ASSIGNMENT

Submitted By

Ashbi Saju

Roll No. 16

1. Compute the complexity of the following snippets of code

a)
```
int i, j, k=0;
for (i = n/2 ; i<=n ; i++) {          — n
    for (j=2 ; j<=n ; j= j*2) {
        k = k + n/2;                  — log2n
    }
}
```

∴ Time complexity = $O(n \log n)$

b)
```
int a=0, i=N;
while (i>0) {
    a+=i;                             — log2n
    i /=2;
}
```

∴ Time complexity = $O(\log n)$

c)
```
void fun (int n) {
    for (int i=0; i<n ; i++) {        — n
        for (j=n ; j<0 ; j = j/2) {   — log2n
            print("*");
        }
    }
}
```

$\therefore$ Time complexity = $O(n \log n)$

2. Solve the recurrence relation:

$T(n) = T(n/2) + \log_2 n$ where $T(1) = 1$

$T(n) = T(n/2) + \log_2 n$

$T(n/2) = T(n/4) + \log(n/2)$

$T(n/4) = T(n/8) + \log(n/4)$

$\therefore T(n) = T(n/2^k) + \sum \log_2 (n/2^i)$

$\log_2 (n/2^i) \Rightarrow \log n - \log_2 2^i$

$\log n - i$

$T(1) = T(n/2^k) \Rightarrow$

$(n/2^k) = 1$

$n = 2^k$

$\log n = k$

$\therefore T(n) = T(1) + \sum (\log n - i)$

$= T(1) + \sum \log n - \sum i$

$= 1 + k \log n - \frac{k(k-1)}{2}$

Since $k = \log n \Rightarrow$

$$T(n) = 1 + \log n \dots \log n - \log n \frac{(\log n - 1)}{2}$$

$$\therefore O(\log^2 n)$$

3. Algorithm A performs $10n^2$ basic operations and B performs $300 \log n$ operations. for what value of n does B starts to show better performance.

Algorithm A $\Rightarrow T_A(n) \Rightarrow 10n^2$

Algorithm B $\Rightarrow T_B(n) \Rightarrow 300 \log n$

$300 \log n < 10n^2$

Dividing by 10, $30 \log n < n^2$

n	$A \Rightarrow n^2$	$B \Rightarrow 30 \log n$
2^2	16	60
2^3	64	90
2^4	256	120

$\therefore$ for all $n > 2^4$, Algorithm B starts to show better performance than A.

4. Sorting Algorithms with Iteration Analysis

4.1 Bubble Sort

Program (C)

```
#include <stdio.h>
void bubbleSort(int arr[], int n, int *outer, int *inner) {
   int i, j, temp;
   for (i = 0; i < n - 1; i++) {
      (*outer)++;
      for (j = 0; j < n - i - 1; j++) {
         (*inner)++;
         if (arr[j] > arr[j + 1]) {
            temp = arr[j];
            arr[j] = arr[j + 1];
            arr[j + 1] = temp;
         }
      }
   }
}

void testBubbleSort(char *caseName, int arr[], int n) {
   int outer = 0, inner = 0;
   bubbleSort(arr, n, &outer, &inner);
   printf("%s Case → Outer: %d, Inner: %d\n", caseName, outer, inner);
}

int main() {
   int n = 5;
   int best[]  = {1, 2, 3, 4, 5};   // Best case
   int avg[]   = {3, 1, 4, 5, 2};   // Average case
```

```
    int worst[] = {5, 4, 3, 2, 1};   // Worst case

    printf("BUBBLE SORT\n");
    testBubbleSort("Best", best, n);
    testBubbleSort("Average", avg, n);
    testBubbleSort("Worst", worst, n);

    return 0;
}
```

Output

BUBBLE SORT

Best Case → Outer: 4, Inner: 10

Average Case → Outer: 4, Inner: 10

Worst Case → Outer: 4, Inner: 10

Iteration Count

Case	Outer Loop	Inner Loop	Total
Best	4	10	14
Average	4	10	14
Worst	4	10	14

Time Complexity

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

4.2 Selection Sort

Program (C)

```
#include <stdio.h>
void selectionSort(int arr[], int n, int *outer, int *inner) {
   int i, j, min, temp;
   for (i = 0; i < n - 1; i++) {
      (*outer)++;
      min = i;
      for (j = i + 1; j < n; j++) {
         (*inner)++;
         if (arr[j] < arr[min])
            min = j;
      }
      temp = arr[i];
      arr[i] = arr[min];
      arr[min] = temp;
   }
}

void testSelectionSort(char *caseName, int arr[], int n) {
   int outer = 0, inner = 0;
   selectionSort(arr, n, &outer, &inner);
   printf("%s Case → Outer: %d, Inner: %d\n", caseName, outer, inner);
}

int main() {
   int n = 5;
   int best[]  = {1, 2, 3, 4, 5};
   int avg[]   = {3, 1, 4, 5, 2};
   int worst[] = {5, 4, 3, 2, 1};
```

```
    printf("SELECTION SORT\n");
    testSelectionSort("Best", best, n);
    testSelectionSort("Average", avg, n);
    testSelectionSort("Worst", worst, n);

    return 0;
}
```

Output

SELECTION SORT

Best Case → Outer: 4, Inner: 10

Average Case → Outer: 4, Inner: 10

Worst Case → Outer: 4, Inner: 10

Iteration Count

Case	Outer Loop	Inner Loop	Total
Best	4	10	14
Average	4	10	14
Worst	4	10	14

Time Complexity

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

4.3 Insertion Sort

Program (C)

```
#include <stdio.h>
void insertionSort(int arr[], int n, int *outer, int *inner) {
   int i, j, key;
   for (i = 1; i < n; i++) {
      (*outer)++;
      key = arr[i];
      j = i - 1;
      while (j >= 0 && arr[j] > key) {
         (*inner)++;
         arr[j + 1] = arr[j];
         j--;
      }
      arr[j + 1] = key;
   }
}

void testInsertionSort(char *caseName, int arr[], int n) {
   int outer = 0, inner = 0;
   insertionSort(arr, n, &outer, &inner);
   printf("%s Case → Outer: %d, Inner: %d\n", caseName, outer, inner);
}

int main() {
   int n = 5;
   int best[]  = {1, 2, 3, 4, 5};
   int avg[]   = {3, 1, 4, 5, 2};
   int worst[] = {5, 4, 3, 2, 1};
   printf("INSERTION SORT\n");
   testInsertionSort("Best", best, n);
```

```
    testInsertionSort("Average", avg, n);
    testInsertionSort("Worst", worst, n);

    return 0;
}
```

Output

INSERTION SORT

Best Case → Outer: 4, Inner: 0

Average Case → Outer: 4, Inner: 4

Worst Case → Outer: 4, Inner: 10

Iteration Count

Case	Outer Loop	Inner Loop	Total
Best	4	0	4
Average	4	6	10
Worst	4	10	14

Time Complexity

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

Comparative Summary Table

Algorithm	Best Case	Average Case	Worst Case
Bubble Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Insertion Sort	O(n)	$O(n^2)$	$O(n^2)$