Johnny Nguyen – ECGR 3180

Answer = **Bolded**

Q1. For each of the following 6 program fragments, give a Big-Oh analysis of the running time (3 points) –

(1)

```
sum = 0 ;  // O(1)
f o r ( i = 0 ; i < n ; i++ )  // O(n)
        ++sum ;   // O(1)
```

Ans: **O(n)**

(2)

```
sum = 0 ;
f o r ( i = 0 ; i < n ; i++ )
        f o r ( j = 0 ; j < n ; j++)
                ++sum ;
```

Ans: **O(n^2)**

(3)

```
sum = 0 ;
f o r ( i = 0 ; i < n ; i++ )
        f o r ( j = 0 ; j < n*m ; j++)  // n * m → O(n^2)
                ++sum ;
```

Ans: **O(n^3)**

(4)

```
sum = 0 ;
f o r ( i = 0 ; i < n ; i++ )
        f o r ( j = 0 ; j < i ; j++)
                ++sum ;
```

Ans: **O(n^2)**

(5)

```
sum = 0 ;
f o r ( i = 0 ; i < n ; i++ )
        f o r ( j = 0 ; j < i*i ; j++)  // O(n^2)
                for (k = 0; k < j; k++)
                        ++sum;
```

Ans: **O(n^4)**

(6)

```
sum = 0 ;
```

```
for ( i = 0 ; i < n ; i++ )
    for ( j = 0 ; j < i*i ; j++)
        if (j % i == 0)  // If necessary to have multiple execution. → O(n)
            for (k = 0; k < j; k++)
                ++sum;
```

Ans: **O(n^4), if the if statement executes.**

Q2. Programs A and B are analyzed and found to have worst-case running times no greater than $150N\log_2 N$ and N^2 , respectively. Answer the following questions (3 points) -

a. Which program has the better guarantee on the running time for large values of N (N > 10,000)?

Ans: Ex. N = 2^14 = 16,384

$150N\log_2 N$ = 150 * 2^14 * 14 = 34,406,400

N^2 = 268,435,456

$150N\log_2 N$ < N^2 → **$150N\log_2 N$ is optimal.**

b. Which program has the better guarantee on the running time for small values of N (N < 100)?

Ans: Ex. N = 2^7 = 128

$150N\log_2 N$ = 150* 2^7 * 7 = 134400

N^2 = 16384

N^2 < $150N\log_2 N$ → **N^2 is more optimal.**

c. Which program will run faster on average for N = 1000?

Ans: $150N\log_2 N$ = 1,494,868

N^2 = 1,000,000

N^2 < $150N\log_2 Nv$ → **N^2 is more optimal.**

Q3. Q3. Solve the following recurrence relations using the Master theorem (2 points) -

a. T(n) = 3T(n/2) + n/2

b = 2, a = 3, d = 1 → 3 > 2^1 (Case 3)

Ans: O(n) = **O(n^($\log_2 3$))**

b. T(n) = 4T(n/2) + $n^{2.5}$

b = 2, a = 4, d = 2.5 → 4 < 2^2.5 (Case 2)

Ans: O(n) = **O(n^2.5)**

Q4. Analyze the run time complexity of the following algorithms (2 points)

a. Given an array (or string), the task is to reverse the array/string.

Algorithm -

1) Initialize start and end indexes as start = 0, end = n-1

2) In a loop, swap arr[start] with arr[end] and change start and end as follows : start = start +1, end = end – 1 3)

Repeat 2) while start < end

Ans: **O(n) because it will loop through indexes once.**

Q5. Given an array A[], the task is to segregate even and odd numbers. All even numbers should appear first, followed by odd numbers.

Algorithm -

1) Initialize two index variables left and right: left = 0, right = size -1
2) Keep incrementing left index until we see an odd number.
3) Keep decrementing right index until we see an even number.
4) Swap arr[left] and arr[right]
5) Repeat 2 - 4 while left < right

Ans: **O(n^2) loop through indexes twice.**