

# Practical File

## Index

| Practical No. | Name of the Practical  | Page No. |
|---------------|--|----------|
| 1             | Write a program to perform Linear Search and also find its complexity in Best, Worst, and Average case.    | 2        |
| 2             | Write a program to perform Binary Search and also find its complexity in Best, Worst, and Average case.    | 4        |
| 3             | Write a program to implement Selection Sort and also find its complexity in Best, Worst, and Average case. | 6        |
| 4             | Write a program to implement Merge Sort and also find its complexity in Best, Worst, and Average case.     | 7        |
| 5             | Write a program to implement Quick Sort and also find its complexity in Best, Worst, and Average case.     | 9        |
| 6             | Write a program to multiply two matrices using Strassen's multiplication algorithm.                        | 11       |

# Practical - 1

**Aim:** Write a program to perform Linear Search and also find its complexity in Best, Worst, and Average case.

## Code

```
1 #include <stdio.h>
2 #include <time.h>
3
4 int main() {
5     int num;
6     int i, key, element_found = 0;
7     printf("Enter number of elements: ");
8     scanf("%d", &num);
9     int arr[num];
10    printf("\nEnter the elements: ");
11    for (i = 0; i < num; i++) {
12        scanf("%d", &arr[i]);
13    }
14
15    printf("\nEnter the element to be searched: ");
16    scanf("%d", &key);
17
18    clock_t start, end;
19    double cpu_time_used;
20    start = clock();
21
22    for (i = 0; i < num; i++) {
23        if (key == arr[i]) {
24            element_found = 1;
25            break;
26        }
27    }
28
29    end = clock();
30    cpu_time_used = ((double)(end - start)) / CLOCKS_PER_SEC;
31
32    if (element_found == 1)
33        printf("Element found at index %d\n", i + 1);
34    else
35        printf("Element not found\n");
36
37    printf("Runtime: %f seconds\n", cpu_time_used);
38    return 0;
39 }
```

## Output

- Input: Number of elements, array elements, and the key to search.
- Output: Whether the element is found (with index) or not, and the runtime.

## Complexity Analysis

- Best Case:  $O(1)$
- Worst Case:  $O(n)$
- Average Case:  $O(n)$

## Practical - 2

**Aim:** Write a program to perform Binary Search and also find its complexity in Best, Worst, and Average case.

### Code

```
1 #include <stdio.h>
2 #include <time.h>
3
4 int main() {
5     int n, i, x;
6     printf("Enter number of elements: ");
7     scanf("%d", &n);
8     int arr[n];
9     printf("Enter %d elements in sorted order: ", n);
10    for (i = 0; i < n; i++) {
11        scanf("%d", &arr[i]);
12    }
13
14    printf("Enter the element to search: ");
15    scanf("%d", &x);
16
17    clock_t start, end;
18    start = clock();
19    int low = 0, high = n - 1, result = -1;
20
21    while (low <= high) {
22        int mid = (low + high) / 2;
23        if (arr[mid] == x) {
24            result = mid;
25            break;
26        } else if (arr[mid] < x) {
27            low = mid + 1;
28        } else {
29            high = mid - 1;
30        }
31    }
32
33    end = clock();
34    double runtime = ((double)(end - start)) / CLOCKS_PER_SEC;
35
36    if (result == -1)
37        printf("Element is not present in array\n");
38    else
39        printf("Element is present at index %d\n", result + 1);
40
41    printf("Runtime: %f seconds\n", runtime);
42    return 0;
43 }
```

## Output

- Input: Number of elements (in sorted order), array elements, and the key to search.
- Output: Whether the element is found (with index) or not, and the runtime.

## Complexity Analysis

- **Best Case:**  $O(1)$
- **Worst Case:**  $O(\log n)$
- **Average Case:**  $O(\log n)$

## Practical - 3

**Aim:** Write a program to implement Selection Sort and also find its complexity in Best, Worst, and Average case.

## Code

```
1 #include<stdio.h>
2 void main(){
3     int n,i,j,temp,minindex,flag=0;
4     printf("enter the number of elements:");
5     scanf("%d",&n);
6     int arr[n];
7     printf("enter the elements:");
8     for(i=0;i<n;i++){
9         scanf("%d",&arr[i]);
10    }
11    for(i=0;i<=n-1;i++){
12        minindex=i;
13        for(j=i+1;j<n;j++){
14            if (arr[j]<arr[minindex]){
15                minindex=j;
16                flag=1;
17            }
18        }
19        if(flag==1){
20            temp=arr[i];
21            arr[i]=arr[minindex];
22            arr[minindex]=temp;
23        }
24    }
25    for(i=0;i<n;i++){
26        printf("%d \t",arr[i]);
27    }
28 }
```

## Output

- Input: Number of elements and array elements.
- Output: Array elements sorted in ascending order.

## Complexity Analysis

- Best Case:  $O(n^2)$
- Worst Case:  $O(n^2)$
- Average Case:  $O(n^2)$

## Practical - 4

**Aim:** Write a program to implement Merge Sort and also find its complexity in Best, Worst, and Average case.

### Code

```
1 #include<stdio.h>
2 void merge(int a[],int low,int mid,int high){
3     int temp[high-low+1],i=0;
4     int left=low,right=mid+1;
5     while(left<=mid && right<=high){
6         if(a[left]<=a[right]){
7             temp[i]=a[left];
8             left++;
9         }
10        else{
11            temp[i]=a[right];
12            right++;
13        }
14        i++;
15    }
16    while(left<=mid){
17        temp[i]=a[left];
18        left++;
19        i++;
20    }
21    while(right<=high){
22        temp[i]=a[right];
23        right++;
24        i++;
25    }
26    for(i=low;i<=high;i++){
27        a[i]=temp[i-low];
28    }
29 }
30 int sort(int a[],int low,int high){
31     if(low<high){
32         int mid=(high+low)/2;
33         sort(a,low,mid);
34         sort(a,mid+1,high);
35         merge(a,low,mid,high);
36     }
37 }
38 int main(){
39     int n;
40     printf("enter number of elements:");
41     scanf("%d",&n);
42     int a[n];
43     printf("enter the elements:");
```

```
44     for(int i=0;i<n;i++){  
45         scanf("%d",&a[i]);  
46     }  
47     sort(a,0,n-1);  
48     for(int i=0;i<n;i++){  
49         printf("%d ",a[i]);  
50     }  
51 }
```

## Output

- Input: Number of elements and array elements.
- Output: Array elements sorted in ascending order.

## Complexity Analysis

- **Best Case:**  $O(n \log n)$
- **Worst Case:**  $O(n \log n)$
- **Average Case:**  $O(n \log n)$



## Practical - 5

**Aim:** Write a program to implement Quick Sort and also find its complexity in Best, Worst, and Average case.

### Code

```
1 #include <stdio.h>
2 #include <stdbool.h>
3
4 int partition(int a[], int low, int high) {
5     int pivot = a[low];
6     int i = low - 1;
7     int j = high + 1;
8     while (true) {
9         do {
10             i++;
11         } while (a[i] < pivot);
12
13         do {
14             j--;
15         } while (a[j] > pivot);
16
17         if (i >= j)
18             return j;
19
20         int temp = a[i];
21         a[i] = a[j];
22         a[j] = temp;
23     }
24 }
25
26 void quicksort(int a[], int low, int high) {
27     if (low < high) {
28         int p = partition(a, low, high);
29         quicksort(a, low, p);
30         quicksort(a, p + 1, high);
31     }
32 }
33
34 int main() {
35     int arr[100], n;
36     printf("Enter the number of elements: ");
37     scanf("%d", &n);
38     printf("Enter the array elements: ");
39     for (int i = 0; i < n; i++) {
40         scanf("%d", &arr[i]);
41     }
42     quicksort(arr, 0, n - 1);
43     printf("Sorted array: ");
```

```
44     for (int i = 0; i < n; i++) {  
45         printf("%d ", arr[i]);  
46     }  
47     printf("\n");  
48     return 0;  
49 }
```

## Output

- Input: Number of elements and array elements.
- Output: Array elements sorted in ascending order.

## Complexity Analysis

- **Best Case:**  $O(n \log n)$
- **Worst Case:**  $O(n^2)$
- **Average Case:**  $O(n \log n)$

## Practical - 6

**Aim:** Write a program to multiply two matrices using Strassen's multiplication algorithm.

### Code

```
1 #include<stdio.h>
2 int main(){
3     int a[2][2], b[2][2], c[2][2], i, j;
4     int m1, m2, m3, m4, m5, m6, m7;
5
6     printf("Enter the 4 elements of first matrix: ");
7     for(i = 0; i < 2; i++)
8         for(j = 0; j < 2; j++)
9             scanf("%d", &a[i][j]);
10
11     printf("Enter the 4 elements of second matrix: ");
12     for(i = 0; i < 2; i++)
13         for(j = 0; j < 2; j++)
14             scanf("%d", &b[i][j]);
15
16     printf("\nThe first matrix is\n");
17     for(i = 0; i < 2; i++){
18         printf("\n");
19         for(j = 0; j < 2; j++)
20             printf("%d\t", a[i][j]);
21     }
22
23     printf("\nThe second matrix is\n");
24     for(i = 0; i < 2; i++){
25         printf("\n");
26         for(j = 0; j < 2; j++)
27             printf("%d\t", b[i][j]);
28     }
29
30     m1= (a[0][0] + a[1][1]) * (b[0][0] + b[1][1]);
31     m2= (a[1][0] + a[1][1]) * b[0][0];
32     m3= a[0][0] * (b[0][1] - b[1][1]);
33     m4= a[1][1] * (b[1][0] - b[0][0]);
34     m5= (a[0][0] + a[0][1]) * b[1][1];
35     m6= (a[1][0] - a[0][0]) * (b[0][0]+b[0][1]);
36     m7= (a[0][1] - a[1][1]) * (b[1][0]+b[1][1]);
37
38     c[0][0] = m1 + m4 - m5 + m7;
39     c[0][1] = m3 + m5;
40     c[1][0] = m2 + m4;
41     c[1][1] = m1 - m2 + m3 + m6;
42
43     printf("\nAfter multiplication using Strassen's algorithm \n");
```

```
44  for(i = 0; i < 2 ; i++){
45      printf("\n");
46      for(j = 0; j < 2; j++)
47          printf("%d\t", c[i][j]);
48  }
49
50  return 0;
51 }
```

## Output

- Input: Elements of two 2x2 matrices.
- Output: Resultant matrix after multiplication using Strassen's algorithm.

## Complexity Analysis

- **Time Complexity:** Approximately  $O(n^{2.81})$  (better than the standard  $O(n^3)$  for matrix multiplication).