VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Analysis and Design of Algorithms

Submitted by

Preethi Narasimhan (1BM22CS207)

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

<u>CERTIFICATE</u>

This is to certify that the Lab work entitled "**Analysis and Design of Algorithms**" carried out by **PREETHI NARASIMHAN (1BM22CS207),** 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.

M Lakshmi Neelima   Dr. Jyothi S Nayak

Assistant Professor   Professor and Head

Department of CSE   Department of CSE

BMSCE, Bengaluru   BMSCE, Bengaluru

Index Sheet

12	Implement Fractional Knapsack using Greedy technique.	38-40
13	From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm.	41-43
14	Implement "N-Queens Problem" using Backtracking.	44-45

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.

1. Leetcode exercises on Stacks, Queues, Circular Queues, Priority Queues.

Repeated Substring Matching

```c
bool repeatedSubstringPattern(char* s) {
    int n = strlen(s);
    for (int len = 1; len <= n / 2; len++) {
        if (n % len == 0) {
            bool isRepeated = true;
            for (int i = len; i < n; i++) {
                if (s[i] != s[i % len]) {
                    isRepeated = false;
                    break;
                }
            }
            if (isRepeated) return true;
        }
    }
    return false;}
```

OUTPUT:
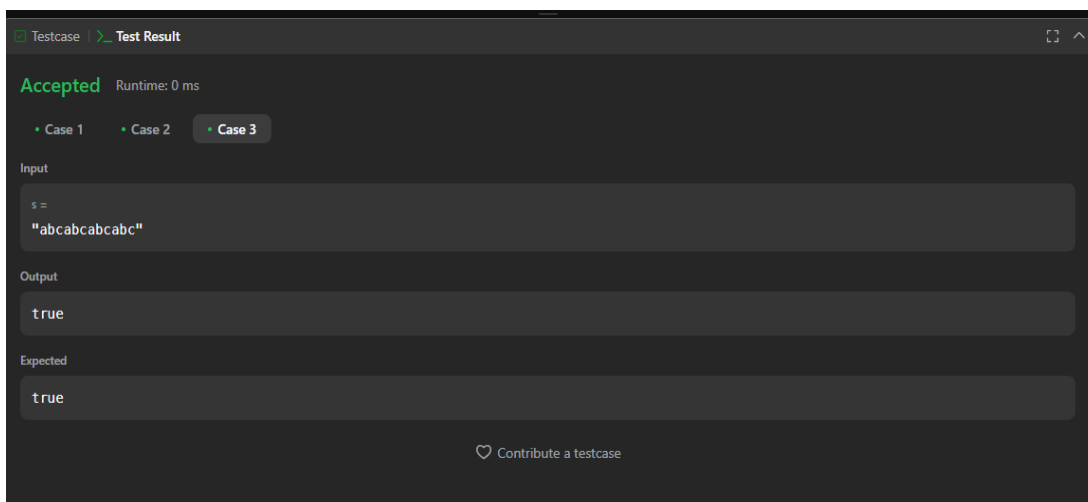
2. Leetcode exercises on DFS, BFS

Kth Largest Sum in a Binary Tree

```c
void dfs(struct TreeNode* root, int level, long long levelSum[]) {

    if (root == NULL)

        return;

    levelSum[level] += root->val;

    dfs(root->left, level + 1, levelSum);

    dfs(root->right, level + 1, levelSum);

}

long long kthLargestLevelSum(struct TreeNode* root, int k) {

    if (root == NULL)

        return -1; // If the tree is empty

    long long* levelSum = (long long*)calloc(1000, sizeof(long long));

    dfs(root, 0, levelSum);


    // Find the kth largest level sum

    long long* levelSums = (long long*)malloc(1000 * sizeof(long long)); // Dynamically
allocate array

    int numLevels = 0;

    for (int i = 0; i < 1000 && levelSum[i] != 0; ++i) {

        levelSums[numLevels++] = levelSum[i];

    }

    for (int i = 0; i < numLevels - 1; ++i) {

        for (int j = i + 1; j < numLevels; ++j) {

            if (levelSums[i] < levelSums[j]) {

                long long temp = levelSums[i];

                levelSums[i] = levelSums[j];

                levelSums[j] = temp;

            }

        }
```

```
        }
    if (k <= numLevels) {

        return levelSums[k - 1];

    }

    else {

        return -1; // If there are fewer than k levels in the tree

    }

}
```

OUTPUT:
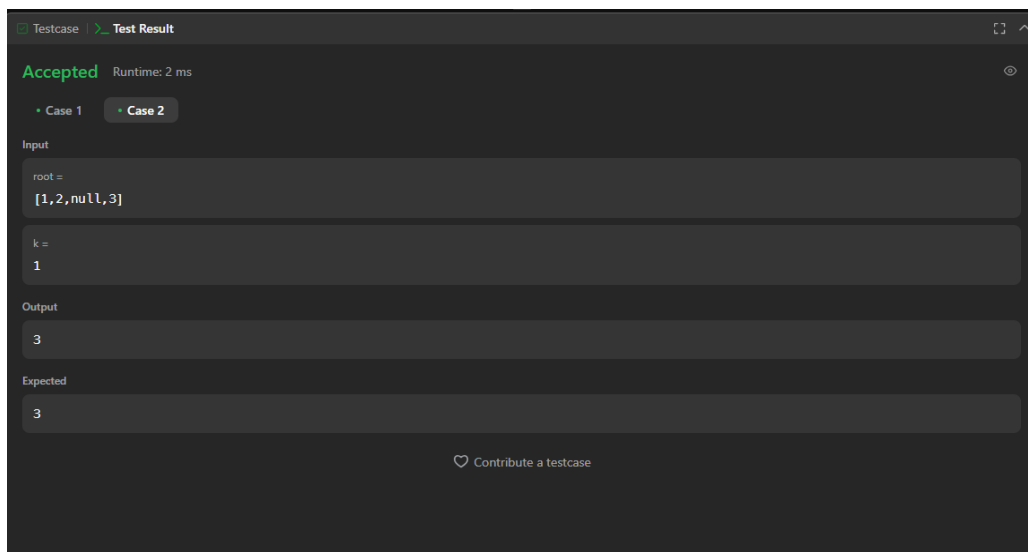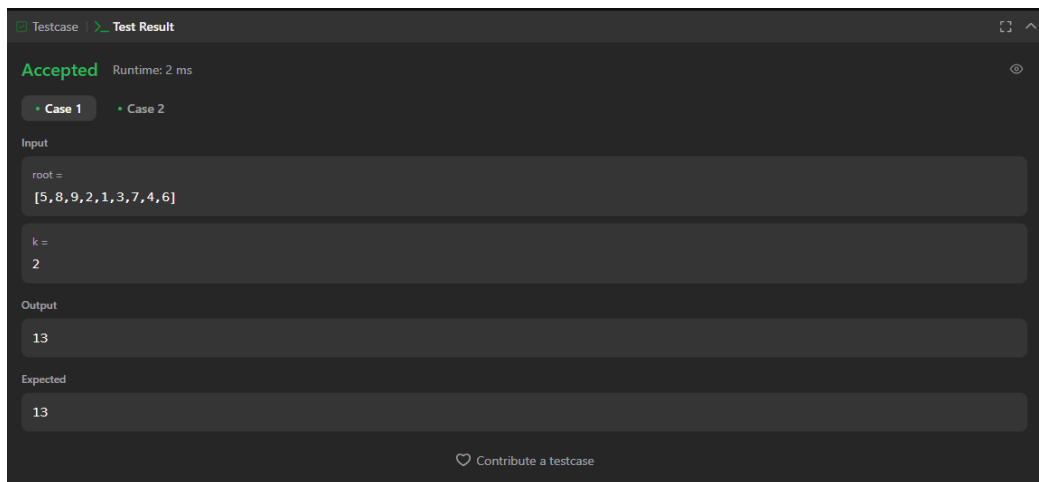
3. **Leetcode exercises on Trees and Graphs.**

   Increasing Order Search Tree

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

OUTPUT:

Testcase | >_ **Test Result**

Accepted Runtime: 2 ms

• Case 1   • Case 2

Input

root =
[5,3,6,2,4,null,8,1,null,null,null,7,9]

Output

[1,null,2,null,3,null,4,null,5,null,6,null,7,null,8,null,9]

Expected

[1,null,2,null,3,null,4,null,5,null,6,null,7,null,8,null,9]

♡ Contribute a testcase
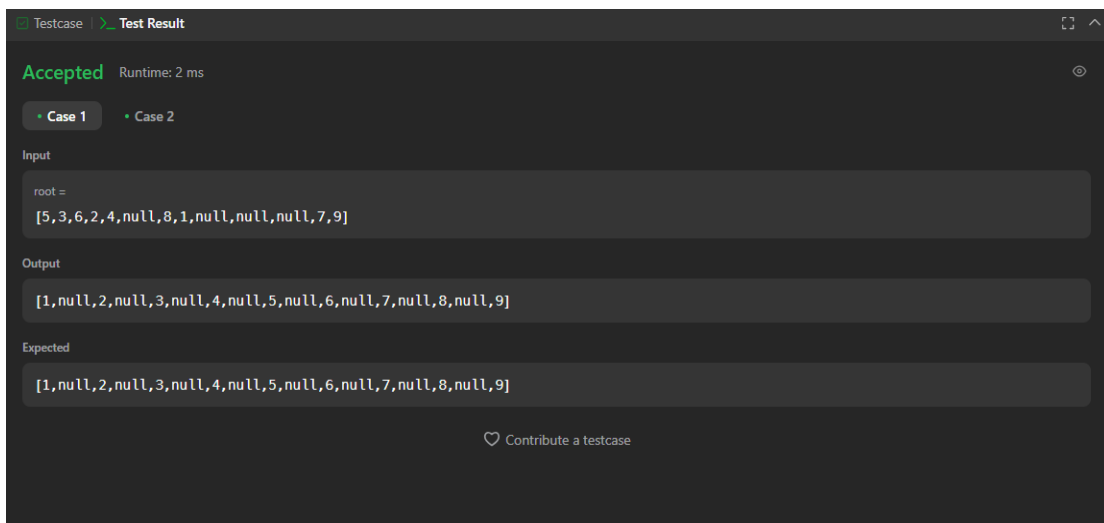
4. **Write program to obtain the Topological ordering of vertices in a given digraph.**

//Topological Sorting – Source Removal Technique

```c
#include <stdio.h>

void topologicalSort(int a[20][20], int n) {
    int i, j, k;
    int sum, top = -1, indegree[20], s[20], u, v, T[20];
    for (j = 0; j < n; j++) {
        sum = 0;
        for (i = 0; i < n; i++) {
            sum += a[i][j];
        }
        indegree[j] = sum;
    }
    for (i = 0; i < n; i++) {
        if (indegree[i] == 0) {
            top++;
            s[top] = i;
        }
    }
    k = 0;
    while (top != -1) {
        u = s[top];
        top--;
        T[k++] = u;
        for (i = 0; i < n; i++) {
            if (a[u][i] != 0) {  // Edge exists
                v = i;
                indegree[v]--;
                if (indegree[v] == 0) {
```

```c
                top++;
                s[top] = v;
            }
        }
    }
    printf("Topological Order:\n");
    for (i = 0; i < n; i++) {
        printf("%d\t", T[i]);
    }
    printf("\n");
}
int main() {
    int a[20][20], n;
    printf("Enter number of vertices (maximum 20): ");
    scanf("%d", &n);
    printf("Enter adjacency matrix:\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            scanf("%d", &a[i][j]);
        }
    }
    topologicalSort(a, n);
    return 0;
}
```

OUTPUT:

```
Enter number of vertices (maximum 20): 4
Enter adjacency matrix:
0 1 1 1
0 0 0 1
0 0 0 0
0 0 1 0
Topological Order:
0    1    3    2


=== Code Execution Successful ===
```

//DFS Topological Sorting

```c
#include <stdio.h>
int a[20][20],n,res[20],visited[20],j=0;
void DFS(int v){
    visited[v]=1;
    for(int i=0;i<n;i++){
        if(a[i][v]==1 && visited[i]==0)
            DFS(i);
    }
    res[j++]=v;
}
void main()
{
    int i,j,v;
    printf("Enter no. of vertices:");
    scanf("%d",&n);
    printf("Enter adjacency matrix:\n");
```

```
    for(int i=0;i<n;i++){
        for(j=0;j<n;j++)
            scanf("%d",&a[i][j]);
    }
    for(int i=0;i<n;i++)
        visited[i]=0;
    for(v=0;v<n-1;v++){
        if(visited[v]==0)
            DFS(v);
    }
    printf("Topological Order:\n");
    for(i=0;i<n;i++)
        printf("%d\t",res[i]);
}
```

OUTPUT:

```
Enter no. of vertices:5
Enter adjacency matrix:
0 1 0 1 0
0 0 0 1 0
0 1 0 0 0
0 0 0 0 0
1 0 0 1 0
Topological Order:
4   0   2   1   3

=== Code Exited With Errors ===
```

5. Implement Johnson Trotter algorithm to generate permutations.

```c
#include <stdio.h>
#include <stdbool.h>
#define MAXN 10
// Direction array, dir[i] stores the direction of ith element in permutation
int dir[MAXN];
int n; // Number of elements in the permutation
// Function to print the current permutation
void printPermutation(int perm[]) {
    for (int i = 0; i < n; i++)
        printf("%d ", perm[i]);
    printf("\n");
}
// Function to swap two integers
void swap(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}
// Function to generate all permutations using Johnson-Trotter algorithm
void generatePermutations() {
    int perm[MAXN]; // Current permutation
    for (int i = 0; i < n; i++) {
        perm[i] = i + 1; // Initialize permutation to 1, 2, ..., n
        dir[i] = -1; // All directions initially set to -1 (left)
    }
    printPermutation(perm); // Print the initial permutation

    int k, mobile, pos;
```

```c
        bool found;
        while (true) {
            // Step 1: Find the largest mobile integer
            mobile = -1;
            for (int i = 0; i < n; i++) {
                if ((dir[i] == -1 && i != 0 && perm[i] > perm[i - 1]) ||
                    (dir[i] == 1 && i != n - 1 && perm[i] > perm[i + 1])) {
                    if (mobile == -1 || perm[i] > perm[mobile]) {
                        mobile = i;
                    }
                }
            }
            if (mobile == -1) // No more mobile integers, algorithm terminates
                break;
            // Step 2: Swap the mobile integer with the adjacent integer it is pointing to
            k = mobile + dir[mobile];
            swap(&perm[mobile], &perm[k]);
            swap(&dir[mobile], &dir[k]);
            // Step 3: Reverse the direction of all integers greater than the mobile integer
            for (int i = 0; i < n; i++) {
                if (perm[i] > perm[k]) {
                    dir[i] = -dir[i];
                }
            }
            // Print the current permutation
            printPermutation(perm);
        }
}
int main() {
    printf("Enter the number of elements (maximum %d): ", MAXN);
```

```c
    scanf("%d", &n);

    if (n <= 0 || n > MAXN) {

        printf("Invalid input. Number of elements should be between 1 and %d.\n",
MAXN);

        return 1;

    }

    printf("Permutations:\n");

    generatePermutations(); // Generate permutations using Johnson-Trotter algorithm

    return 0;

}
```

OUTPUT:

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


=== Code Execution Successful ===
```

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

```c
#include<stdio.h>

#include<time.h>

#include<stdlib.h>

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

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

void main()

{

  int a[15000],n, i,j,ch, temp;

  clock_t start,end;

   printf("1: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\n");

  while(1)  {

   printf("\nEnter your choice:");

   scanf("%d", &ch);

   switch(ch){

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

              scanf("%d",&n);

              printf("Enter array elements: ");

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

              scanf("%d",&a[i]);

               }

              start=clock();

              split(a,0,n-1);

              end=clock();

              printf("Sorted array is: ");
```

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

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

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

                break;

    case 2:

            n=500;

            while(n<=14500) {

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

                {

                 //a[i]=random(1000);

                 a[i]=n-i;

                }

            start=clock();

            split(a,0,n-1);

        //Dummy loop to create delay

            for(j=0;j<500000;j++){ temp=38/600;}

            end=clock();

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

                n=n+1000;

                }

            break;

    case 3: exit(0);

    }

getchar();

    }

}

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

 int mid;
```

13

```
 if(low<high){
  mid=(low+high)/2;
  split(a,low,mid);
  split(a,mid+1,high);
  combine(a,low,mid,high);
 }
}
void combine(int a[],int low,int mid,int high){
 int c[15000],i,j,k;
 i=k=low;
 j=mid+1;
 while(i<=mid&&j<=high){
  if(a[i]<a[j]){
   c[k]=a[i];
   ++k;
   ++i;
  }
  else{
   c[k]=a[j];
   ++k;
   ++j;
  }
 }
 if(i>mid){
  while(j<=high){
  c[k]=a[j];
   ++k;
   ++j;
  }
 }
```

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

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:2
Time taken to sort 500 numbers is 0.000550 Secs
Time taken to sort 1500 numbers is 0.000605 Secs
Time taken to sort 2500 numbers is 0.000643 Secs
Time taken to sort 3500 numbers is 0.000727 Secs
Time taken to sort 4500 numbers is 0.000792 Secs
Time taken to sort 5500 numbers is 0.000879 Secs
Time taken to sort 6500 numbers is 0.000978 Secs
Time taken to sort 7500 numbers is 0.001026 Secs
Time taken to sort 8500 numbers is 0.001104 Secs
Time taken to sort 9500 numbers is 0.001184 Secs
Time taken to sort 10500 numbers is 0.001271 Secs
Time taken to sort 11500 numbers is 0.001362 Secs
Time taken to sort 12500 numbers is 0.001440 Secs
Time taken to sort 13500 numbers is 0.001542 Secs
Time taken to sort 14500 numbers is 0.001622 Secs
```

```
Enter your choice:1
Enter the number of elements: 6
Enter array elements: 8 3 4 1 6 7
Sorted array is: 1  3   4   6   7   8
Time taken to sort 6 numbers is 0.000003 Secs

Enter your choice:3


=== Code Execution Successful ===
```

Graph:

N	Time
500	0.00055
1500	0.000605
2500	0.000643
3500	0.000727
4500	0.000792
5500	0.000879
6500	0.000978
7500	0.001026
8500	0.001104
9500	0.001184
10500	0.001271
11500	0.001362
12500	0.00144
13500	0.001542
14500	0.001622

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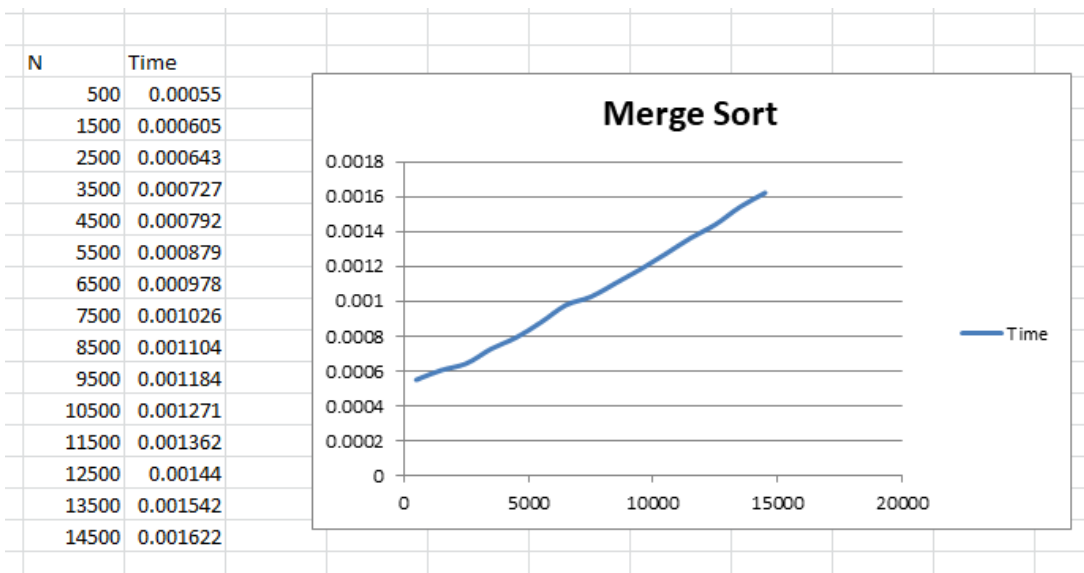
7. Sort a given set of N integer elements using Quick Sort technique and compute its time taken.

```c
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
void quicksort(int [], int, int);
int partition(int [], int, int);
void swap(int *, int *);
int main() {
    int a[15000], n, i, j, temp;
    clock_t start, end;
    printf("1: For manual entry of N value and array elements\n");
    printf("2: To display time taken for sorting number of elements N in the range 500 to 14500\n");
    printf("3: To exit\n");
    int ch;
    while (1) {
        printf("\nEnter your choice: ");
        scanf("%d", &ch);
        switch (ch) {
            case 1:
                printf("Enter 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();
```

```c
        printf("Sorted array: ");
         for (i = 0; i < n; i++) {
           printf("%d ", a[i]);
        }
        printf("\nTime taken to sort %d numbers is %f seconds\n", 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);
           end = clock();
           printf("Time taken to sort %d numbers is %f seconds\n", n, ((double)(end -
start)) / CLOCKS_PER_SEC);
           n = n + 1000; // Increment n after each iteration
         }
         break;
      case 3:
         exit(0);
      default:
         printf("Invalid choice! Please enter again.\n");
         break;
    }
    getchar();
  }
```

```c
        return 0;
    }

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

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

        do {
            do {
                i++;
            } while (a[i] < pivot && i <= high);

            do {
                j--;
            } while (a[j] > pivot && j >= low);

            if (i < j) {
                swap(&a[i], &a[j]);
            }
        } while (i < j);
        swap(&a[low], &a[j]);
        return j;
```

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

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: 2
Time taken to sort 500 numbers is 0.000163 seconds
Time taken to sort 1500 numbers is 0.001387 seconds
Time taken to sort 2500 numbers is 0.003564 seconds
Time taken to sort 3500 numbers is 0.008917 seconds
Time taken to sort 4500 numbers is 0.015828 seconds
Time taken to sort 5500 numbers is 0.019442 seconds
Time taken to sort 6500 numbers is 0.037293 seconds
Time taken to sort 7500 numbers is 0.047849 seconds
Time taken to sort 8500 numbers is 0.052973 seconds
Time taken to sort 9500 numbers is 0.072288 seconds
Time taken to sort 10500 numbers is 0.086499 seconds
Time taken to sort 11500 numbers is 0.109287 seconds
Time taken to sort 12500 numbers is 0.128227 seconds
Time taken to sort 13500 numbers is 0.147480 seconds
Time taken to sort 14500 numbers is 0.174636 seconds
```
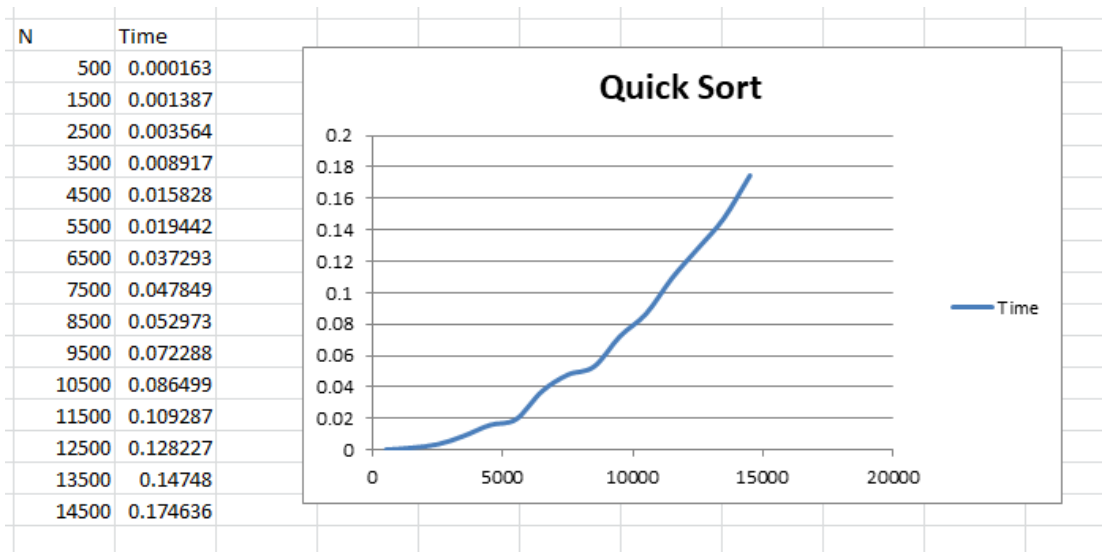
```
Enter your choice: 1
Enter the number of elements: 7
Enter array elements: 10 4 3 7 5 1 9
Sorted array: 1 3 4 5 7 9 10
Time taken to sort 7 numbers is 0.000003 seconds

Enter your choice: 3


=== Code Execution Successful ===
```

Graph:

N	Time
500	0.000163
1500	0.001387
2500	0.003564
3500	0.008917
4500	0.015828
5500	0.019442
6500	0.037293
7500	0.047849
8500	0.052973
9500	0.072288
10500	0.086499
11500	0.109287
12500	0.128227
13500	0.14748
14500	0.174636

Quick Sort

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

```c
#include<stdio.h>

#include<time.h>

#include<stdlib.h>

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

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

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

void main(){

   int a[15000], n, i, j, ch, temp;

   clock_t start, end;

   printf("\n 1:For sorting of 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");

   while (1)

   {

      printf("\nEnter your choice:");

      scanf("%d", &ch);

      switch (ch){

      case 1:

         printf("Enter the number of elements: ");

         scanf("%d", &n);

         printf("Enter array elements: ");

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

         {

            scanf("%d", &a[i]);

         }

         start = clock();

         heapsort(n, a);
```

```c
        end = clock();
         printf("Sorted array elements are\n");
    for (i = 0; i < n; i++){
       printf("%d ", a[i]);
    }
        printf("\n");
        printf("Time taken to sort %d numbers is %f Secs\n", 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();
    heapsort(n,a);
       //Dummy loop to create delay
   for(j=0;j<500000;j++){ temp=38/600;}
         end=clock();
        printf("Time taken to sort %d numbers is %f Secs\n",n, (((double)(end-
start))/CLOCKS_PER_SEC));
        n=n+1000;
 }
        break;
     case 3:
       exit(0);
     }
     getchar(); // Consume newline character left in input buffer
```

```
        }
    }
    void heapify(int n, int a[])
    {
        int i, p, c, item;
        for (p = (n - 1) / 2; p >= 0; p--){
            item = a[p];
            c = 2 * p + 1;
            while (c < n){
                if (c + 1 < n && a[c] < a[c + 1])
                {
                    c++;
                }
                if (item >= a[c])
                    break;
                a[p] = a[c];
                p = c;
                c = 2 * p + 1;
            }
            a[p] = item;
        }
    }
    void swap(int* a, int* b)
    {
        int temp;
        temp = *a;
        *a = *b;
        *b = temp;
    }
    void heapsort(int n, int a[])
```

```
{
    int i;

    heapify(n, a);

    for (i = n - 1; i > 0; i--)

    {

        swap(&a[0], &a[i]);

        heapify(i, a);

    }

}
```

OUTPUT:

```
 1:For sorting of 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 0.001497 Secs
Time taken to sort 1500 numbers is 0.006219 Secs
Time taken to sort 2500 numbers is 0.016747 Secs
Time taken to sort 3500 numbers is 0.034821 Secs
Time taken to sort 4500 numbers is 0.048089 Secs
Time taken to sort 5500 numbers is 0.072304 Secs
Time taken to sort 6500 numbers is 0.097209 Secs
Time taken to sort 7500 numbers is 0.136986 Secs
Time taken to sort 8500 numbers is 0.183444 Secs
Time taken to sort 9500 numbers is 0.281507 Secs
Time taken to sort 10500 numbers is 0.330495 Secs
Time taken to sort 11500 numbers is 0.355252 Secs
Time taken to sort 12500 numbers is 0.389277 Secs
Time taken to sort 13500 numbers is 0.429252 Secs
Time taken to sort 14500 numbers is 0.559296 Secs
```
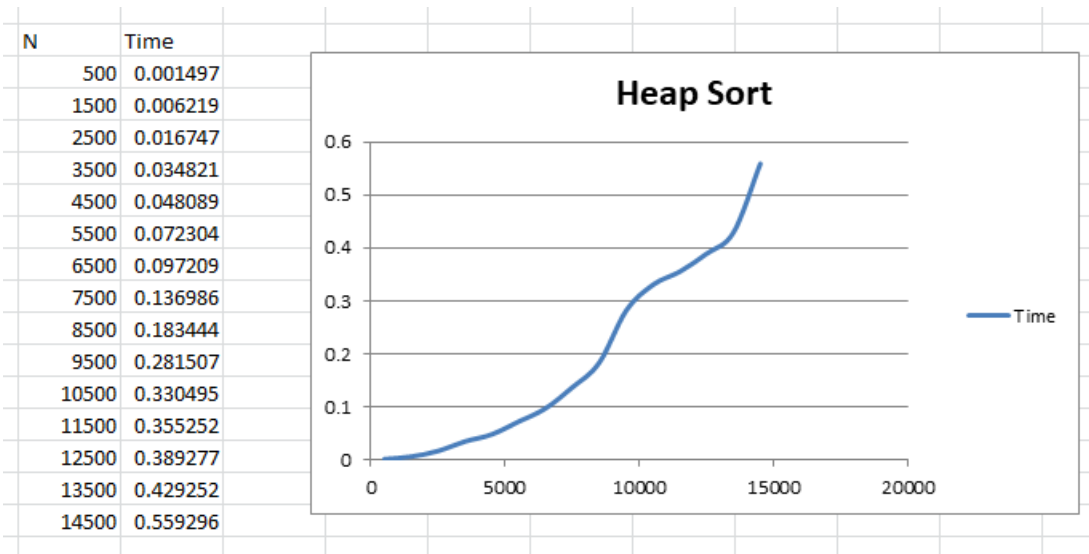
```
Enter your choice:1
Enter the number of elements: 6
Enter array elements: 1 7 2 1 6 4
Sorted array elements are
1 1 2 4 6 7
Time taken to sort 6 numbers is 0.000002 Secs

Enter your choice:3


=== Code Execution Successful ===
```

Graph:

N	Time
500	0.001497
1500	0.006219
2500	0.016747
3500	0.034821
4500	0.048089
5500	0.072304
6500	0.097209
7500	0.136986
8500	0.183444
9500	0.281507
10500	0.330495
11500	0.355252
12500	0.389277
13500	0.429252
14500	0.559296

Heap Sort

9. Implement 0/1 Knapsack problem using dynamic programming.

```c
#include <stdio.h>
#define N 4
int max(int a, int b) {
   return (a > b) ? a : b;
}
void knapsack(int W, int weights[], int profits[]) {
   int dp[N + 1][W + 1];
   for (int i = 0; i <= N; i++) {
      for (int w = 0; w <= W; w++) {
         if (i == 0 || w == 0)
            dp[i][w] = 0;
         else if (weights[i - 1] <= w)
            dp[i][w] = max(profits[i - 1] + dp[i - 1][w - weights[i - 1]], dp[i - 1][w]);
         else
            dp[i][w] = dp[i - 1][w];
      }
   }
   int maxProfit = dp[N][W];
   printf("Maximum Profit: %d\n", maxProfit);
   int w = W;
   printf("Objects selected in the knapsack:\n");
   for (int i = N; i > 0 && maxProfit > 0; i--) {
      if (maxProfit == dp[i - 1][w])
         continue;
      else {
         printf("Object %d (Weight = %d, Profit = %d)\n", i, weights[i - 1], profits[i - 1]);
         maxProfit -= profits[i - 1];
         w -= weights[i - 1];
```

```c
        }
    }
}
int main() {
    int weights[20],n;
    int profits[20],W;
    printf("Enter number of weights: ");
    scanf("%d", &n);
    printf("Enter Maximum wight:");
    scanf("%d",&W);
    printf("Enter the weights:\n");
    for (int j = 0; j < n; j++) {
        scanf("%d",&weights[j]);
    }
    printf("Enter the profits:\n");
    for (int j = 0; j < n; j++) {
        scanf("%d",&profits[j]);
    }
    knapsack(W, weights, profits);
    return 0;
}
```

OUTPUT:

```
Enter number of weights: 5
Enter Maximum wight:8
Enter the weights:
1 4 3 2 1
Enter the profits:
10 20 15 13 11
Maximum Profit: 45
Objects selected in the knapsack:
Object 3 (Weight = 3, Profit = 15)
Object 2 (Weight = 4, Profit = 20)
Object 1 (Weight = 1, Profit = 10)


=== Code Execution Successful ===
```

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

```c
#include <stdio.h>

#include <limits.h>

void floyd(int n, int cost[][n], int D[][n]) {

  int i, j, k;

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

    for (j = 0; j < n; j++) {

      D[i][j] = cost[i][j];

    }

  }

  for (k = 0; k < n; k++) {

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

      for (j = 0; j < n; j++) {

        if (D[i][k] != INT_MAX && D[k][j] != INT_MAX && D[i][j] > D[i][k] +
D[k][j]) {

          D[i][j] = D[i][k] + D[k][j];

        }

      }

    }

  }

}

void printShortestPaths(int n, int D[][n]) {

  printf("Shortest paths between every pair of vertices:\n");

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

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

      if (D[i][j] == INT_MAX) {

        printf("INF\t");
```

```c
            } else {

                printf("%d\t", D[i][j]);

            }

        }

        printf("\n");

    }

}

int main() {

    int n;

    printf("Enter the number of vertices in the graph: ");

    scanf("%d", &n);

    int cost[n][n];

    printf("Enter the cost adjacency matrix (use '-1' for infinity):\n");

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

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

            scanf("%d", &cost[i][j]);

            if (cost[i][j] == -1) {

                cost[i][j] = INT_MAX;

            }

        }

    }

    int D[n][n];

    floyd(n, cost, D);

    printShortestPaths(n, D);

    return 0;

}
```

OUTPUT:

```
Enter the number of vertices in the graph: 5
Enter the cost adjacency matrix (use '-1' for infinity):
0  2  -1  -1  3
4  0   3  -1  -1
7  -1  0  -1  -1
8  -1  1  0   -1
-1  9  2  -1  0
Shortest paths between every pair of vertices:
0   2   5   INF 3
4   0   3   INF 7
7   9   0   INF 10
8   10  1   0   11
9   9   2   INF 0


=== Code Execution Successful ===
```

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

```c
#include <stdio.h>
#define MAX 9999
void prims(int n, int cost[n][n]) {
    int i, j, u, min, sum = 0, source, K = 0;
    int S[n], d[n], P[n], T[n-1][2];
    min = MAX;
    source = 0;
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            if (cost[i][j] != 0 && cost[i][j] < min) {
                min = cost[i][j];
                source = i;
            }
        }
    }
    for (i = 0; i < n; i++) {
        S[i] = 0;
        d[i] = cost[source][i];
        P[i] = source;
    }
    S[source] = 1;
    for (i = 1; i < n; i++) {
        min = MAX;
        u = -1;
        for (j = 0; j < n; j++) {
            if (S[j] == 0 && d[j] <= min) {
                min = d[j];
                u = j;
```

```c
        }
      }
      T[K][0] = u;
      T[K][1] = P[u];
      K++;
      sum += cost[u][P[u]];
      S[u] = 1;
      for (j = 0; j < n; j++) {
        if (S[j] == 0 && cost[u][j] < d[j]) {
          d[j] = cost[u][j];
          P[j] = u;
        }
      }
    }
    if (sum >= MAX) {
      printf("Spanning tree does not exist.\n");
    } else {
      printf("Spanning tree exists and MST is:\n");
      for (i = 0; i < n-1; i++) {
        printf("%d - %d\n", T[i][0], T[i][1]);
      }
      printf("The cost of spanning tree (MST) is %d\n", sum);
    }
}
int main() {
    int n;
    printf("Enter number of vertices: ");
    scanf("%d", &n);
    int cost[n][n];
    printf("Enter the cost adjacency matrix:\n");
```

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

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

            scanf("%d", &cost[i][j]);

        }

    }

    prims(n, cost);

    return 0;

}
```

OUTPUT:

```
Enter number of vertices: 4
Enter the cost adjacency matrix:
0 1 3 9999
1 0 1 9999
3 1 0 2
9999 9999 2 0
Spanning tree exists and MST is:
1 - 0
2 - 1
3 - 2
The cost of spanning tree (MST) is 4


=== Code Execution Successful ===
```

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

```c
#include <stdio.h>

#define MAX 9999 // Infinity value assumed

void kruskals(int c[][100], int n);

int main() {

    int n, i, j;

    int c[100][100]; // Assuming a maximum size for the cost matrix

    printf("Enter the number of nodes: ");
```

```c
    scanf("%d", &n);
    printf("Enter the cost matrix:\n");
    for (i = 1; i <= n; i++) {
        for (j = 1; j <= n; j++) {
            scanf("%d", &c[i][j]);
            if (c[i][j] == 0) // Assuming 0 represents no edge, set it to a large value
                c[i][j] = MAX;
        }
    }
    kruskals(c, n);
    return 0;
}
void kruskals(int c[][100], int n) {
    int ne = 0, mincost = 0;
    int parent[100];
    int min, u, v, a, b, i, j;
    for (i = 1; i <= n; i++)
        parent[i] = 0;
    while (ne != n - 1) {
        min = MAX;
        for (i = 1; i <= n; i++) {
            for (j = 1; j <= n; j++) {
                if (c[i][j] < min) {
                    min = c[i][j];
                    u = a = i;
                    v = b = j;
                }
            }
        }
        while (parent[u] != 0)
```

```
        u = parent[u];
      while (parent[v] != 0)
        v = parent[v];
      if (u != v) {
        printf("Edge %d-%d: %d\n", a, b, min);
        parent[v] = u;
        mincost += min;
        ne++;
      }
      c[a][b] = c[b][a] = MAX;
    }
    printf("Minimum cost of spanning tree: %d\n", mincost);
  }
```

OUTPUT:

```
Enter the number of nodes: 4
Enter the cost matrix:
0 6 1 4
2 0 3 4
3 1 0 5
1 1 1 0
Edge 1-3: 1
Edge 3-2: 1
Edge 4-1: 1
Minimum cost of spanning tree: 3


=== Code Execution Successful ===
```

12. Implement Fractional Knapsack using Greedy technique.

```c
#include<stdio.h>
void knapsack(int n, float weight[], float profit[], float capacity)
{
  float x[20], tp = 0;
  int i, j, u;
  u = capacity;
  for (i = 0; i < n; i++)
    x[i] = 0.0;
  for (i = 0; i < n; i++) {
    if (weight[i] > u)
      break;
    else {
      x[i] = 1.0;
      tp = tp + profit[i];
      u = u - weight[i];
    }
  }
  if (i < n)
    x[i] = u / weight[i];
  tp = tp + (x[i] * profit[i]);
  printf("\nThe result vector is:- ");
  for (i = 0; i < n; i++)
    printf("%f\t", x[i]);
  printf("\nMaximum profit is:- %f", tp);
}
int main() {
  float weight[20], profit[20], capacity;
  int num, i, j;
```

```c
float ratio[20], temp;
printf("Enter the no. of objects:-");
scanf("%d", &num);
printf("Enter the wts of each object:-\n");
for (i = 0; i < num; i++) {
  scanf("%f", &weight[i]);
}
printf("Enter the profits of each object:-\n");
for (i = 0; i < num; i++) {
  scanf("%f", &profit[i]);
}
printf("Enter the capacity of knapsack:-");
scanf("%f", &capacity);
for (i = 0; i < num; i++) {
  ratio[i] = profit[i] / weight[i];
}
for (i = 0; i < num; i++) {
  for (j = i + 1; j < num; j++) {
    if (ratio[i] < ratio[j]) {
      temp = ratio[j];
      ratio[j] = ratio[i];
      ratio[i] = temp;
      temp = weight[j];
      weight[j] = weight[i];
      weight[i] = temp;
      temp = profit[j];
      profit[j] = profit[i];
      profit[i] = temp;
    }
  }
```

```
  }
  knapsack(num, weight, profit, capacity);
  return(0);
}
```

OUTPUT:

```
Enter the no. of objects:-4
Enter the wts of each object:-
3 1 2 4
Enter the profits of each object:-
20 26 22 21
Enter the capacity of knapsack:-8

The result vector is:- 1.000000 1.000000    1.000000    0.500000
Maximum profit is:- 78.500000

=== Code Execution Successful ===
```

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

```c
#include <stdio.h>
#define MAX 9999 // Infinity value assumed
void dijkstras(int c[][100], int n, int src);
int main() {
    int n, src, i, j;
    int c[100][100]; // Assuming a maximum size for the cost matrix
    printf("Enter the number of nodes: ");
    scanf("%d", &n);
    printf("Enter the cost matrix:\n");
    for (i = 1; i <= n; i++) {
        for (j = 1; j <= n; j++) {
            scanf("%d", &c[i][j]);
        }
    }
    printf("Enter the source node (1 to %d): ", n);
    scanf("%d", &src);
    dijkstras(c, n, src);
    return 0;
}
void dijkstras(int c[][100], int n, int src) {
    int dist[100], vis[100];
    int count, min, u, i, j;
    // Initialization
    for (j = 1; j <= n; j++) {
        dist[j] = c[src][j];
        vis[j] = 0;
    }
    dist[src] = 0;
```

```c
        vis[src] = 1;
        count = 1;
        // Main loop
        while (count != n) {
            min = MAX;


            // Find the minimum distance vertex from the set of vertices not yet processed
            for (j = 1; j <= n; j++) {
                if (dist[j] < min && vis[j] != 1) {
                    min = dist[j];
                    u = j;
                }
            }
            vis[u] = 1;
            count++;
            // Update dist value of the adjacent vertices of the picked vertex
            for (j = 1; j <= n; j++) {
                if (min + c[u][j] < dist[j] && vis[j] != 1) {
                    dist[j] = min + c[u][j];
                }
            }
        }


        // Output shortest distances
        printf("Shortest distances from node %d:\n", src);
        for (j = 1; j <= n; j++) {
            printf("Distance to node %d from node %d: %d\n", j, src, dist[j]);
        }
}
```

OUTPUT:

```
Enter the number of nodes: 4
Enter the cost matrix:
0 2 3 4
1 0 3 4
2 1 0 3
5 6 2 0
Enter the source node (1 to 4): 2
Shortest distances from node 2:
Distance to node 1 from node 2: 1
Distance to node 2 from node 2: 0
Distance to node 3 from node 2: 3
Distance to node 4 from node 2: 4


=== Code Execution Successful ===
```

14. Implement "N-Queens Problem" using Backtracking.

```c
#include  <stdio.h>
#include  <stdlib.h>
#define MAX_N 10
int x[MAX_N];
int Place(int k, int i, int n) {
   for (int j = 1; j <= k - 1; j++) {
      if (x[j] == i || abs(x[j] - i) == abs(j - k)) {
         return 0;
      }
   }
   return 1;
}
void NQueens(int k, int n) {
   for (int i = 1; i <= n; i++) {
      if (Place(k, i, n)) {
         x[k] = i;
         if (k == n) {
            printf("Solution: ");
            for (int j = 1; j <= n; j++) {
               printf("%d ", x[j]);
            }
            printf("\n");
         } else {
            NQueens(k + 1, n);
         }
      }
   }
}
```

```
int main() {
    int n;
    printf("Enter the number of queens (n): ");
    scanf("%d", &n);
    for (int i = 0; i <= n; i++) {
        x[i] = 0;
    }
    NQueens(1, n);
    return 0;
}
```

OUTPUT:

```
Enter the number of queens (n): 5
Solution: 1 3 5 2 4
Solution: 1 4 2 5 3
Solution: 2 4 1 3 5
Solution: 2 5 3 1 4
Solution: 3 1 4 2 5
Solution: 3 5 2 4 1
Solution: 4 1 3 5 2
Solution: 4 2 5 3 1
Solution: 5 2 4 1 3
Solution: 5 3 1 4 2


=== Code Execution Successful ===
```