VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

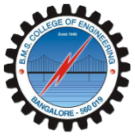
LAB REPORT
on

Analysis and Design of Algorithms

Submitted by

Pranav Y (1BM22CS204)

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 **Pranav Y (1BM22CS204),** 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.

Name of the Lab-In charge:	Dr. Jyothi S Nayak
Designation	Professor and Head
Department of CSE	Department of CSE
BMSCE, Bengaluru	BMSCE, Bengaluru

Index Sheet

	using Kruskal's algorithm. □ From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm. □ Implement Fractional Knapsack using Greedy technique.	
11	Implement "N-Queens Problem" using Backtracking.	45

Course Outcome

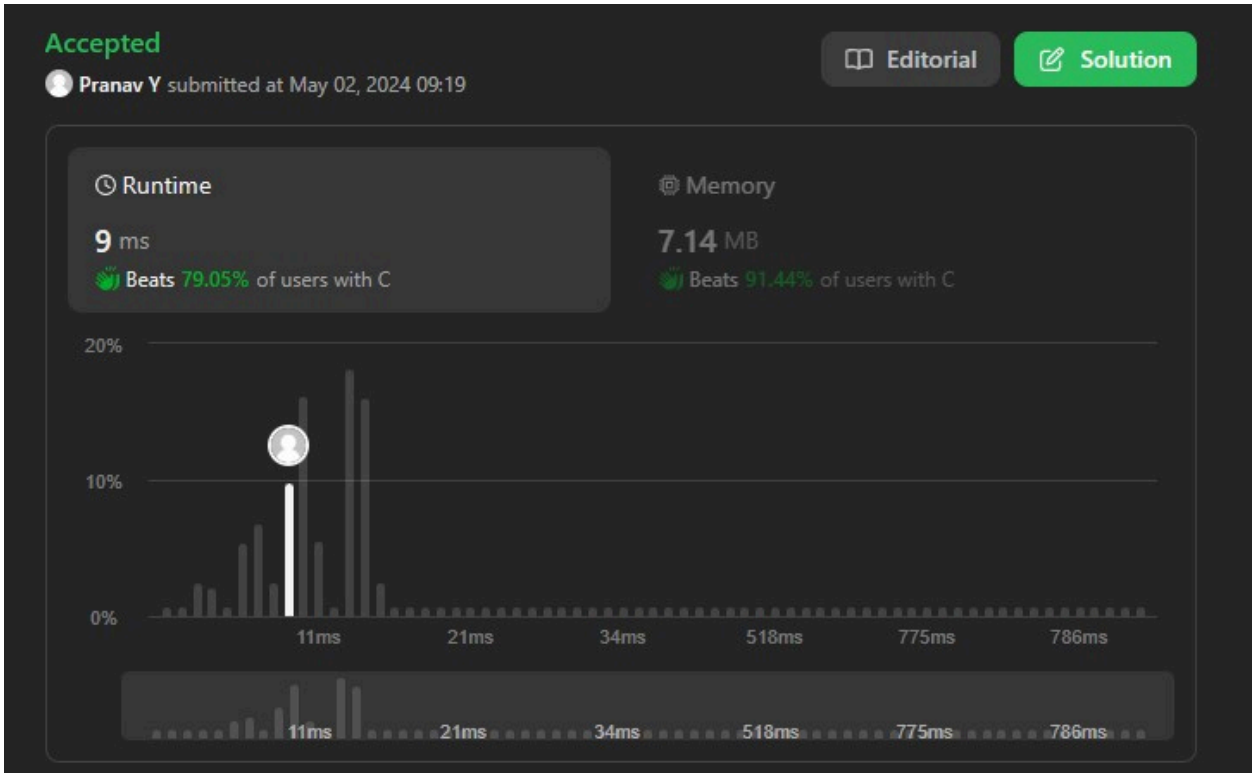
CO1	Analyze time complexity of Recursive and Non-recursive algorithms using asymptotic notations.
CO2	Apply various design techniques for the given problem.
CO3	Apply the knowledge of complexity classes P, NP, and NP-Complete and prove certain problems are NP-Complete
CO4	Design efficient algorithms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

Lab 1

LeetCode: Removing Duplicates from sorted array

```c
int removeDuplicates(int* nums, int numsSize) {
    int k = 1;
    for(int i = 1; i < numsSize; i++){
        if (nums[i] != nums[k-1]){
            nums[k] = nums[i];
            k++;
        }
    }
    return k;
}
```

Output

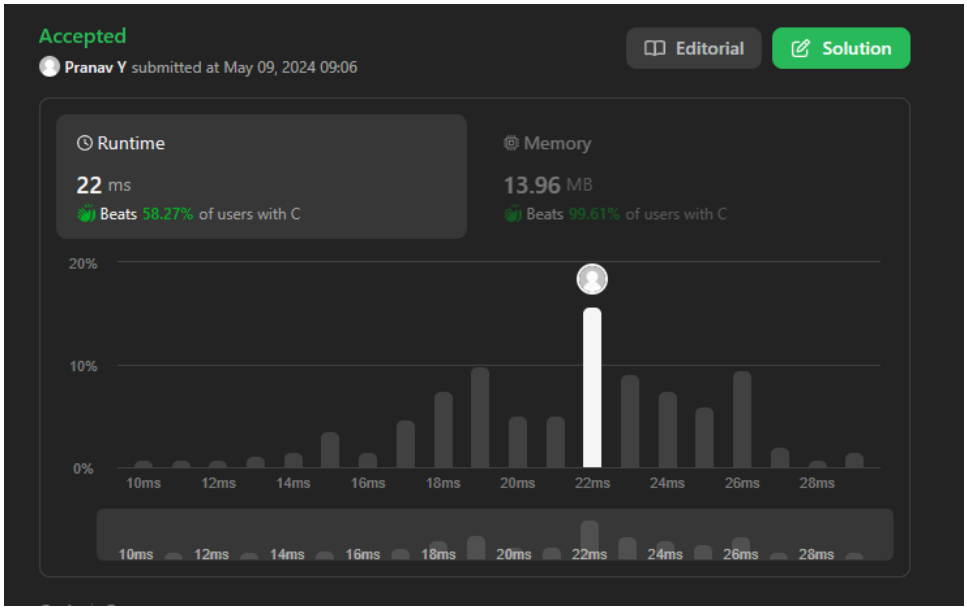


Lab 2

LeetCode: Kth Largest Sum in a Binary Tree

```c
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     struct TreeNode *left;
 *     struct TreeNode *right;
 * };
 */
struct TreeNode* mergeTrees(struct TreeNode* root1, struct TreeNode* root2)
{
    if (root1 == NULL && root2 == NULL) return NULL;
    if (root1 == NULL) return root2;
    else if (root2 == NULL) return root1;
    root1->val += root2->val;
    root1->left = mergeTrees(root1->left, root2->left);
    root1->right = mergeTrees(root1->right, root2->right);
    return root1;

}
```

Output



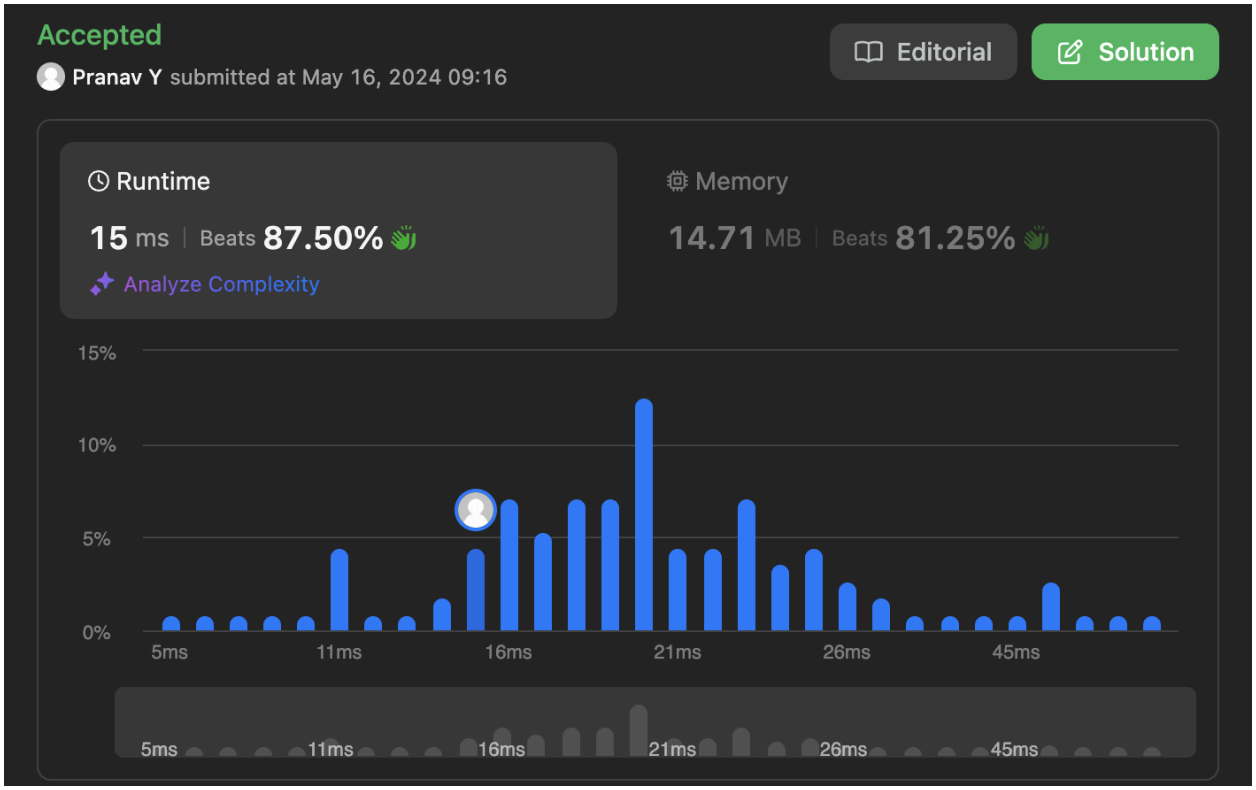
Lab 3

LeetCode: Increasing Order Search Tree

```c
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     struct TreeNode *left;
 *     struct TreeNode *right;
 * };
 */

bool exists(struct TreeNode* root, int num){
    if (!root) return false;
    int temp = root->val;
    if (temp == num){
        return true;
    } else if (temp > num){
        return exists(root->left, num);
    } else if (temp < num){
        return exists(root->right, num);
    }
    return false;
}

void findComplement(struct TreeNode* head, struct TreeNode* root,  bool
*res, int k){
```

```
    if (!root) return;
    if (*res) return;
    int temp = k - root->val;
    if (temp > root->val){
        if (exists(head, temp)){
            *res = true;
            return;
        }
    } else if (temp < root->val){
        if (exists(head, temp)){
            *res = true;
            return;
        }
    }
    findComplement(head, root->left, res, k);
    findComplement(head, root->right, res, k);
}
bool findTarget(struct TreeNode* root, int k) {
    if (!root) return false;
    bool res = false;
    findComplement(root, root, &res, k);
    return res;
}
```

Output



Lab 4

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

DFS

```c
#include <stdio.h>

int s[100];
int res[100];
int m=0;

void dfs(int u, int n, int a[n][n]);
void dfs_tp(int n, int a[n][n]);

void dfs_tp(int n, int a[n][n]){
```

```
    for(int i=0;i<n;i++)
        s[i]=0;

    for(int u=0; u<n; u++){
        if(s[u]==0)
            dfs(u,n,a);
    }
}

void dfs(int u, int n, int a[n][n]){

    s[u]=1;
    res[m]=u;
    m++;

    for(int v=0; v<n; v++){
        if(a[u][v]==1 && s[v]==0)
            dfs(v,n,a);
    }
}

int main()
{
    int i,n;
    n=6;

    int a[6][6] = {

        {0, 0, 0, 0, 0, 0},  // Node 0
        {0, 0, 0, 1, 0, 0},  // Node 1
        {0, 0, 0, 1, 0, 0},  // Node 2
        {0, 0, 0, 0, 0, 0},  // Node 3
        {1, 1, 0, 0, 0, 0},  // Node 4
        {1, 0, 1, 0, 0, 0}   // Node 5

    };


    dfs_tp(n,a);

    printf("DFS Traversal order:\n");
    for(int i=n-1;i>0;i--)
```

```
    printf("%d\t",res[i]);

    return 0;
}
```

Output

```
DFS Traversal order:
5       4       2       3       1
```

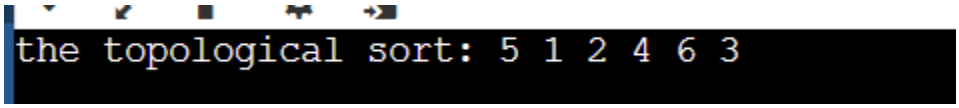
Source Removal

```
#include <stdio.h>
#define v 100
int top = -1;

void indegree(int a_matrix[v][v], int n, int in[v])
{
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            if (a_matrix[i][j])
            {
                in[j]++;
            }
        }
    }
}

void toposort(int a_matrix[v][v], int n)
{
    int in[v] = {0};
    int topo[v];
    int k = 0;
    int s[v] = {0};

    indegree(a_matrix, n, in);

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

```
    {
        if (in[i] == 0)
        {
            top++;
            s[top] = i;
        }
    }

    while (top != -1)
    {
        int vertex = s[top];
        top--;
        topo[k++] = vertex;

        for (int i = 0; i < n; i++)
        {
            if (a_matrix[vertex][i])
            {
                in[i]--;
                if (in[i] == 0)
                {
                    top++;
                    s[top] = i;
                }
            }
        }
    }

    if (k != n)
    {
        printf("cycle exists");
    }
    else
    {
        printf("the topological sort: ");
        for (int i = 0; i < n; i++)
        {
            printf("%d ", topo[i] + 1);
        }
    }
}

int main()
```

```
{
    int a_matrix[v][v] = {
        {0, 1, 0, 0, 0, 1},
        {0, 0, 1, 1, 0, 0},
        {0, 0, 0, 0, 0, 0},
        {0, 0, 0, 0, 0, 1},
        {1, 0, 0, 0, 0, 0},
        {0, 0, 0, 0, 0, 0}
    };
    int n = 6;

    toposort(a_matrix, n);
    return 0;
}
```

Output



Lab 5

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

Merge Sort

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

void merge(int a[],int l,int m,int high)
{
    int i,h;h=l;i=0;
    int j=m+1;int b[high-l+1];
    while(h<=m && j<=high)
```

```
    {
        if(a[h]<=a[j])
        {
            b[i]=a[h];
            h++;
        }
        else
        {
            b[i]=a[j];
            j++;
        }i++;
    }
    if(h<=m)
        for(int k=h;k<m;k++)
    {
        b[i++]=a[k];
    }
    if(j<=high)
        for(int k=j;k<high;k++)
    {
        b[i++]=a[k];
    }
    for(int k=0,j=l;k<=high;k++,j++)
    {
        a[j]=b[k];
    }
}

void mergesort(int a[],int l,int h)
{
    int m;
    if(l<h)
    {
        m=(l+h)/2;
        mergesort(a,l,m);
        mergesort(a,m+1,h);
        merge(a,l,m,h);
    }
}

int main()
{
    clock_t start,end;
```

```
    int n;
    printf("Enter the number of array elements:");
    scanf("%d",&n);
    int arr[n];
    srand(time(NULL));
    for (int i=0;i<n;i++)
        arr[i]=rand();
    start=clock();
    mergesort(arr,0,n-1);
    end=clock();
    printf("\nTime taken:%f",((double)(end - start)) / CLOCKS_PER_SEC);
    return 0;
}
```

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 15000 to 100000
3: To exit
Enter your choice: 1

Enter the number of elements: 6

Enter array elements: 3 5 1 2 6 4

Sorted array is: 1      2       3       4       5       6
```

Selection Sort

```
#include<stdio.h>
#include<time.h>
#include<stdlib.h> /* To recognise exit function when compiling with gcc*/

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

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

   while(1){
```

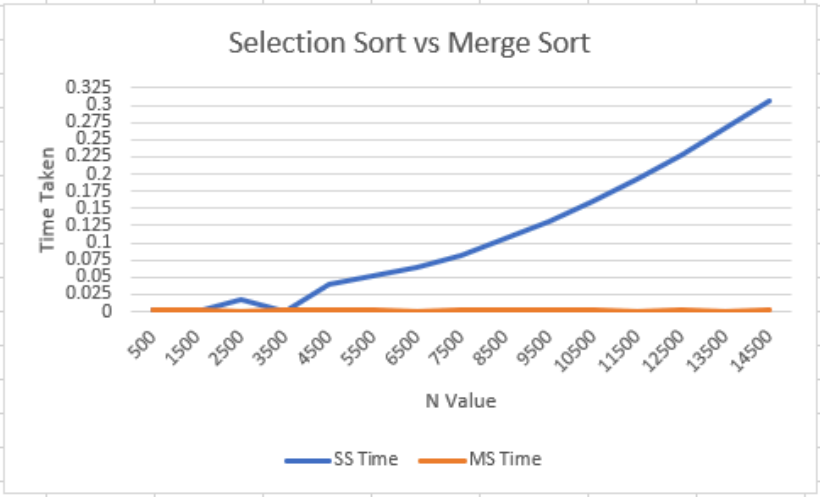
```
    printf("\n1:For manual entry of N value and array elements");
    printf("\n2:To display time taken for sorting number of elements N in
the range 500 to 14500");
    printf("\n3:To exit");
    printf("\nEnter your choice:");
    scanf("%d", &ch);
     switch(ch){
       case 1:
           printf("\nEnter 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);
          //Dummy loop to create delay
              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

```
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: 4 1 2 6 3 5

Sorted array is: 1       2       3       4       5       6
```

N Value	SS Time	MS Time
500	0.003295	0.002698
1500	0.010222	0.002907
2500	0.01843	0.003055
3500	0.029586	0.003391
4500	0.040886	0.003037
5500	0.051843	0.002826
6500	0.063836	0.003703
7500	0.082561	0.003794
8500	0.105688	0.003021
9500	0.131809	0.003072
10500	0.161104	0.003144
11500	0.193703	0.003546
12500	0.22794	0.003321
13500	0.265653	0.003507
14500	0.305674	0.003624



Lab 6

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

```c
#include<stdio.h>
#include<time.h>
#include<stdlib.h> /* To recognise exit function when compiling with gcc*/
void swap(int* a, int* b);
int partition(int arr[],int low, int high);
void quicksort(int arr[], int low, int high);
void main()
{
   int i,j,ch, temp;
   clock_t start,end;
   int a[110000], n;
```

```
   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 7500 to 25000");
 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();
            quicksort(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=7500;
             while(n<=25500) {
             for(i=0;i<n;i++)
              {
                //a[i]=random(1000);
                a[i]=n-i;
              }
             start=clock();
             quicksort(a,0,n-1);
           //Dummy loop to create delay
         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;
```

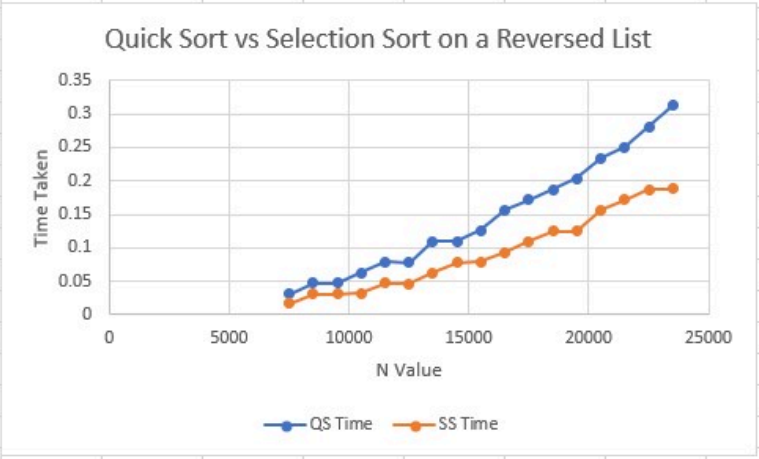
```
            }
          break;
   case 3: exit(0);
   }
   getchar();
    }
}




void swap(int* p1, int* p2)
{
    int temp;
    temp = *p1;
    *p1 = *p2;
    *p2 = temp;
}

int partition(int arr[], int low, int high)
{
    // choose the pivot
    int pivot = arr[high];

    // Index of smaller element and Indicate
    // the right position of pivot found so far
    int i = (low - 1);

    for (int j = low; j <= high; j++) {
        // If current element is smaller than the pivot
        if (arr[j] < pivot) {
            // Increment index of smaller element
            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 pivot = partition(arr, low, high);

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

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 7500 to 25000
3:To exit
Enter your choice:1

Enter the number of elements: 6

Enter array elements: 1 6 3 5 2 4

Sorted array is: 1      2       3       4       5       6
 Time taken to sort 6 numbers is 0.000001 Secs
```

N Value	QS Time	SS Time
7500	0.031	0.016
8500	0.047	0.031
9500	0.047	0.031
10500	0.062	0.032
11500	0.079	0.047
12500	0.078	0.046
13500	0.109	0.063
14500	0.109	0.078
15500	0.126	0.079
16500	0.156	0.093
17500	0.172	0.11
18500	0.187	0.125
19500	0.204	0.125
20500	0.234	0.156
21500	0.25	0.172
22500	0.281	0.187
23500	0.313	0.188



Lab 7

Implement Johnson Trotter algorithm to generate permutations.

```c
#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: ");
    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: 4
total permutations = 24
All possible permutations are:
 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
```

Substring matching or pattern matching of substring in text return the position of it.

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

int main(){
    int m, n;
    printf("Enter m and n: ");
    scanf("%d%d", &m, &n);
    char s1[m], s2[n];
    fflush(stdin);
    printf("Enter string 1: ");
    scanf("%s", s1);
```

```
    printf("Enter string 2: ");
    scanf("%s", s2);
    // printf("%s %s", s1, s2);
    int i = 0, j = 0;
    while (i < n - m + 1){
        if (s1[0] == s2[i]){
            bool found = true;
            while (j < m){
                if (s1[j] != s2[i+j]) found = false;
                j++;
            }
            if (found) printf("Found at %d", i);
        }
        i++;
    }
    return 0;
}
```

Output

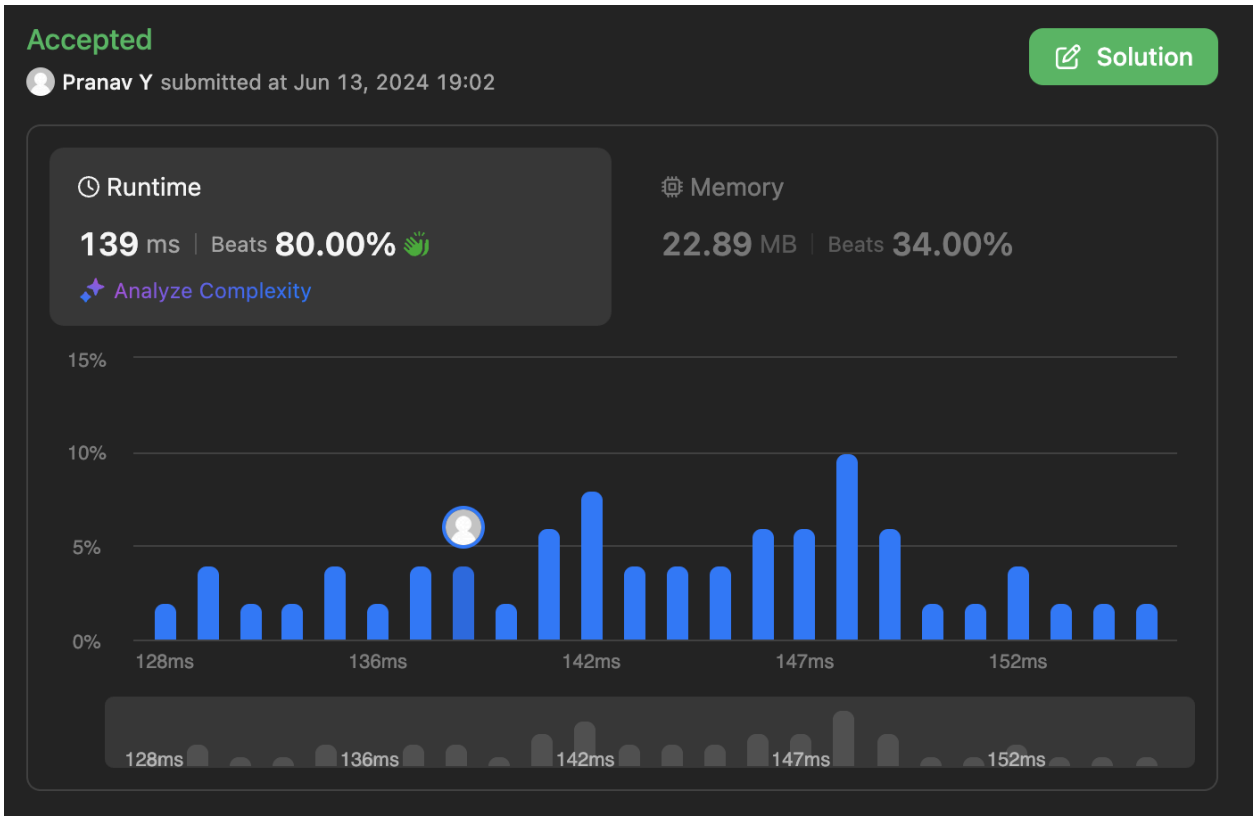
```
Enter m and n: 3 7
Enter string 1: hin
Enter string 2: abhinav
Found at 2
```

LeetCode:

Find the 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

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

```c
#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, 3, INF, 1 },
                        { INF, 0, 6, 2 },
                        { 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      3      9      1
    INF      0      6      2
    INF    INF      0      1
    INF    INF    INF      0
```

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

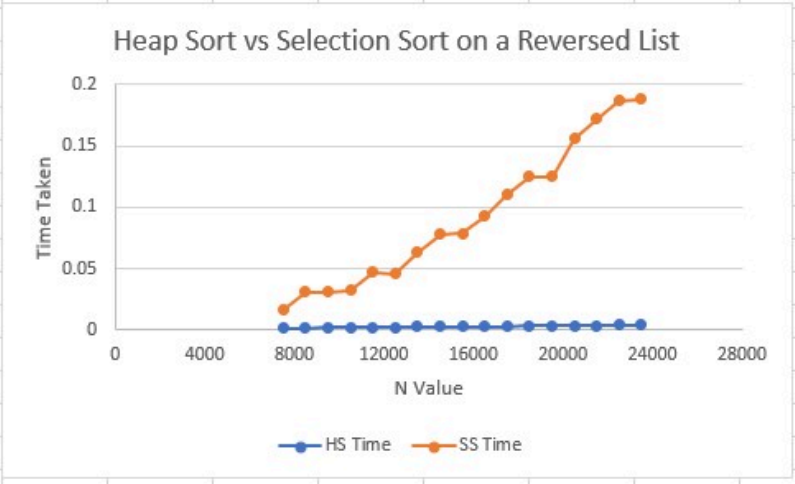
```c
#include <stdio.h>

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

    int temp = *a;
    *a = *b;
    *b = temp;
}

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

    int largest = i;

    int left = 2 * i + 1;

    int right = 2 * i + 2;

    if (left < N && arr[left] > arr[largest])

        largest = left;

    if (right < N && arr[right] > arr[largest])

        largest = right;
```

```
    if (largest != i) {

        swap(&arr[i], &arr[largest]);

        heapify(arr, N, largest);
    }
    printf("heapify:");
    display(arr,N);
}

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--) {

        swap(&arr[0], &arr[i]);
        heapify(arr, i, 0);
        printf("heapsort:");
        display(arr,N);
    }

}

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

int main()
{
    int arr[] = { 14,8,3,9,44,32};
    int N = sizeof(arr) / sizeof(arr[0]);
    heapSort(arr, N);
    printf("Sorted array is\n");
    printArray(arr, N);
}
```

Output

```
Input array: 14 8 3 9 44 32
Sorted array is:3 8 9 14 32 44
```

N Value	HS Time	SS Time
7500	0.00161	0.016
8500	0.001656	0.031
9500	0.001809	0.031
10500	0.001969	0.032
11500	0.002117	0.047
12500	0.002291	0.046
13500	0.002479	0.063
14500	0.002607	0.078
15500	0.002781	0.079
16500	0.00293	0.093
17500	0.003081	0.11
18500	0.003267	0.125
19500	0.003449	0.125
20500	0.003658	0.156
21500	0.003729	0.172
22500	0.003957	0.187
23500	0.004099	0.188



Lab 9

Implement 0/1 Knapsack problem using dynamic programming.

```c
#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
```

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

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

#define MAX_VERTICES 5

int minKey(int n, 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;
        }
    }
```

```c
    return min_index;
}

void printMST(int n, int parent[], int cost[MAX_VERTICES][MAX_VERTICES]) {
    int sum = 0;

    printf("Edges in MST:\n");
    for (int i = 1; i < n; i++) {
        printf("%d - %d\n", parent[i], i);
        sum += cost[i][parent[i]];
    }

    printf("Cost of MST is: %d\n", sum);
}


void primMST(int n, int cost[MAX_VERTICES][MAX_VERTICES]) {
    int parent[MAX_VERTICES];
    int key[MAX_VERTICES];
    int mstSet[MAX_VERTICES];

    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(n, key, mstSet);
        mstSet[u] = 1;


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

            if (cost[u][v] && mstSet[v] == 0 && cost[u][v] < key[v]) {
                parent[v] = u;
                key[v] = cost[u][v];
            }
        }
```

```
    }

    printMST(n, parent, cost);
}

int main() {
    int n = MAX_VERTICES;
    int cost[MAX_VERTICES][MAX_VERTICES] = {
        {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}
    };

    printf("Cost adjacency matrix:\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            printf("%d ", cost[i][j]);
        }
        printf("\n");
    }

    primMST(n, cost);

    return 0;
}
```

Output

```
Cost adjacency matrix:
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
Edges in MST:
0 - 1
1 - 2
0 - 3
1 - 4
Cost of MST is: 16
```

Lab 10

1. **Find Minimum Cost Spanning Tree of a given undirected graph using Kruskal's algorithm.**
2. **From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm.**

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <limits.h>

#define MAX 100
#define INF 9999

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

int find(int parent[], int i) {
    if (parent[i] == i)
        return i;
```

```
    return find(parent, 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) {
    Edge* edgeA = (Edge*) a;
    Edge* edgeB = (Edge*) b;
    return edgeA->weight > edgeB->weight;
}

void KruskalMST(int graph[MAX][MAX], int numVertices) {
    int E = 0;
    Edge edges[MAX * MAX];
    for (int i = 0; i < numVertices; i++) {
        for (int j = i + 1; j < numVertices; j++) {
            if (graph[i][j] != 0 && graph[i][j] != INF) {
                edges[E].src = i;
                edges[E].dest = j;
                edges[E].weight = graph[i][j];
                E++;
            }
        }
    }

    qsort(edges, E, sizeof(edges[0]), compareEdges);

    int parent[numVertices];
    int rank[numVertices];
    for (int v = 0; v < numVertices; v++) {
        parent[v] = v;
```

```
        rank[v] = 0;
    }

    Edge result[numVertices];
    int e = 0, i = 0;

    while (e < numVertices - 1 && i < E) {
        Edge next_edge = edges[i++];
        int x = find(parent, next_edge.src);
        int y = find(parent, next_edge.dest);

        if (x != y) {
            result[e++] = next_edge;
            Union(parent, rank, x, y);
        }
    }

    printf("Edges in the MST:\n");
    for (i = 0; i < e; i++)
        printf("%d -- %d == %d\n", result[i].src, result[i].dest,
result[i].weight);
}

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

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

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

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

```
    dist[src] = 0;

    for (int count = 0; count < numVertices - 1; count++) {
        int u = minDistance(dist, sptSet, numVertices);
        sptSet[u] = true;

        for (int v = 0; v < numVertices; 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, numVertices);
}

int main() {
    int numVertices = 5;
    int graph[MAX][MAX] = {
        {0, 2, INF, 6, INF},
        {2, 0, 3, 8, 5},
        {INF, 3, 0, INF, 7},
        {6, 8, INF, 0, 9},
        {INF, 5, 7, 9, 0}
    };

    printf("Kruskal's MST:\n");
    KruskalMST(graph, numVertices);

    printf("\nDijkstra's Shortest Paths from vertex 0:\n");
    dijkstra(graph, 0, numVertices);

    return 0;
}
```

Output

```
Kruskal's MST:
Edges in the MST:
0 -- 1 == 2
1 -- 2 == 3
1 -- 4 == 5
0 -- 3 == 6

Dijkstra's Shortest Paths from vertex 0:
Vertex   Distance from Source
0                0
1                2
2                5
3                6
4                7
```

Implement Fractional Knapsack using Greedy technique.

```c
#include <stdio.h>

void knapsack(int n, int p[], int w[], int W) {
    int used[n];
    for (int i = 0; i < n; ++i) {
        used[i] = 0;
    }

    int cur_w = W;
    float tot_v = 0.0;
    int i, maxi;

    while (cur_w > 0) {
        maxi = -1;
        for (i = 0; i < n; ++i) {
```

```
            if ((used[i] == 0) && ((maxi == -1) || ((float)p[i]/w[i] >
(float)p[maxi]/w[maxi]))) {
                maxi = i;
            }
        }

        if (maxi == -1) break; // no more items to select

        used[maxi] = 1;

        if (w[maxi] <= cur_w) {
            cur_w -= w[maxi];
            tot_v += p[maxi];
            printf("Added object %d (weight: %d, profit: %d) completely in
the bag. Space left: %d.\n", maxi + 1, w[maxi], p[maxi], cur_w);
        } else {
            int taken = cur_w;
            cur_w = 0;
            tot_v += (float)taken/w[maxi] * p[maxi];
            printf("Added %d%% (weight: %d, profit: %d) of object %d in the
bag.\n", (int)((float)taken/w[maxi] * 100), w[maxi], p[maxi], maxi + 1);
        }
    }

    printf("Filled the bag with objects worth %.2f.\n", tot_v);
}

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

    int p[n], w[n];

    printf("Enter the profits of the objects: ");
    for (int i = 0; i < n; i++) {
        scanf("%d", &p[i]);
    }

    printf("Enter the weights of the objects: ");
    for (int i = 0; i < n; i++) {
        scanf("%d", &w[i]);
    }
```

```
    printf("Enter the maximum weight of the bag: ");
    scanf("%d", &W);

    knapsack(n, p, w, W);

    return 0;
}
```

Output

```
Enter the number of objects: 4
Enter the profits of the objects: 20 30 66 40
Enter the weights of the objects: 10 20 30 40
Enter the maximum weight of the bag: 50
Added object 3 (weight: 30, profit: 66) completely in the bag. Space left: 20.
Added object 1 (weight: 10, profit: 20) completely in the bag. Space left: 10.
Added 50% (weight: 20, profit: 30) of object 2 in the bag.
Filled the bag with objects worth 101.00.
```

Lab 11

Implement "N-Queens Problem" using Backtracking.

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

bool place(int[], int);
void printSolution(int[], int);
void nQueens(int);

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

```
}

void nQueens(int n) {
    int x[10] = {0};  // Initialize the array to zero
    int count = 0;
    int k = 1;

    while (k != 0) {
        x[k] = x[k] + 1;

        while (x[k] <= n && !place(x, k)) {
            x[k] = x[k] + 1;
        }

        if (x[k] <= n) {
            if (k == n) {
                printSolution(x, n);
                printf("Solution found\n");
                count++;
            } else {
                k++;
                x[k] = 0;
            }
        } else {
            k--;
        }
    }

    printf("Total solutions: %d\n", count);
}

bool place(int x[], int k) {
    for (int i = 1; i < k; i++) {
        if ((x[i] == x[k]) ||
            (i - x[i] == k - x[k]) ||
            (i + x[i] == k + x[k])) {
            return false;
        }
    }
    return true;
}

void printSolution(int x[], int n) {
```

```
    for (int i = 1; i <= n; i++) {
        printf("%d ", x[i]);
    }
    printf("\n");
}
```

Output

```
Enter the number of queens: 4
2 4 1 3
Solution found
3 1 4 2
Solution found
Total solutions: 2
```