VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Analysis and Design of Algorithms

Submitted by

SHRADDHA (1BM22CS357)

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 **SHRADDHA (1BM22CS357),** 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.	Dr. Jyothi S Nayak
Designation	Professor and Head
Department of CSE	Department of CSE
BMSCE, Bengaluru	BMSCE, Bengaluru

Index Sheet

9	➢ Implement 0/1 Knapsack problem using dynamic programming. Find Minimum Cost Spanning Tree of a given undirected graph using Prim's algorithm.	
10	➢ From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm. ➢ Find Minimum Cost Spanning Tree of a given undirected graph using Kruskal's algorithm. ➢ Implement Fractional Knapsack using Greedy technique. Implement "N-Queens Problem" using Backtracking	

Course Outcome

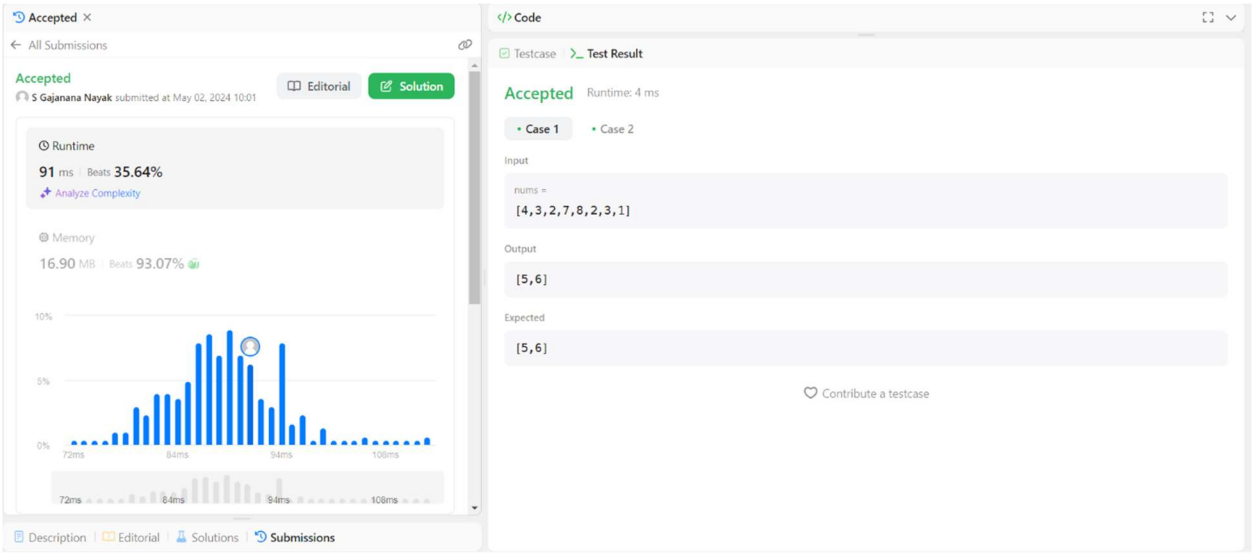
CO1	Analyze time complexity of Recursive and Non-recursive algorithms using asymptotic notations.
CO2	Apply various design techniques for the given problem.
CO3	Apply the knowledge of complexity classes P, NP, and NP-Complete and prove certain problems are NP-Complete
CO4	Design efficient algorithms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

LAB-1:

Leetcode-1 : Find Disappeared Numbers in the array

```
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;
        }
    }
    return nums;
}
```

Output:

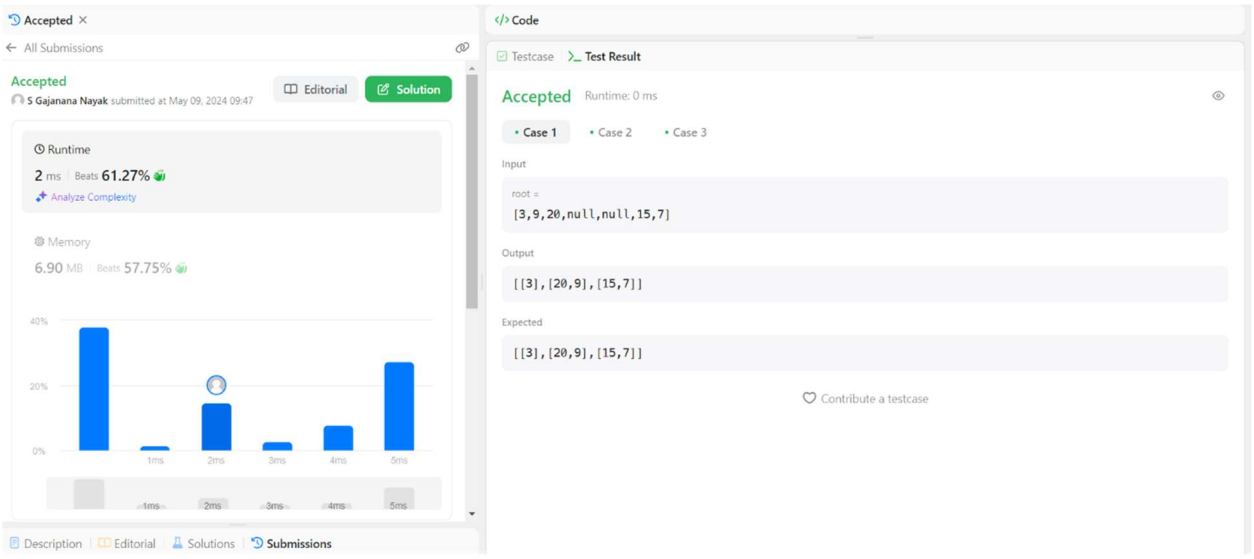


LAB-2:

Leetcode-2: Zigzag Traversal of BST

```
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)++;
    }
    return ans;
}
```

Output:



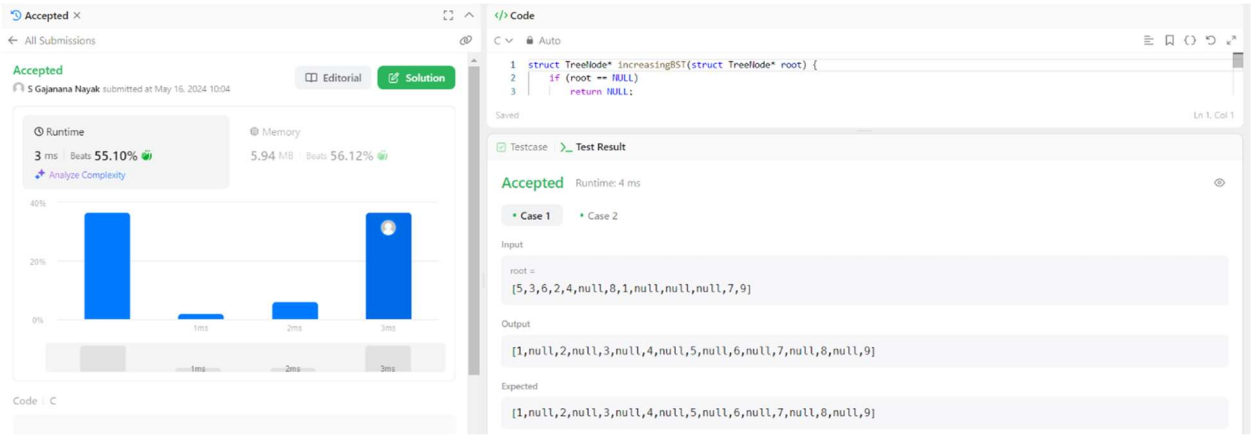
LAB-3:

Leetcode-3: Increasing Order Search Tree

```
struct TreeNode* increasingBST(struct TreeNode* root) {
    if (root == NULL)
        return NULL;
    struct TreeNode* newRoot = NULL;
    struct TreeNode* prev = NULL;
    struct TreeNode* curr = root;
    struct TreeNode* stack[2000];
    int top = -1;

    while (curr != NULL || top != -1) {
        while (curr != NULL) {
            stack[++top] = curr;
            curr = curr->left;
        }
        curr = stack[top--];
        if (newRoot == NULL) {
            newRoot = curr;
        } else {
            prev->right = curr;
        }
        prev = curr;
        curr->left = NULL;
```

```
        curr = curr->right;
    }
    return newRoot;
}
```

Output:



LAB-4:

Topological ordering using DFS

```
#include <stdio.h>
#define MAX 100
int adj[MAX][MAX]; // Adjacency matrix
int visited[MAX];  // Visited array
int stack[MAX];    // Stack for storing the topological order
int top = -1;      // Top of the stack
int n;             // Number of vertices

void push(int v) {
    stack[++top] = v;
}

int pop() {
    return stack[top--];
}

void dfs(int v) {
    visited[v] = 1;
    for (int i = 0; i < n; i++) {
        if (adj[v][i] && !visited[i]) {
            dfs(i);
```

```
        }
    }
    push(v);
}

void topologicalSort() {
    for (int i = 0; i < n; i++) {
        visited[i] = 0;
    }
    for (int i = 0; i < n; i++) {
        if (!visited[i]) {
            dfs(i);
        }
    }
    while (top != -1) {
        printf("%d ", pop());
    }
    printf("\n");
}
int main() {
    int edges, start, end;
    printf("Enter the number of vertices: ");
    scanf("%d", &n);
    printf("Enter the number of edges: ");
    scanf("%d", &edges);
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            adj[i][j] = 0;
        }
    }
    for (int i = 0; i < edges; i++) {
        printf("Enter start and end vertices of edge %d: ", i + 1);
        scanf("%d %d", &start, &end);
        adj[start][end] = 1;
    }
    printf("Topological sort: ");
    topologicalSort();
    return 0;
}
```

Output:

```
Enter the number of vertices: 5
Enter the number of edges: 6
Enter start and end vertices of edge 1: 0 1
Enter start and end vertices of edge 2: 0 2
Enter start and end vertices of edge 3: 1 3
Enter start and end vertices of edge 4: 1 4
Enter start and end vertices of edge 5: 3 5
Enter start and end vertices of edge 6: 4 5
Topological sort: 0 2 1 4 3
```

Topological sorting using Source Removal Method

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

int st[100];
int top = -1;

void degree(int adj[][20], int n) {
   int indegree[20];
   int sum = 0;
   for (int j = 0; j < n; j++) {
      sum = 0;
      for (int i = 0; i < n; i++) {
         sum = sum + adj[i][j];
      }
      indegree[j] = sum;
   }

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

   while (top != -1) {
      int u = st[top];
      top--;
      printf("%d ", u);
```

```
      for (int v = 0; v < n; v++) {
         if (adj[u][v] == 1) {
            indegree[v]--;
            if (indegree[v] == 0) {
               top++;
               st[top] = v;
            }
         }
      }
   }
}

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

   int adj[20][20];

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

   printf("Topological order of nodes: ");
   degree(adj, n);

   return 0;
}
```

Output:

```
Enter the number of nodes: 4
Enter the adjacency matrix:
0 1 1 0
0 0 1 0
0 0 0 1
0 0 0 0
Topological order of nodes: 0 1 2 3
```

LAB-5:

Merge Sort

```
#include <stdio.h>

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

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

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

   while (i <= mid) {
      c[k++] = a[i++];
```

```
    }

    while (j <= high) {
        c[k++] = a[j++];
    }

    for (i = low; i <= high; i++) {
        a[i] = c[i];
    }
}

int main() {
    int n, i;
    int a[1000]; // Adjust size as needed

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

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

    split(a, 0, n - 1);

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

    return 0;
}
```

Output:

```
Enter the number of elements: 4

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 0.032000 Secs
 Time taken to sort 1500 numbers is 0.046000 Secs
 Time taken to sort 2500 numbers is 0.032000 Secs
 Time taken to sort 3500 numbers is 0.047000 Secs
 Time taken to sort 4500 numbers is 0.046000 Secs
 Time taken to sort 5500 numbers is 0.032000 Secs
 Time taken to sort 6500 numbers is 0.047000 Secs
 Time taken to sort 7500 numbers is 0.031000 Secs
 Time taken to sort 8500 numbers is 0.047000 Secs
 Time taken to sort 9500 numbers is 0.047000 Secs
 Time taken to sort 10500 numbers is 0.031000 Secs
 Time taken to sort 11500 numbers is 0.047000 Secs
 Time taken to sort 12500 numbers is 0.047000 Secs
 Time taken to sort 13500 numbers is 0.031000 Secs
 Time taken to sort 14500 numbers is 0.047000 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 : 95.566 s
Press any key to continue.
```

Selection Sort

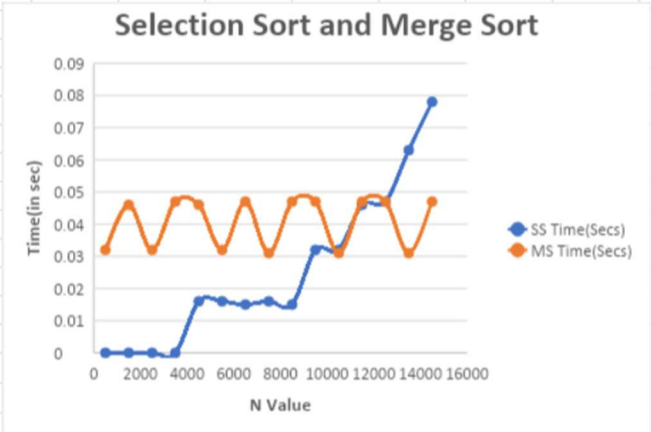
```
#include<stdio.h>
#include<time.h>
#include<stdlib.h>
void selsort(int n,int a[]);

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();
       selsort(n,a);
       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();
       selsort(n,a);
       for(j=0;j<500000;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 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:

N value	SS Time(Secs)	MS Time(Secs)
500	0	0.032
1500	0	0.046
2500	0	0.032
3500	0	0.047
4500	0.016	0.046
5500	0.016	0.032
6500	0.015	0.047
7500	0.016	0.031
8500	0.015	0.047
9500	0.032	0.047
10500	0.032	0.031
11500	0.046	0.047
12500	0.047	0.047
13500	0.063	0.031
14500	0.078	0.047



```
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: 5

Enter array elements: 45 53 33 87 16

Sorted array is: 16      33      45      53      87
 Time taken to sort 5 numbers is 0.000002 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 0.001472 Secs
 Time taken to sort 1500 numbers is 0.004287 Secs
 Time taken to sort 2500 numbers is 0.009888 Secs
 Time taken to sort 3500 numbers is 0.017972 Secs
 Time taken to sort 4500 numbers is 0.028829 Secs
 Time taken to sort 5500 numbers is 0.042597 Secs
 Time taken to sort 6500 numbers is 0.059367 Secs
 Time taken to sort 7500 numbers is 0.079522 Secs
 Time taken to sort 8500 numbers is 0.102420 Secs
 Time taken to sort 9500 numbers is 0.130204 Secs
 Time taken to sort 10500 numbers is 0.155147 Secs
 Time taken to sort 11500 numbers is 0.182382 Secs
 Time taken to sort 12500 numbers is 0.219833 Secs
 Time taken to sort 13500 numbers is 0.256604 Secs
 Time taken to sort 14500 numbers is 0.294683 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
```

LAB-6:

Quick Sort

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

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

```
int partition(int a[], int low, int high){
   int pivot = a[low];
   int i = low + 1;
   int j = high;

   while (i <= j) {
      while (i <= j && a[i] <= pivot)
         i++;
      while (i <= j && a[j] > pivot)
         j--;
      if (i < j)
         swap(&a[i], &a[j]);
   }
   swap(&a[low], &a[j]);
   return j;
}

void quicksort(int a[],int low,int high){
   int point;
   if(low<high){
         point=partition(a,low,high);
         quicksort(a,low,point-1);
         quicksort(a,point+1,high);
   }
}

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

   while (1) {
      printf("\n 1:For manual entry of N value and array elements");
      printf("\n 2:To display time taken for sorting number of elements N in the range 500 to
14500");
      printf("\n 3:To exit");
      printf("\nEnter your choice:");
      scanf("%d", &ch);

      switch (ch) {
         case 1:
```

```
            printf("\nEnter the number of elements: ");
            scanf("%d", &n);
            printf("Enter array elements: ");
            for (i = 0; i < n; i++) {
               scanf("%d", &a[i]);
            }
            start = clock();
            quicksort(a,0,n-1);
            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] = n - i;
               }
               start = clock();
               quicksort(a, 0, n - 1);
                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();
   }
}
```

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: 6
Enter array elements: 1 45 78 25 33 72

Sorted array is: 1      25      33      45      72      78
Time taken to sort 6 numbers is 0.000001 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 0.000732 Secs
Time taken to sort 1500 numbers is 0.002560 Secs
Time taken to sort 2500 numbers is 0.004844 Secs
Time taken to sort 3500 numbers is 0.007802 Secs
Time taken to sort 4500 numbers is 0.013731 Secs
Time taken to sort 5500 numbers is 0.020636 Secs
Time taken to sort 6500 numbers is 0.029014 Secs
Time taken to sort 7500 numbers is 0.035724 Secs
Time taken to sort 8500 numbers is 0.057654 Secs
Time taken to sort 9500 numbers is 0.071883 Secs
Time taken to sort 10500 numbers is 0.085053 Secs
Time taken to sort 11500 numbers is 0.098687 Secs
Time taken to sort 12500 numbers is 0.097453 Secs
Time taken to sort 13500 numbers is 0.108320 Secs
Time taken to sort 14500 numbers is 0.128737 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
```

LAB-7:

Johnson Trotter Algorithm

```
#include <stdio.h>
#include <stdlib.h>
int flag = 0;
int swap(int *a,int *b)
{
   int t =*a;
   *a = *b;
   *b = t;
}
int search(int arr[],int num,int mobile)
```

```
{
   int g;
   for(g=0; g<num; g++) {
      if(arr[g] == mobile)
         return g+1;
      else          {
         flag++;
      }
   }
   return -1;
}

int find_Moblie(int arr[],int d[],int num)
{
   int mobile = 0;
   int mobile_p =0;
   int i;
   for(i=0; i<num; i++) {
      if((d[arr[i]-1] == 0) && i != 0) {
         if(arr[i]>arr[i-1] && arr[i]>mobile_p) {
            mobile = arr[i];
            mobile_p = mobile;
         }          else          {
            flag++;
         }
      } else if((d[arr[i]-1] == 1) & i != num-1) {
         if(arr[i]>arr[i+1] && arr[i]>mobile_p) {
            mobile = arr[i];
            mobile_p = mobile;
         }          else          {
            flag++;
         }
      }       else {
         flag++;
      }
   }
   if((mobile_p == 0) && (mobile == 0))       return 0;
   else       return mobile;
}

void permutations(int arr[],int d[],int num)
```

```
{
   int i;
   int mobile = find_Moblie(arr,d,num);
   int pos = search(arr,num,mobile);
   if(d[arr[pos-1]-1]==0)        swap(&arr[pos-1],&arr[pos-2]);
   else
      swap(&arr[pos-1],&arr[pos]);
   for(int i=0; i<num; i++) {
      if(arr[i] > mobile) {
         if(d[arr[i]-1]==0)               d[arr[i]-1] = 1;
         else               d[arr[i]-1] = 0;
      }
   }
   for(i=0; i<num; i++) {
      printf(" %d ",arr[i]);
   }
}

int factorial(int k)
{
   int f = 1;
   int i = 0;
   for(i=1; i<k+1; i++) {
      f = f*i;
   }
   return f;
}
int main()
{
   int num =0;
   int i;
   int j;
   int z =0;
   printf("Johnson trotter algorithm to find all permutations of given numbers \n");
   printf("Enter the number\n");
   scanf("%d",&num);
   int arr[num],d[num];
   z = factorial(num);
   printf("total permutations = %d",z);
   printf("\nAll possible permutations are: \n");
   for(i=0; i<num; i++) {
```

```
        d[i] = 0;
        arr[i] = i+1;
        printf(" %d ",arr[i]);
    }
    printf("\n");
    for(j=1; j<z; j++) {
        permutations(arr,d,num);
        printf("\n");
    }
    return 0;
}
```

Output:

```
Johnson trotter algorithm to find all permutations of given numbers
Enter the number
3
total permutations = 6
All possible permutations are:
 1  2  3
 1  3  2
 3  1  2
 3  2  1
 2  3  1
 2  1  3
```

Pattern Matching Program

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

// Function to find the index of the first occurrence of the pattern in the text
int substringMatch(char text[], char pattern[]) {
    int textLength = strlen(text);
    int patternLength = strlen(pattern);

    for (int i = 0; i <= textLength - patternLength; i++) {
        int j;
        for (j = 0; j < patternLength; j++) {
```

```
            if (text[i + j] != pattern[j]) {
                break;
            }
        }
        if (j == patternLength) {
            return i; // Pattern found at index i
        }
    }
    return -1; // Pattern not found
}

int main() {
    char text[100], pattern[100];

    printf("Enter the main text: ");
    scanf("%s", text);

    printf("Enter the pattern to search: ");
    scanf("%s", pattern);

    int index = substringMatch(text, pattern);
    if (index != -1) {
        printf("Substring found at index: %d\n", index);
    } else {
        printf("Substring not found.\n");
    }

    return 0;
}
```

Output:

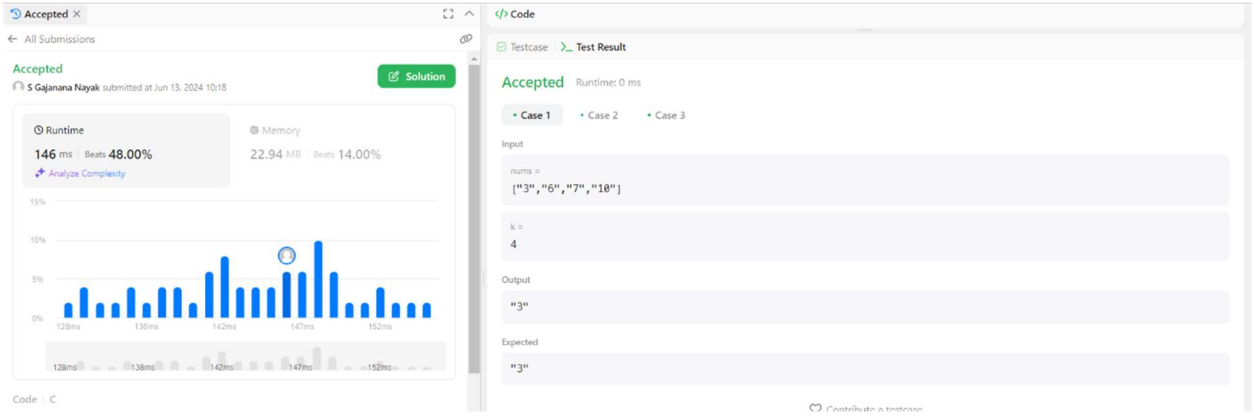
```
Enter the main text: hello_brother
Enter the pattern to search: bro
Substring found at index: 6
```

Leetcode-4: Find kth Largest Integer in the array:

```
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];

}
```

Output:



LAB-8:

Heap Sort

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

// Function to swap two elements
void swap(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

// Function to heapify a subtree rooted with node i which is an index in arr[]
void heapify(int arr[], int N, int i) {
```

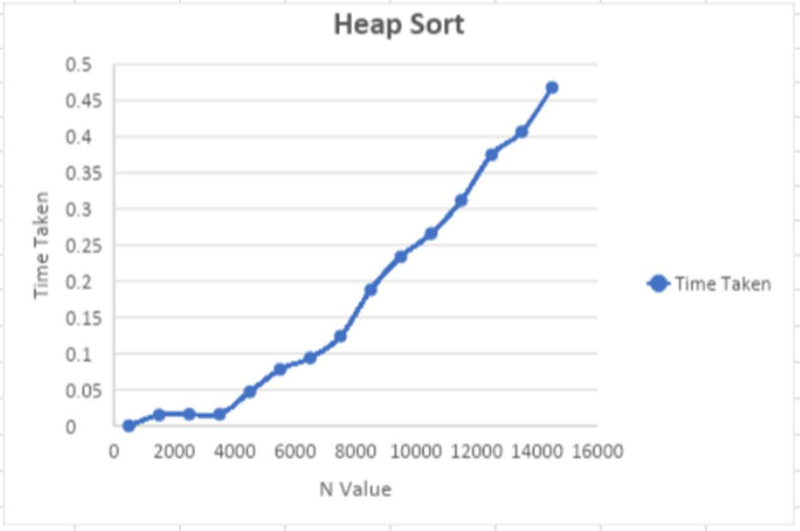
```
    int largest = i; // Initialize largest as root
    int left = 2 * i + 1; // left = 2*i + 1
    int right = 2 * i + 2; // right = 2*i + 2

    // If left child is larger than root
    if (left < N && arr[left] > arr[largest])
        largest = left;

    // If right child is larger than largest so far
    if (right < N && arr[right] > arr[largest])
        largest = right;

    // If largest is not root
    if (largest != i) {
        swap(&arr[i], &arr[largest]);

        // Recursively heapify the affected sub-tree
        heapify(arr, N, largest);
    }
}

// Function to do heap sort
void heapSort(int arr[], int N) {
    // Build heap (rearrange array)
    for (int i = N / 2 - 1; i >= 0; i--)
        heapify(arr, N, i);

    // One by one extract an element from heap
    for (int i = N - 1; i > 0; i--) {
        // Move current root to end
        swap(&arr[0], &arr[i]);

        // call max heapify on the reduced heap
        heapify(arr, i, 0);
    }
}

// Function to print an array
void printArray(int arr[], int N) {
    for (int i = 0; i < N; ++i)
        printf("%d ", arr[i]);
```

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

int main() {
    int N;

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

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

    // Record the start time
    clock_t start, end;
    double cpu_time_used;

    start = clock();

    // Call heap sort
    heapSort(arr, N);

    // Record the end time
    end = clock();

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

    printf("Sorted array is: \n");
    printArray(arr, N);
    printf("Time taken for sorting: %f seconds\n", cpu_time_used);

    return 0;
}
```

Output:

```
Enter number of elements: 5
Enter 5 elements: 4 10 5 3 1
Sorted array is:
1 3 4 5 10
Time taken for sorting: 0.000001 seconds
```

N Value	Time Taken
500	0
1500	0.015
2500	0.016
3500	0.016
4500	0.047
5500	0.078
6500	0.094
7500	0.124
8500	0.188
9500	0.234
10500	0.266
11500	0.312
12500	0.375
13500	0.407
14500	0.468



Floyd's Algorithm

```
#include <stdio.h>
#define V 4
#define INF 99999

void printSolution(int dist[][V]);

void floydWarshall(int dist[][V])
{
   int i, j, k;
   for (k = 0; k < V; k++) {
      for (i = 0; i < V; i++) {
         for (j = 0; j < V; j++) {
            if (dist[i][k] + dist[k][j] < dist[i][j])
               dist[i][j] = dist[i][k] + dist[k][j];
```

```
        }
      }
   }

   printSolution(dist);
}

void printSolution(int dist[][V])
{
   printf(
      "The following matrix shows the 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] == INF)
            printf("%7s", "INF");
         else
            printf("%7d", dist[i][j]);
      }
      printf("\n");
   }
}

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

   floydWarshall(graph);
   return 0;
}
```

Output:

```
The following matrix shows the shortest distances between every pair of vertices
      0      5      8      9
    INF      0      3      4
    INF    INF      0      1
    INF    INF    INF      0
```

LAB-9:

Knapsack using Dynamic Programming

```
#include <stdio.h>

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

void knapsack(int n, int W, int weights[], int values[]) {
   int i, w;
   int dp[n+1][W+1];

   for (i = 0; i <= n; i++) {
      for (w = 0; w <= W; w++) {
         if (i == 0 || w == 0) {
            dp[i][w] = 0;
         } else if (weights[i-1] <= w) {
            dp[i][w] = max(dp[i-1][w], dp[i-1][w - weights[i-1]] + values[i-1]);
         } else {
            dp[i][w] = dp[i-1][w];
         }
      }
   }

   printf("DP Table:\n");
   for (i = 0; i <= n; i++) {
      for (w = 0; w <= W; w++) {
         printf("%d\t", dp[i][w]);
      }
      printf("\n");
   }

   printf("Selected items: ");
   int res = dp[n][W];
   w = W;
   for (i = n; i > 0 && res > 0; i--) {
      if (res == dp[i-1][w])
         continue;
      else {
         printf("%d ", i);
```

```
            res = res - values[i-1];
            w = w - weights[i-1];
        }
    }
    printf("\nMaximum profit: %d\n", dp[n][W]);
}

int main() {
    int n = 4;
    int weights[] = {2, 3, 4, 5};
    int values[] = {3, 4, 5, 8};
    int W = 5;

    knapsack(n, W, weights, values);
    return 0;
}
```

Output:

```
DP Table:
0       0       0       0       0       0
0       0       3       3       3       3
0       0       3       4       4       7
0       0       3       4       5       7
0       0       3       4       5       8
Selected items: 4
Maximum profit: 8
```

Prims Algorithm

```
#include <stdio.h>
#include <limits.h>
#define N 4
int minKey(int key[], int mstSet[]) {
    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 printMST(int parent[], int graph[N][N]) {
   printf("Edge \tWeight\n");
   for (int i = 1; i < N; i++)
      printf("%d - %d \t%d \n", parent[i], i, graph[i][parent[i]]);
}
void primMST(int graph[N][N]) {
   int parent[N];
   int key[N];
   int mstSet[N];
   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);
      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];
   }
   printMST(parent, graph);
}

int main() {
   int graph[N][N] = { { 0, 10, 6, 5 },
                       { 10, 0, 0, 15 },
                       { 6, 0, 0, 4 },
                       { 5, 15, 4, 0 } };
   primMST(graph);

   return 0;
}
```

Output:

```
Edge    Weight
0 - 1   10
3 - 2   4
0 - 3   5
```

LAB-10:

Dijkstra's Algorithm

```
#include <stdio.h>
#define INF 9999
#define MAX 10

void dijkstra(int c[MAX][MAX], int n, int src) {
   int dist[MAX], vis[MAX], count, min, u;

   // Initialize distances and visited nodes
   for (int j = 0; j < n; j++) {
      dist[j] = c[src][j];
      vis[j] = 0;
   }

   dist[src] = 0;
   vis[src] = 1;
   count = 1;

   while (count != n) {
      min = INF;

      // Find the unvisited node with the smallest distance
      for (int j = 0; j < n; j++) {
         if (dist[j] < min && vis[j] != 1) {
            min = dist[j];
            u = j;
         }
      }

      vis[u] = 1;
      count++;
      // Update the distances of the neighbors of the selected node
      for (int j = 0; j < n; j++) {
         if (min + c[u][j] < dist[j] && vis[j] != 1) {
            dist[j] = min + c[u][j];
         }
      }
   }
   // Output the shortest distances
```

```
    printf("Shortest distances are:\n");
    for (int j = 0; j < n; j++) {
        printf("From %d to %d: %d\n", src, j, dist[j]);
    }
}
int main() {
    int c[MAX][MAX], n, src;
    printf("Enter the number of nodes: ");
    scanf("%d", &n);
    printf("Enter the cost matrix:\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            scanf("%d", &c[i][j]);
        }
    }

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

    dijkstra(c, n, src);

    return 0;
}
```

Output:

```
Enter the number of nodes: 5
Enter the cost matrix:
0 1 3 5 4
1 0 2 6 0
3 2 0 4 1
5 6 4 0 2
4 0 1 2 0
Enter the source node: 1
Shortest distances are:
From 1 to 0: 1
From 1 to 1: 0
From 1 to 2: 1
From 1 to 3: 2
From 1 to 4: 0
```

Kruskal's Algorithm

```
#include <stdio.h>

#define INF 9999
#define MAX 10

int find(int parent[], int i) {
   while (parent[i] != 0) {
      i = parent[i];
   }
   return i;
}

void unionSets(int parent[], int u, int v) {
   parent[v] = u;
}

void kruskal(int c[MAX][MAX], int n) {
   int parent[MAX], ne = 0;
   int mincost = 0;

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

   printf("Edges in the minimum spanning tree are:\n");

   while (ne < n - 1) {
      int min = INF;
      int a = -1, b = -1, u = -1, v = -1;

      for (int i = 0; i < n; i++) {
         for (int j = 0; j < n; j++) {
            if (c[i][j] < min) {
               min = c[i][j];
               a = u = i;
               b = v = j;
            }
         }
      }
```

```
        u = find(parent, u);
        v = find(parent, v);

        if (u != v) {
            printf("%d - %d: %d\n", a, b, min);
            unionSets(parent, u, v);
            ne++;
            mincost += min;
        }

        c[a][b] = c[b][a] = INF;
    }

    printf("Minimum cost = %d\n", mincost);
}

int main() {
    int c[MAX][MAX], n;

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

    printf("Enter the cost matrix:\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            scanf("%d", &c[i][j]);
            if (c[i][j] == 0) {
                c[i][j] = INF;
            }
        }
    }
    kruskal(c, n);
    return 0;
}
```

Output:

```
Enter the number of nodes: 5
Enter the cost matrix:
0 5 7 2 9
5 0 6 8 4
7 6 0 3 9
2 8 3 0 4
9 4 9 4 0
Edges in the minimum spanning tree are:
0 - 3: 2
2 - 3: 3
1 - 4: 4
3 - 4: 4
Minimum cost = 13
```

Greedy knapsack Problem

```
#include <stdio.h>
void main() {
    int n;
    float m;
    printf("Enter the capacity\n");
    scanf("%f", &m);

    printf("Enter the number of objects\n");
    scanf("%d", &n);
    printf("Enter the elements of Profit/ Weight of %d objects\n", n);
   float w[n], p[n], x[n];
   float ratio[n];
   for (int i = 0; i < n; i++) {
      scanf("%f %f", &p[i], &w[i]);
      x[i] = 0;
      ratio[i] = p[i] / w[i];
   }

   for (int i = 0; i < n - 1; i++) {
      for (int j = 0; j < n - i - 1; j++) {
         if (ratio[j] < ratio[j + 1]) {
```

```
            //Swap profits float
            tp = p[j + 1]; p[j + 1]
            = p[j]; p[j] = tp;
            //Swap weights float
            tw = w[j + 1]; w[j + 1]
            = w[j]; w[j] = tw;
            //      Swap ratios
            float tr = ratio[j + 1];
            ratio[j + 1] = ratio[j];
            ratio[j] = tr;
          }
        }
      }
      float rc = m;
      float mp = 0;
      for (int i = 0; i < n; i++) {
// If weight is less than remaining capacity
        if (w[i] <= rc) {
          //make it visited
          x[i] = 1;
          //Subtract weight from remaining capacity
          rc -= w[i];
          //Add to total Profit
          mp += p[i];
        }
        // If weight is greater than capacity
        else {
          //Take portion of remaining capacity
          x[i] = rc / w[i];
          //add to profit
          mp += x[i] * p[i];
          break; // No more capacity left
        }
      }
```

```
    printf("The Selected objects are:\n");
    for (int i = 0; i < n; i++) {
        if (x[i]) {
            printf("Object %d (fraction: %.2f)\n", i + 1, x[i]);
        }
    }
    printf("The Maximum Profit is: %.2f\n", mp);

}
```

Output:

N Queens Problem

```
#include <stdio.h>
#include <stdbool.h>
#define N 8 // You can change N to any number to solve for different board size

void printSolution(int board[N][N]) {

    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            printf("%2d ", board[i][j]);
        }
```

```
Enter the capacity
40
Enter the number of objects
3
Enter the elements of Profit/ Weight of 3 objects
30 20
40 25
35 10
The Selected objects are:
Object 1 (fraction: 1.00)
Object 2 (fraction: 1.00)
Object 3 (fraction: 0.25)
The Maximum Profit is: 82.50

Process returned 29 (0x1D)   execution time : 13.050 s
Press any key to continue.
```

```
        printf("\n");
    }
}
bool isSafe(int board[N][N], int row, int col) {
    int i, j;
    //Check this row on the left side
    for (i = 0; i < col; i++)
        if (board[row][i])
            return false;
    //     Check upper diagonal on the left side
    for (i = row, j = col; i >= 0 && j >= 0; i--, j--)
        if (board[i][j])
            return false;
    // Check lower diagonal on the left side
    for (i = row, j = col; j >= 0 && i < N; i++, j--)
        if (board[i][j])
            return false;
    return true;
}
bool solveNQUtil(int board[N][N], int col) {
    //If all queens are placed
    if (col >= N)
        return true;

    //Consider this column and try placing this queen in all rows one by one
    for (int i = 0; i < N; i++) {
        //Check if the queen can be placed on board[i][col]
        if (isSafe(board, i, col)) {
            //Place this queen in board[i][col]
            board[i][col] = 1;
            //Recur to place the rest of the queens
            if (solveNQUtil(board, col + 1))
                return true;
```

```
            //If placing queen in board[i][col] doesn't lead to a solution
            //then backtrack
            board[i][col] = 0; // Remove queen from board[i][col]
        }
    }
    //If the queen cannot be placed in any row in this column, return

    false return false;

bool solveNQ() {
    int board[N][N] = {0};
    if (!solveNQUtil(board, 0)) {
        printf("Solution does not exist");
        return false;
    }
    printSolution(board);
    return true;
}
int main() {
    solveNQ();
    return 0;
}
```

Output: