secs",n, (((double)(end-
start))/CLOCKS_PER_SEC));

        n=n+1000;

        }

        break;

    case 3: exit(0);

    }

    getchar();

    }

}

```

```

void split(int a[],int low,int high)

{

    int mid;

    if(low<high)

```

```

{
    mid=(low+high)/2;
    split(a,low,mid);
    split(a,mid+1,high);
    combine(a,low,mid,high);
}
}

```

```

void combine(int a[],int low,int mid,int high)

```

```

{
    int c[15000],i,j,k;
    i=k=low;
    j=mid+1;
    while(i<=mid&& j<=high)
    {
        if(a[i]<a[j])
        {
            c[k]=a[i];
            ++k;
            ++i;
        }
        else

```



```
{  
    c[k]=a[j];  
    ++k;  
    ++j;  
}  
}  
if(i>mid)  
{  
    while(j<=high)  
    {  
        c[k]=a[j];  
        ++k;  
        ++j;  
    }  
}  
if(j>high)  
{  
    while(i<=mid)  
    {  
        c[k]=a[i];  
        ++k;  
        ++i;
```

```

    }

}

for(i=low;i<=high;i++)

{

    a[i]=c[i];

}

}

```

- 6. Sort a given set of N integer elements using Quick 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>

#define MAX_N 10

#define LEFT 0

#define RIGHT 1

typedef struct
{
    int value;
    int direction;
} Element;

void printPermutations(int n);

void generatePermutations(Element permutation[], int n);

int findLargestMobile(Element permutation[], int n);

int main()
{

```

```

int n;

printf("Enter the number of elements (max %d): ", MAX_N);

scanf("%d", &n);

if (n > MAX_N || n <= 0)
{
    printf("Invalid input size. Please enter a valid number between 1 and %d.\n", MAX_N);
    return 1;
}

printf("Permutations of %d elements:\n", n);
printPermutations(n);
return 0;
}

void printPermutations(int n)
{
    Element permutation[MAX_N];
    for (int i = 0; i < n; i++)
    {
        permutation[i].value = i + 1;
        permutation[i].direction = LEFT;
    }
    for (int i = 0; i < n; i++)
    {
        printf("%d ", permutation[i].value);
    }
    printf("\n");
    generatePermutations(permutation, n);
}

```

```

}

void generatePermutations(Element permutation[], int n)
{
    while (true)
    {
        int mobileIdx = findLargestMobile(permutation, n);
        if (mobileIdx == -1)
        { break }
        int swapIdx = mobileIdx + (permutation[mobileIdx].direction == LEFT ? -1 : 1);
        Element temp = permutation[mobileIdx];
        permutation[mobileIdx] = permutation[swapIdx];
        permutation[swapIdx] = temp;
        for (int i = 0; i < n; i++)
        { if (permutation[i].value > permutation[swapIdx].value)
            {
                permutation[i].direction = (permutation[i].direction == LEFT) ? RIGHT : LEFT;
            }
        }
        for (int i = 0; i < n; i++)
        { printf("%d ", permutation[i].value);
        }
        printf("\n"); } }

int findLargestMobile(Element permutation[], int n)
{
    int mobileIdx = -1;
    int maxMobileValue = -1;
    for (int i = 0; i < n; i++)
    {

```

```

int direction = permutation[i].direction;

int adjacentIdx = i + (direction == LEFT ? -1 : 1);

if (adjacentIdx >= 0 && adjacentIdx < n &&
    permutation[i].value > permutation[adjacentIdx].value &&
    permutation[i].value > maxMobileValue)
{
    mobileIdx = i;

    maxMobileValue = permutation[i].value;
} }

return mobileIdx;
}

```

OUTPUT:

```

1:For manual entry of N value and array elements
2:To display time taken for sorting number of elements N in the range 500 to 14500
3:To exit
Enter your choice:1

Enter the number of elements: 8
Enter array elements: 5 3 1 9 8 2 4 7

Sorted array is: 1      2      3      4      5      7      8      9
Time taken to sort 8 numbers is 0.000000 Secs
1:For manual entry of N value and array elements
2:To display time taken for sorting number of elements N in the range 500 to 14500
3:To exit
Enter your choice:2
.
Time taken to sort 500 numbers is 0.000000 Secs
Time taken to sort 1500 numbers is 0.016000 Secs
Time taken to sort 2500 numbers is 0.000000 Secs
Time taken to sort 3500 numbers is 0.000000 Secs
Time taken to sort 4500 numbers is 0.000000 Secs
Time taken to sort 5500 numbers is 0.015000 Secs
Time taken to sort 6500 numbers is 0.000000 Secs
Time taken to sort 7500 numbers is 0.000000 Secs
Time taken to sort 8500 numbers is 0.031000 Secs
Time taken to sort 9500 numbers is 0.016000 Secs
Time taken to sort 10500 numbers is 0.031000 Secs
Time taken to sort 11500 numbers is 0.031000 Secs
Time taken to sort 12500 numbers is 0.047000 Secs
Time taken to sort 13500 numbers is 0.032000 Secs
Time taken to sort 14500 numbers is 0.062000 Secs
1:For manual entry of N value and array elements
2:To display time taken for sorting number of elements N in the range 500 to 14500
3:To exit
Enter your choice:3

Process returned 0 (0x0)   execution time : 33.657 s
Press any key to continue.

```

## 7.Implement Johnson Trotter algorithm to generate permutations.

```
#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;
    int j = 0;
    int k = left;
    while (i < n1 && j < n2)
    {
        if (L[i] <= R[j])
        {
            arr[k] = L[i];
            i++;
        }
        else
        {
            arr[k] = R[j];
            j++;
        }
        k++;
    }
}
```

```

        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);
    }
}

```

```

}

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

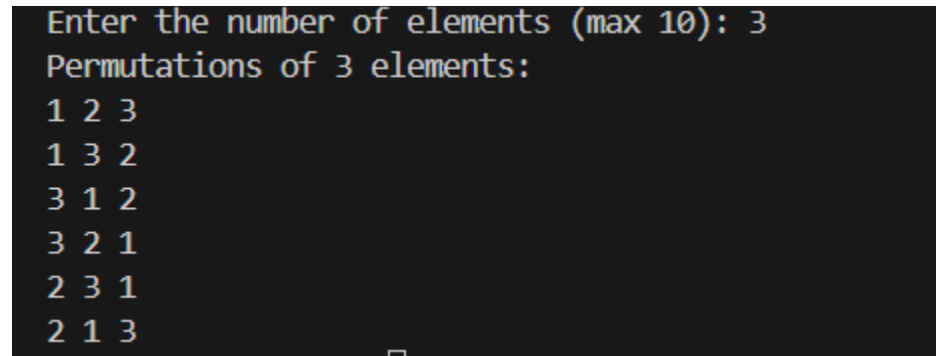
int main()
{
    int n;
    clock_t start, end;
    double cpu_time_used;
    printf("Enter the number of elements: ");
    scanf("%d", &n);
    int arr[n];
    printf("Enter %d integers:\n", n);
    for (int i = 0; i < n; i++)
        scanf("%d", &arr[i]);
    printf("Unsorted array: ");
    printArray(arr, n);
    start = clock();
    mergeSort(arr, 0, n - 1);
    end = clock();
    cpu_time_used = ((double)(end - start)) / CLOCKS_PER_SEC;
    printf("Sorted array: ");
    printArray(arr, n);

```



```
printf("Time taken to sort: %f seconds\n", cpu_time_used);  
return 0;  
}
```

OUTPUT:



```
Enter the number of elements (max 10): 3  
Permutations of 3 elements:  
1 2 3  
1 3 2  
3 1 2  
3 2 1  
2 3 1  
2 1 3
```

## B.Substring Matching Program.

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

```
#include <string.h>
```

```
void substringMatch(char *str, char *sub) {  
    int strLen = strlen(str);  
    int subLen = strlen(sub);  
    int found = 0;  
  
    for (int i = 0; i <= strLen - subLen; i++) {  
        int j;  
        for (j = 0; j < subLen; j++) {  
            if (str[i + j] != sub[j]) {  
                break;  
            }  
        }  
        if (j == subLen) {  
            printf("Substring found at index %d\n", i);  
            found = 1;  
        }  
    }  
  
    if (!found) {
```

```

        printf("Substring not found\n");
    }
}

int main() {
    char str[100], sub[100];

    printf("Enter the main string: ");
    gets(str); // Using gets() for simplicity, but it's better to use fgets() in practice.
    printf("Enter the substring to search for: ");
    gets(sub);

    substringMatch(str, sub);

    return 0;
}

```

OUTPUT:

Input:

```

vbnet Copy code

Enter the main string: This is a simple example.
Enter the substring to search for: simple

```

Output:

```

perl Copy code

Substring found at index 10

```

Input:

```

vbnet Copy code

Enter the main string: This is a simple example.
Enter the substring to search for: test

```

Output:

```

Copy code

Substring not found 


```

## C.Leetcode : exercise on Find Kth Largest Integer

```
int cmp(const void*a,const void*b) {  
    const char* str1 = *(const char**)a;  
    const char* str2 = *(const char**)b;  
  
    if (strlen(str1) == strlen(str2)) {  
        return strcmp(str1, str2);  
    }  
    return strlen(str1) - strlen(str2);  
}  
  
char * kthLargestNumber(char ** nums, int numsSize, int k){  
    qsort(nums,numsSize,sizeof(char*),cmp);  
    return nums[numsSize-k];  
}
```

Accepted

 Rohitgandhi02599 submitted at Jun 13, 2024 10:00

 Solution

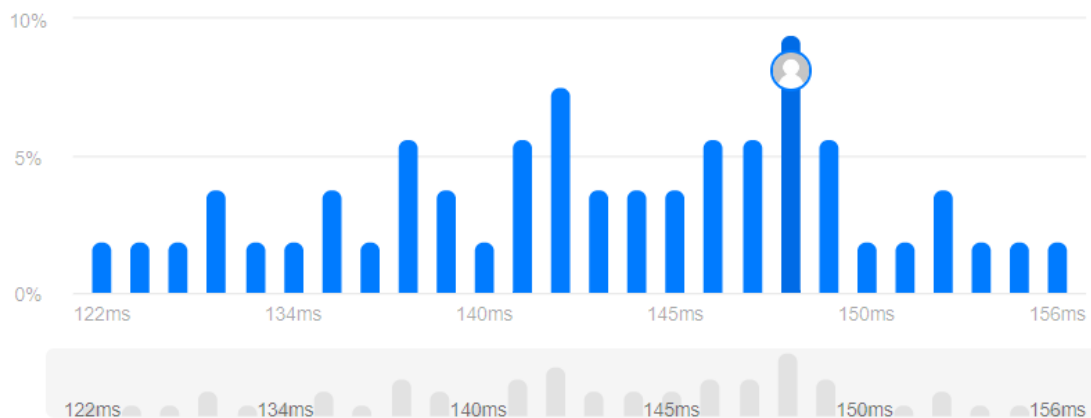
⌚ Runtime

148 ms | Beats 33.96%

 Analyze Complexity

💾 Memory

22.89 MB | Beats 37.74%



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

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

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

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

```
void 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 - 1; 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);
    }
}

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

int main()
{
    int n;
    clock_t start, end;
    double cpu_time_used;
    printf("Enter the 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]);
printf("Unsorted array: ");
printArray(arr, n);
start = clock();
quickSort(arr, 0, n - 1);
end = clock();
cpu_time_used = ((double)(end - start)) / CLOCKS_PER_SEC;
printf("Sorted array: ");
printArray(arr, n);
printf("Time taken to sort: %f seconds\n", cpu_time_used);
return 0;
}

```

```

Enter the number of elements: 7
Enter 7 integers:
4 2 3 5 1 6 7
Unsorted array: 4 2 3 5 1 6 7
Sorted array: 1 2 3 4 5 6 7
Time taken to sort: 0.000000 seconds

```

## B.Floyds Algorithm:

```

#include <stdio.h>
#include <limits.h>
#define V 4
void printSolution(int dist[][V])
{
    printf("Shortest distances between every pair of vertices:\n");
    for (int i = 0; i < V; i++)
    {
        for (int j = 0; j < V; j++)

```

```

    {
        if (dist[i][j] == INT_MAX)
            printf("INF\t");
        else
            printf("%d\t", dist[i][j]);
    }
    printf("\n");
}

void floydWarshall(int graph[][V])
{
    int dist[V][V];
    for (int i = 0; i < V; i++)
        for (int j = 0; j < V; j++)
            dist[i][j] = graph[i][j];
    for (int k = 0; k < V; k++)
    {
        for (int i = 0; i < V; i++)
        {
            for (int j = 0; j < V; j++)
            {
                if (dist[i][k] != INT_MAX && dist[k][j] != INT_MAX && dist[i][k] + dist[k][j] < dist[i][j])
                    dist[i][j] = dist[i][k] + dist[k][j];
            }
        }
    }
}

```

```

    printSolution(dist);
}

int main()
{
    int graph[V][V] = {
        {0, 5, INT_MAX, 10},
        {INT_MAX, 0, 3, INT_MAX},
        {INT_MAX, INT_MAX, 0, 1},
        {INT_MAX, INT_MAX, INT_MAX, 0}};
    floydWarshall(graph);
    return 0;
}

```

Shortest distances between every pair of vertices:

0	5	8	9
INF	0	3	4
INF	INF	0	1
INF	INF	INF	0

### 9.(a ) 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 W, int wt[], int val[], int n)
{
    int i, w;
    int K[n + 1][W + 1];

```



```

for (i = 0; i <= n; i++)
{
    for (w = 0; w <= W; w++)
    {
        if (i == 0 || w == 0)
            K[i][w] = 0;
        else if (wt[i - 1] <= w)
            K[i][w] = max(val[i - 1] + K[i - 1][w - wt[i - 1]], K[i - 1][w]);
        else
            K[i][w] = K[i - 1][w];
    }
}
return K[n][W];
}

int main()
{
    int n, W;
    printf("Enter number of items: ");
    scanf("%d", &n);
    int val[n], wt[n];
    printf("Enter values and weights of items:\n");
    for (int i = 0; i < n; i++)
    {
        printf("Enter value and weight for item %d: ", i + 1);
        scanf("%d %d", &val[i], &wt[i]);
    }
    printf("Enter maximum weight capacity of knapsack: ");

```

```

scanf("%d", &W);
int max_value = knapsack(W, wt, val, n);
printf("Maximum value that can be obtained: %d\n", max_value);
return 0;
}

```

```

Enter number of items: 3
Enter values and weights of items:
Enter value and weight for item 1: 60 10
Enter value and weight for item 2: 100 20
Enter value and weight for item 3: 120 30
Enter maximum weight capacity of knapsack: 50
Maximum value that can be obtained: 220

```

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

```

#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
#define V 5
int minKey(int key[], int mstSet[])
{
    int min = INT_MAX, min_index;
    for (int v = 0; v < V; v++)
        if (mstSet[v] == 0 && key[v] < min)
            min = key[v], min_index = v;
    return min_index;
}

```

```

}

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

void primMST(int graph[V][V])
{
    int parent[V];
    int key[V];
    int mstSet[V];
    for (int i = 0; i < V; i++)
        key[i] = INT_MAX, mstSet[i] = 0;
    key[0] = 0;
    parent[0] = -1;
    for (int count = 0; count < V - 1; count++)
    {
        int u = minKey(key, mstSet);
        mstSet[u] = 1;
        for (int v = 0; v < V; v++)
            if (graph[u][v] && mstSet[v] == 0 && graph[u][v] < key[v])
                parent[v] = u, key[v] = graph[u][v];
    }
    printMST(parent, graph);
}

int main()

```

```

{
    int graph[V][V] = {
        {0, 2, 0, 6, 0},
        {2, 0, 3, 8, 5},
        {0, 3, 0, 0, 7},
        {6, 8, 0, 0, 9},
        {0, 5, 7, 9, 0},
    };
    primMST(graph);
    return 0;
}

```

Edge	Weight
0 - 1	2
1 - 2	3
0 - 3	6
1 - 4	5

**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 <stdbool.h>

#define MAX 100

```

```

int minDistance(int dist[], bool sptSet[], int V)
{

```

```

    int min = INT_MAX, min_index;
    for (int v = 0; v < V; v++)
        if (sptSet[v] == false && dist[v] <= min)
            min = dist[v], min_index = v;
    return min_index;
}

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

void dijkstra(int graph[MAX][MAX], int src, int V)
{
    int dist[V];
    bool sptSet[V];

    for (int i = 0; i < V; i++)
        dist[i] = INT_MAX, sptSet[i] = false;

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

```

```

    }
    printSolution(dist, V);
}

int main()
{
    int V;
    printf("Enter the number of vertices: ");
    scanf("%d", &V);
    int graph[MAX][MAX];
    printf("Enter the adjacency matrix (enter 0 if there is no edge between two vertices):\n");
    for (int i = 0; i < V; i++)
        for (int j = 0; j < V; j++)
            scanf("%d", &graph[i][j]);
    int src;
    printf("Enter the source vertex: ");
    scanf("%d", &src);
    dijkstra(graph, src, V);
    return 0;
}

```

```

Enter the number of vertices: 5
Enter the adjacency matrix (enter 0 if there is no edge between two vertices):
0 10 0 0 5
10 0 1 0 2
0 1 0 4 0
0 0 4 0 3
5 2 0 3 0
Enter the source vertex: 0
Vertex    Distance from Source
0          0
1          7
2          8
3          8
4          5

```

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

```
#include <stdio.h>

#include <stdlib.h>

#define MAX_VERTICES 20

struct Edge
{
    int src, dest, weight;
};

void Union(int parent[], int rank[], int x, int y);
int find(int parent[], int i);
void KruskalMST(struct Edge *edges, int V, int E);
int find(int parent[], int i)
{
    if (parent[i] != i)
        parent[i] = find(parent, parent[i]);
    return parent[i];
}

void Union(int parent[], int rank[], int x, int y)
{
    int xroot = find(parent, x);
    int yroot = find(parent, y);
    if (rank[xroot] < rank[yroot])
        parent[xroot] = yroot;
    else if (rank[xroot] > rank[yroot])
        parent[yroot] = xroot;
```

```

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

int compareEdges(const void *a, const void *b)
{
    struct Edge *edge1 = (struct Edge *)a;
    struct Edge *edge2 = (struct Edge *)b;
    return edge1->weight - edge2->weight;
}

void KruskalMST(struct Edge *edges, int V, int E)
{
    struct Edge result[V];
    int e = 0;
    int i = 0;
    qsort(edges, E, sizeof(struct Edge), compareEdges);
    int parent[V];
    int rank[V];
    for (int v = 0; v < V; ++v)
    {
        parent[v] = v;
        rank[v] = 0;
    }

    while (e < V - 1 && i < E)
    {

```



```

    struct Edge next_edge = edges[i++];
    int u = find(parent, next_edge.src);
    int v = find(parent, next_edge.dest);
    if (u != v)
    {
        result[e++] = next_edge;
        Union(parent, rank, u, v);
    }
}
printf("Edges in the Minimum Spanning Tree:\n");
for (i = 0; i < e; ++i)
{
    printf("%d -- %d == %d\n", result[i].src, result[i].dest, result[i].weight);
}
}

int main()
{
    int V, E;
    printf("Enter the number of vertices: ");
    scanf("%d", &V);
    printf("Enter the number of edges: ");
    scanf("%d", &E);
    struct Edge edges[E];
    printf("Enter the source, destination, and weight of each edge:\n");
    for (int i = 0; i < E; ++i)
    {
        scanf("%d %d %d", &edges[i].src, &edges[i].dest, &edges[i].weight);
    }
}

```

```
}  
KruskalMST(edges, V, E);  
return 0;  
}
```

```
Enter the number of vertices: 4  
Enter the number of edges: 5  
Enter the source, destination, and weight of each edge:  
0 1 10  
0 2 6  
0 3 5  
1 3 15  
1 2 6  
Edges in the Minimum Spanning Tree:  
0 -- 3 == 5  
0 -- 2 == 6  
1 -- 2 == 6
```