1BM22CS280
4E

INDEX

Name Siri. Sathish Sub. ADA LAB

Std.: Div. Roll No.

Telephone No. E-mail ID.

Blood Group. Birth Day.

Sr.No.	Title	Page No.	Sign./Remarks
01	Linear Search	10	Sneha 3/5/24
02	Binary Search		
03	Bubble Sort		
04	Selection Sort		
02	Topological Sort using DFS, Source removal, Tower of Hanoi, Lomuto Partition.	10	Sneha 7/6/24
03	Merge sort and quick sort		
04	Warshall, Floyd, Johson Torter, Knapsack	10	Sneha 21/6/24
05	Heapsort, horspool.		
06	Prims, Krustkal, Dijkstras, Knapsack.	10	Sneha 5/7/24
07	N-Queens	10	Sneha 19/7/24

PAGE NO :
DATE :

Linear Search:

```
#include <stdio.h>
int arr[50];
int data, size;
void linear_search();

int main(){
    linear_search();
    return 0;
}

void linear_search(){
    printf("enter the size of the array:");
    scanf("%d", &size);
    printf("enter the array elements:\n");
    for(int i=0; i<size; i++){
        scanf("%d", &arr[i]);
    }
printf("enter the element you want to
search:");
    scanf("%d", &data);
    int found=0;
    for(int i=0; i<size; i++){
        if(arr[i]==data){
            printf("element %d found at %d
location", data, i+1);
            found = 1;
            break;
    }
}
    if(!found){
        printf("element not found!");
    }
}
```

O/P:

```
Enter the size of the array: 5
Enter the array elements:
1 2 3 4 5
Enter the element you want to search: 4
Element 4 found at 4 location.
```

Binary Search:

```
#include <stdio.h>
int arr[50];
int data, size;
void binary_search();

int main(){
  binary_search();
  return 0;
}

void binary_search(){
  printf("Enter the size of the array:");
  scanf("%d", &size);
  printf("Enter the array elements:\n");
  for(int i=0; i<size; i++){
    scanf("%d", &arr[i]);
  }
  printf("Enter the element you want to
  search");
  scanf("%d", &data);
    int l=0;
    int r = size-1; int mid;
  while(l<=r){
      mid = (l+r)/2;
```

```
if (arr[mid] == data) {
    printf("Element %d found at %d location",
data, mid+1);
        break;
}
  else if (data > arr[mid]){
        l = mid+1;
  }
    else {
        r = mid-1;
  }
}
}
```

Bubble sort

O/P

Enter the size of the array: 5
Enter the array elements:
1 2 3 4 5
Enter the element you want to search: 5
Element 5 found at 5 location.

Bubble Sort:

```
#include <stdio.h>
int arr[50];
int data, size;
void bubble_sort();

int main(){
    bubble_sort();
    return 0;
}
```

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

    for(int i=0; i<size-1; i++){
    for(int j=0; j<size-i-1; j++){
        if(arr[j] > arr[j+1]){
            int temp = arr[j];
            arr[j] = arr[j+1];
            arr[j+1] = temp;
        }
    }
    }

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

O/P:

Enter the size of the array: 5
Enter the array elements:
6 7 3 2 5
Sorted array in ascending order:
2 3 5 6 7

PAGE NO :
DATE :

Selection Sort:

```
#include <stdio.h>
int arr[50];
int size;
void selection_sort();

int main(){
    selection_sort();
    return 0;
}

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

for(int i=0; i<size-1; i++){
    int min_index = i;
for(int j = i+1; j<size; j++){
   if (arr[j] < arr[min_index]){
}
}
   int temp = arr[i];
   arr[i] = arr[min_index];
   arr[min_index] = temp;
}
   printf("sorted array in ascending
order: \n");
```

float a;
clock_t time_req;
time_req = clock();

time_req = clock() - time_req
if(Time is),
(float) time_req / CLOCKS_PER_SEC;

min_index = j;

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

O/P:

Enter the size of the array: 5
Enter the array elements:
3 4 8 9 2
Sorted array in ascending order:
2 3 4 8 9.

31/5/24

PAGE NO:
DATE:

Topological Sort Using DFS:

O/P:

Enter the no: of nodes: 5
Enter the adjacency matrix:
0 0 1 1 0
1 0 0 1 0
0 0 0 0 1
0 0 1 0 1
0 0 0 0 0

The topologically sorted array is: 1 0 3 2 4

Topological Sort Using Source Removal:

Enter the no of nodes: 5
Enter the adjacency matrix:
0 0 1 1 0
1 0 0 1 0
0 0 0 0 1
0 0 1 0 1
0 0 0 0 0

The topologically sorted array is: 1 0 3 2 4

GCD:

O/P:

Enter the value of a and b:
6 10
The gcd of the numbers is 2

PAGE NO :
DATE :

Towers of Hanoi:

O/P:

3 disks:
Move d1 from tA to tC
Move d2 from tA to tB
Move d1 from tC to tB.
Move d3 from tA to tC.
Move d1 from tB to tA.
Move d2 from tB to tC.
Move d1 from tA to tc.

Lomuto Partition:

Original Array:
10 7 8 9 1 5
Array after partitioning:
7 8 9 1 5 10.
Partition index: 5

7/2/24

PAGE NO:
DATE:

Merge Sort:

```
#include <stdio.h>
#define MAX_SIZE 50

int a[MAX_SIZE], c[MAX SIZE];
int size = 5;

void simple_merge (int[], int, int, int);
void mergesort (int[], int, int);

int main(){
    printf("Enter the array elements in an
unsorted manner: \n");
    for (int i=0; i<size; i++){
        scanf("%d", &a[i]);
    }

printf("Sorted array is: \n");
  mergesort (a, 0, size-1);
  for (int i=0; i<size; i++){
      printf("%d", a[i]);
  }
  printf("\n");
  return 0;
}

void simple_merge (int a[], int loco, int mid, int high)
{
  int i = loco;
  int j = mid+1;
  int k = 0;
```

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

while (i<=mid){
    c[k++] = a[i++];
}
while (j<=high){
    c[k++] = a[j++];
}
for (i=low, k=b; i<=high; i++, k++)
    a[i] = c[k];
}
}

void mergesort (int a[], int low, int high){
    if (low < high){
        int mid = (low+high)/2;
        mergesort (a, low, mid);
        mergesort (a, mid+1, high);
        Simple_merge (a, low, mid, high);
    }
}
```

Output:

Enter the array elements in an unsorted manner:

5 6 8 3 2

PAGE NO:
DATE:

Sorted array is:
2 3 5 6 8

Quick Sort:

```
#include <stdio.h>
#define MAX_SIZE 50

int a[MAX_SIZE];
int size = 5;

int partition (int[], int, int);
void quicksort (int[], int, int);

int main(){
    printf("Enter the array elements in an unsorted manner: \n");
    for(int i=0; i<size; i++){
        scanf("%d", &a[i]);
    }
    printf("\n");
    return 0;
}

int partition (int a[], int low, int high){
    int pivot = a[(low+high)/2];
    int i = low-1;
    int j = high+1;

    while(1){
        do{
            i++;
        } while(a[i] < pivot);
```

```
do {
    j--;
} while (a[j] > pivot);

if (i >= j) {
    return j;
}
    int temp = a[i];
    a[i] = a[j];
    a[j] = temp;
}
}

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

O/P:

Enter the array elements in an unsorted manner:

7 8 7 2 4

Sorted array is:

2 4 7 8 7

13/6/24 Warshall:

```
#include<stdio.h>
void warshall (int [0] [50], int);

int main() {
    int n;
    int a[50][50];
    printf("Enter the number of vertices
for the adjacency matrix: \n");
    scanf("%d", &n);
    printf("Enter the adjacency matrix
values: \n");
    for(int i=0; i<n; i++){
        for(int j=0; j<n; j++){
            scanf("%d", &a[i][j]);
        }
    }
  printf("The path matrix is: \n");
   warshall(a, n);
   return 0;
}
void warshall(int a[][50], int n){
        int p[50][50];
        for(int i=0; i<=n-1; i++){
            for(int j=0; j<=n-1; j++){
                p[i][j] = a[i][j];
            }
        }
```

```
for (int k = 0; k<=n-1; k++){
    for (int i = 0; i<=n-1; i++){
        for (int j=0; j<=n-1; j++){
            if (p[i][j]==0 && p[i][k]!=1 &&
p[k][j] == 1){
}
}
}
}
for (int i = 0; i<=n-1; i++){
        for (int j=0; j<=n-1; j++){
            printf("%d \t", p[i][j]);
        }
printf("\n");
}
}
```

O/P:

Enter the no: of vertices for the adjacency matrix: 4

Enter the adjacency matrix values:

0 1 0 0
0 0 0 1
0 0 0 0
1 0 1 0

The path matrix is:

1 1 1 1
1 1 1 1
0 0 0 0
1 1 1 1

Floyd's:

```
#include<stdio.h>
void floyds(int [][50], int);
int min(int, int);

int main(){
    int n;
    int a[50][50];
    printf("Enter the no: of vertices for
cost adjaceney matrix: \n");
    scanf("%d", &n);
    printf("Enter the cost adjaceney
matrix values: \n");
    for(int i=0; i<n; i++){
        for(int j=0; j<n; j++){
            scanf("%d", &a[i][j]);
    }
}
    printf("The distance matrix is \n");
    floyds(a,n);
    return 0;
}

void floyds(int a[][50), int n){
    int d[50][50);

    for(int i=0; i<=n-1; i++){
        for(int j=0; j<=n-1; j++){
            d[i][j] = a[i][j];
        }
    }
```

```
for (int K = 0; K <= n-1; K++){
    for (int i = 0; i <= n-1; i++){
        for (int j = 0; j <= n-1; j++){
            d[i][j] = min(d[i][j], d[i][k] + d[k]
        }
    }
}

for (int i=0; i<=n-1; i++){
    for (int j=0; j<=n-1; j++){
        printf("%d\t", d[i][j]);
    }
    printf("\n");
}

int min(int a, int b){
    if (a <= b){
        return a;
    }
    else{
        return b;
    }
}
```

O/P:

Enter the no: of vertices for the cost adjacency matrix:

4

Enter the cost adjacency matrix values:

0	∞	3	∞
2	0	∞	∞
∞	6	0	1
7	∞	∞	0

The distance matrix is:

0	9	8	4
2	0	5	6
8	6	0	1
7	15	10	0

Johnson Trotter:

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

#define MAXN 20

int p[MAXN];
int pi[MAXN];
int dir[MAXN];

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

void printpermutation(int n){
    for(int i=0; i<n; i++){
        printf("%d", p[i]);
    }
        printf("\n");
}
```

```
int mobile, mobileIndex;
bool found;
while (1){
    mobile = -1;
    found = false;
    for (int i = 0; i<n; i++){
    int next = i+dir[i];
    if (next >= 0 && next<n && p[i] > p[next])
{
if(p[i] > mobile ) {
    mobile = p[i];
    mobileIndex = i;
    found = true;}}}

if (! found)
    break;

int next = mobileIndex + dir[mobile
Index];
swap (&p[mobileIndex], &p[next]);
swap (&pi[p[mobileIndex]], &pi[p
[next]]);
swap (&dis[mobileIndex], &dis[next]);
printPermutation(n);

for (int i =0; i<n; i++)
  {
    if (p[i] > mobile) {
      dir[i] *= -1;
    }
  }
}
}
```

```
int main() {
    int n;
    printf("Enter the value of n:");
    scanf("%d", &n);
    if (n>maxN || n<1) {
        printf("Invalid input\n");
        return 1;
    }
```

OIP:

```
Enter the value of n: 3
123
132
312
321
231
213
```

Knapsack Problem:

```
#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][w
};
}
return k[n][w];
}
int main() {
    int n;
printf("Enter the no of items: \n");
scanf("%d", &n);
    int wt[100];
    int val[100];
    int w = 50;
printf("Enter the weight of each item
and its corresponding value: \n");
    for(int i=0; i<n; i++){
      scanf("%d %d", &wt[i], &val[i]);
}
printf("Max value that can be obtained
  is %d \n", knapsack(50, wt, val, n));
return 0;
}
```

O/P:

Enter no. of items:

3

Enter the weight of each item and its corresponding value:

10

60

20

100

30

120 Max value that is obtained is 220.

21/6/24

Horspool

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

#define MAX
void ShiftTable (char p[], int s[]){
    int m = strlen (p);

  for (int i=0 ; i<127; I++) {
      s[i] = m;
  }

    for (int j = 0; j<m-1 ; j++){
     s [(unsigned char) p[j]] = m-1-j;
    }
}

int horspool (char p[] , char t[]){
      int s [127];
      ShiftTable (p,s);
      int m = strlen(p);
      int n = strlen (t);
      int i = m-1;

    while (i<n){
        int k=0;
        while (k<m && p[m-1-k]==t
          k++;
      }
      if (k==m){
        return i-m+1;
      }
```

```
        else{
            i+=s[(unsigned char)t[i]];
        }
    }
    return -1;
}
int main(){
    char p[50]
    char t[50];
    pf("Enter the pattern string:\n");
    Scanf("%s", p);

    printf("Enter the text string:\n");
    scanf("%s,t);
    int position = Horspool(p,t);
    if(position == -1){
        pf("Pattern not found");
    }
    else{
        pf("Pattern found at position
%d\n", position);
}
    return 0;
}
```

O/P: Enter the text string:
Jim saw me at a barber shop
Enter the pattern string:
barber
The position where the pattern starts is
16

PAGE NO:
DATE:

Heapsort:

```
#include <stdio.h>
int a[50];
 int n;

void heapify(int[], int);
int main(){
    printf("Enter the size of the array\n");
    scanf("%d", &n);

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

heapify(a,n);
    for(int i=0; i<n; i++){
        printf("%d\t", a[i]);
    }
}

void heapify(int a[], int n){
    int c, key, p;
    for(int k=1; k<=n-1; k++){
        key = a[k];
        c = k;
        p = (c-1)/2;
    }
```

PAGE NO :
DATE :

```
while (c > 0 && key > a[p]) {
        a[c] = a[p];
        c = p;
        p = (c-1)/2;
    }
    a[c] = key;
}
```

O/P:

Enter the size of array
5
Enter array elements in unsorted manner!
3 4 5 1 2
1 2 3 4 5.

21/6/24

PAGE NO:
DATE:

Prim Algorithm:

```
#include <stdio.h>
int i, j, u, v, sum, k, min; source;
int t[10][2], cost[10][10];
int p[10], d[10], s[10];

void prim(int cost[10][10], int n);

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

    printf("Enter the adjacency matrix:\n
    for(i=0; i<n; i++){
        for(j=0; j<n; j++){
            scanf("%d", &cost[i][j]);
        }
    }
    min=999;
    prim(cost, n);
    printf("The edges of the minimum
spanning tree are:\n");
    for(i=0; i<n-1; i++){
        printf("%d, %d)\n", t[i][0], t[i][1]);
}
    printf("The minimum cost of spanning tree
is: %d\n", sum);
}
```

```
void prims(int cost[10][10], int n){
    for(int i=0; i<n; i++){
        for(int j=0; j<n; j++){
            if(cost[i][j]!=0 && cost[i][j]<min){
                min = cost[i][j];
                source = i;
            }
        }
    }
    for(int i=0; i<n; i++){
        d[i] = cost[source][i];
        s[i] = 0;
        p[i] = source;
    }

    s[source]=1;
    sum=0;
    k=0
    for(int i=1; i<n; i++){
        min=999;
        u=-1;
        for(int j=0; j<n; j++){
            if(s[j]==0){
                if(d[j]<min){
                    min=d[j];
                    u=j;
                }
            }
        }
        t[k][0]=u;
        t[k][1]=p[u];
        k++;
        sum=sum+cost[u][p[u]];
```

```
        S[u] = 1;

        for(int v = 0; v < n; v++){
            if [S[v] == 0 && cost [u][v] < d[v]){
                d[v] = cost[u][v];
                p[v] = u;
            }
        }
    }
}
```

OIP:

Enter the no of vertices: 4
Enter the adjacency matrix:
0 1 5 2
1 0 9999 9999
5 9999 0 3
2 9999 3 0
The edges of minimum spanning tree are
(1,0)
(3,0)
(2,3)
The minimum cost of spanning tree is: 6

Kruskal:

```
#include<stdio.h>
int count=0;
int i, j, u, v, k=0, min, sum=0;
int t[10][10], cost[10][10];
int p[10], d[10];
```

```
void kruskal(int cost[10][10], int n);
int find(int i);
void union1(int i, int j);

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

    printf("Enter the cost adjacency matrix");
    for(i=0; i<n; i++){
        for(j=0; j<n; j++){
            scanf("%d", &cost[i][j]);
            if(cost[i][j]==0){
                cost[i][j]=999;
            }
        }
    }
    kruskal(cost, n);

printf("The edges in the minimum spanning tree are:\n");
for(i=0; i<k; i++){
    printf("(%d, %d)\n", t[i][0], t[i][1]);
}
    printf("Minimum cost: %d\n", sum);
}

void kruskal(int cost[10][10], int n){
    for(int i=0; i<n; i++){
        p[i]=i;
    }
```

```
while (count < n-1){
        min = 999;
        for(int i=0; i<n; i++){
          for(int j=0; j<n; j++){
            if (cost[i][j] < min){
              min = cost[i][j];
              u = i;
              v = j;
            }
          }
        }
      if (find(u) != find(v)){
          t[k][0] = u;
          t[k][1] = v;
          k++;
          count++;
          sum += min;
          union1(u, v);
      }
      cost [u][v] = cost[v][u] = 999;
  }
}
int find (int i){
  while (p[i] != i){
      i = p[i];
  }
  return i;
}

void union1 (int i, int j){
      int a = find(i);
      int b = find(j);
      p[a] = b;
}
```

PAGE NO:
DATE:

Enter the no of vertices: 4
Enter the cost adjacency matrix:
0 1 5 2
1 0 9999 9999
5 9999 0 3
2 9999 3 0
The edges in the minimum spanning spanning tree are:
(0,1)
(0,3)
(2,3)
Minimum cost: 6

Dijkstras:

```
#include <stdio.h>
int c[10][10];
int d[10], p[10], visited[10];
int s, min, u;

void dijkstras(int c[10][10], int n);

void main(){
    int n;
    printf("Enter the no: of vertices:");
    scanf("%od", &n);

    printf("Enter the cost adjaconcy
matrix:\n");
    for(int i=0; i<n; i++){
        for(int j=0; j<n; j++){
            scanf("%od", &c[i][j]);
            if(c[i][j]==0 && i!=j){
```

```
            c[i][j] = 999;
        }
    }
}

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

dijkstra(c, n);

printf("The shortest paths from vertex %d are \n", s);
for(int i=0; i<n; i++){
    if(i != s){
        printf("To vertex %d: Distance = %d, path = %d", i, d[i]);
        int j=i;
        while(p[j] != s){
            j = p[j];
            printf(" <- %d", j);
        }
        printf(" <- %d\n", s);
    }
}
}

void dijkstra(int c[10][10], int n){
    for(int i=0; i<n; i++){
        d[i] = c[s][i];
        visited[i] = 0;
        p[i] = s;
    }
    visited[s] = 1;
    d[s] = 0;
```

PAGE NO :
DATE :

```
for(int count = 1; count < n-1; count++){
    min = 999;
    for(int i=0; i<n; i++){
        if(!visited[i] && d[i] < min){
            min = d[i];
            u = i;
        }
    }
    visited[u] = 1;
    for(int v=0; v<n; v++){
        if(!visited[v] && c[u][v] != 9999 &&
(d[u] + c[u][v] < d[v])){
            d[v] = d[u] + c[u][v];
            p[v] = u;
        }
    }
}
```

O/P:

```
Enter the no of vertices: 4
Enter the cost adjacency matrix:
0 1 5 2
1 0 9999 9999
5 9999 0 3
2 9999 3 0
Enter the source vertex: 1
The shortest paths from vertex 1 are:
To vertex 0: Dist = 1; Path = 0 <- 1
To Vertex 2: Dist = 6; Path = 2 <- 0 <- 1
To vertex 3: Dist = 3; Path = 3 <- 0 <- 1.
```

PAGE NO:
DATE:

Knapsack:

```
#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;
    while ( cur_w>0) {
       maxi = -1;
      if (w[maxi] <= cur_w) {
      cur_w = w [maxi];
      total_v + p [maxi];
      } else {
          int taken = cur_w;
          cur_w =0;
          total v = taken/p [maxi] * p [max
  } }
}
int main(){
    printf (" Enter the no 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 max weight of the bag");
scanf("%d", &W);

knapsack(n, p, w, W);
return 0;
}
```

O/P

Enter the no: of objects: 7
Enter the profits of objects: 5 10 15 7 8 9 4
Enter the weights of objects: 1 3 5 4 1 3 2
Enter the maximum weight of the bag - 15

Added object 4 (4, 7) completely in the bag.
Space left 11.
Added object 7 (2, 4) completely in bag.
Space left 9.
Added object 3 (5, 15) completely in bag.
Space left 4.
Added object 6 (3, 9) completely in bag.
Space left 1
Added 33% (3, 10) of object 2 in the bag.
Filled the bag with objects worth 36.00.

12/7/24

PAGE NO
DATE

N-Queens:

```
#include<stdio.h>
#include<stdbool.h>
bool place(int [], int);
void printSolution(int [], int);
void nQueens(int);
int main(){
int n;
printf("Enter the no of queens: ");
scanf("%d", &n);
nQueens(n);
return 0;
}
void nQueens(int n){
int x[10];
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[10], int k) {
  int i;
  for(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[10], int n){
  int i;
  for(i=1; i<=n; i++){
  printf("%d", x[i]);
}
  printf("\n");
}
```

```
Enter the no of queens: 4
2 4 1 3
Solution found
3 1 4 2
Solution found
Total Solutions: 2
```

19/7/24