GIT Department of Computer Engineering
CSE 222/505 - Spring 2022
Homework 2 Report

Muhammed Sinan Pehlivanoğlu
1901042664

Q1

1-)

a-) $\log_2 n^2 + 1 = O(n)$

Def: $f(n) = O(g(n))$ $\quad f(n) \leq c\,g(n)$ $\quad \forall n \geq n_0$.

$\log_2 n^2 + 1 = cn \quad \rightarrow \quad n^2 \leq 2^{cn-1}$ $\quad$ for $c=1$ and $n_0 = 7$ $\forall n \geq 7$ it is true.

b-) $\sqrt{n.(n+1)} = \Omega(n)$

Def: $f(n) = \Omega(g(n))$ $\quad f(n) \geq c\,g(n)$ $\quad \forall n \geq n_0$

$n^2 + n \geq cn^2$ $\quad$ for $c = 1$ and $n_0 = 0$

$\left(\sqrt{n.n+1}\right)^2 \geq (cn)^2$ $\quad$ so $\forall n > 0$ it is true.

c-) $n^{n-1} = \Theta(n^n)$

Def: $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ and $\forall n \geq n_0$ it must be true.

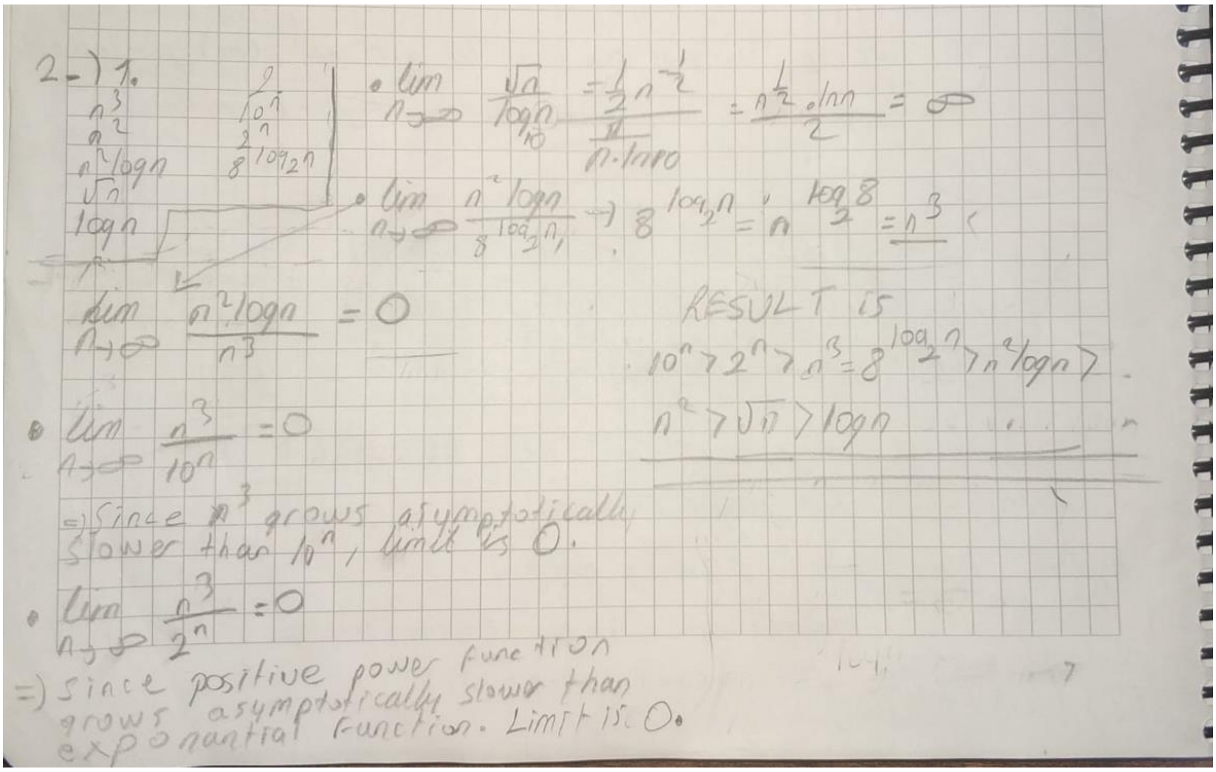
$c_1 n^n \leq \frac{n^n}{n} \leq c_2 n^n$, $\quad c_1$ and $c_2 \neq 0$

$c_1 \cancel{n^n} \leq \frac{\cancel{n^n}}{n} \rightarrow c_1 \leq \frac{1}{n}$

$\frac{\cancel{n^n}}{n} \leq c_2 \cancel{n^n} \rightarrow \frac{1}{n} \leq c_2$

There is no c_1, c_2 and n_0 value providing above definition. So it is false.

Q2



2-) 1.

n^3 | 10^n
n^2 | 2^n
$n^2 \log n$ | $8^{\log_2 n}$
$\sqrt{n}$
$\log n$

$\lim_{n \to \infty} \frac{\sqrt{n}}{\log_{10} n} = \frac{\frac{1}{2} n^{-\frac{1}{2}}}{\frac{1}{n \cdot \ln 10}} = \frac{n^{\frac{1}{2}} \cdot \ln n}{2} = \infty$

$\lim_{n \to \infty} \frac{n^2 \log n}{8^{\log_2 n}} \to 8^{\log_2 n} = n^{\log_2 8} = n^3$

$\lim_{n \to \infty} \frac{n^2 \log n}{n^3} = 0$

RESULT is

$10^n > 2^n > n^3 = 8^{\log_2 n} > n^2 \log n >$
$n^2 > \sqrt{n} > \log n$

$\lim_{n \to \infty} \frac{n^3}{10^n} = 0$

=) Since n^3 grows asymptotically slower than 10^n, limit is 0.

$\lim_{n \to \infty} \frac{n^3}{2^n} = 0$

=) Since positive power function grows asymptotically slower than exponantial function. Limit is 0.

Q3

a-)

```
int p_1 ( int my_array[]){
    for(int i=2; i<=n; i++){
        if(i%2==0){
            count++;
        } else{
            i=(i-1)i;
        }
    }
}
```

I could not find.

b-)

```
int p_2 (int my_array[]){
    first_element = my_array[0];
    second_element = my_array[0];
    for(int i=0; i<sizeofArray; i++){
        if(my_array[i]<first_element){
            second_element=first_element;
            first_element=my_array[i];}
        else if(my_array[i]<second_element){
            if(my_array[i]!= first_element){
                second_element= my_array[i];
            }
        }
    }
}
```

1.Line is Θ(1)

2.Line is Θ(1)

Best case and worst case is equal in if statements. İt is Θ(1).

For loop is executed n times. İt is Θ(n).

So T(n) = Θ(n)* Θ(1) = Θ(n).

c-)

```
int p_3 (int array[]) {
    return array[0] * array[2];
}
```

The time complexitiy is constant Θ(1).

d-)

```c
int p_4(int array[], int n) {
    Int sum = 0
    for (int i = 0; i < n; i=i+5)
        sum += array[i] * array[i];
    return sum;
}
```

For loop works n/5 times. İts complexity is Θ(n/5). We can ignore multiplication with constant value. So İts complexity is Θ(n).

Lines

1. $\Phi(1)$
2. for loop $\Phi(\frac{n}{5})$ $\Big\}$ $\Phi(n)$ $\Rightarrow$ $T(n) = \Phi(1) + \Phi(n) + \Phi(1) = \underline{\Phi(n)}$
3. $\Phi(1)$
4. $\Phi(1)$

e-)

```c
void p_5 (int array[], int n){
    for (int i = 0; i < n; i++)
        for (int j = 1; j < i; j=j*2)
            printf("%d", array[i] * array[j]);
}
```

- Inner loop's iteration number is $O(\log n)$.

For(int j=0 ; j<i ; j=j*2)

j increase exponantional so $O(\log_2 n)$

Printf $= \Phi(1)$, first loop $= \Phi(n)$

$T(n) = \Phi(1) + \Phi(n) * O(\log_2 n) = O(n \log_2 n)$

f-)

```
int p_6(int array[], int n) {
    If(p_4(array, n)) > 1000)
        p_5(array, n)
    else printf("%d", p_3(array) * p_4(array, n))
}
```

- Worst case $= \Phi(n)$ since p-3 function has $\Phi(1)$

p-4 function has $\Phi(n)$

$T_w(n) = \Phi(1) + \Phi(n) + \underbrace{\Phi(1)}_{print} = \Phi(n)$

- Best Case :

p-4 function $= \Phi(n)$

p-5 function $= O(n \log_2 n)$

$T_b(n) = \Phi(n) + O(n \log_2 n) = O(n \log_2$

Best case time Complexity is Θ(nlogn).

g-)

```java
int p_7( int n ){
    int i = n;
    while (i > 0) {
        for (int j = 0; j < n; j++)
            System.out.println("*");
        i = i / 2;
    }
}
```

- while loop works $\Phi(\log_2 n)$
- for loop works $\Phi(n)$ times
- $i = i/2 = \Phi(1)$
- $\text{int } i = n = \Phi(1)$

$$T(n) = \Phi(\log_2 n) \cdot \Phi(n) + \Phi(1) + \Phi(1) = \underline{\Phi(n \log_2 n)}$$

h-)

```java
int p_8( int n ){
    while (n > 0) {
        for (int j = 0; j < n; j++)
            System.out.println("*");
        n = n / 2;
    }
}
```

Geometric Sum formula

$$a \cdot \frac{1-r^n}{1-r}$$

a: first element
r = ratio
n = number of terms.

• In while loop, for loop iteration number is $\log_2 n$. And n is divided by 2 every while loop execution. Total iteration number of for loop calculated by this formula.

Ex: $n = 16$, for loop iteration ~~number~~ → 16 8 4 2 1 0

So first element = n, ratio is $\frac{1}{2}$, number of term $(\log_2 n) + 1 = \log_2 2n$

$$T(n) = O\left(\frac{n \cdot \left(1 - \frac{1}{2}^{\log_2 2n}\right)}{1 - \frac{1}{2}}\right)$$

$$\to \frac{1}{2}^{\log_2 2n} = 2n \to T(n) = O\left(n \cdot (1+2n) \cdot 2\right) = O(n^2) = \text{Worst}$$

$\text{Best} = \Phi(1)$

i)

```
int p_9(n){
    if (n = 0)
        return 1
    else
        return n * p_9(n-1)
}
```

$$T(n) = \begin{cases} 1 & n=0 \\ T(n-1)+1 & n>0 \end{cases}$$

$T(n) = T(n-k) + k$

for $n=k$

$T(n) = \underbrace{T(0)}_{1} + n$

$T(n) = 1 + n = O(n)$

j-)

```
int p_10 (int A[ ], int n) {
    if (n == 1)
        return;
    p_10 (A, n - 1);
    j = n - 1;
    while (j > 0 and A[j] < A[j - 1]) {
        SWAP(A[j], A[j - 1]);
        j = j - 1;
    }
}
```

- $T(n) \quad 1 \neq n = 1$

  $1 + T(n-1) + 1 + n \quad n > 1$

$T(n) = T(n-k) + 2k + nk$

$n - k = 1$
$k = n - 1$

$T(n) = \underbrace{T(1)}_{1} + 2n - 2 + n^2 - n$

$T(n) = n^2 + n - 1 = O(n^2)$

Q4

4-)

a-) Big O notation is used to determine upper bound of running time of algorithm. So statement is false.

b-)

- $F(n) = 2^n \cdot 2$

$c_1 2^n \leq 2^n \cdot 2 \leq c_2 2^n$

$\left.\begin{array}{l} c_1 = 1 \\ c_2 = 2 \\ n_0 = 0 \end{array}\right\}$ are provading the theata notation.

$\forall n \geq 0$ it is true.

- $F(n) = 2^{2n}$

$c_1 2^n \leq 2^{2n} \leq c_2 2^n$

$\left.\begin{array}{l} c_1 = 1 \\ c_2 = 1 \\ n_0 = 1 \end{array}\right\}$ are provading the theta notation.

$\forall n \geq 1$ it is true.

c-) $f(n) = O(n^2)$

$g(n) = \Theta(n^2)$

$\Rightarrow f(n) \leq c_1 n^2$

$c_2 n^2 \leq f(n) \leq c_3 n^2$

We can set the upper bound of running time.

$f \cdot g \leq c_1 \cdot c_3 n^4$

but we can not determine lower bound of running time.

We can say that

$\Theta(n^2) \cdot O(n^2) \subset O(n^4)$

not $\Theta(n^4)$

Q5

$T(n) = \Theta(n) ? \Theta(1) = \Theta(n)$

5-)

- $T(n) = 2T(n/2) + n$

$T(\frac{n}{2}) = 2T(\frac{n}{4}) + \frac{n}{2}$

$\rightarrow T(n) = 2(2T(\frac{n}{4}) + \frac{n}{2}) + n = 2^2 \cdot T(\frac{n}{4}) + \underbrace{n+n}_{2n}$

$T(\frac{n}{4}) = 2T(\frac{n}{8}) + \frac{n}{4}$

$\rightarrow T(n) = 2(2(2T(\frac{n}{8}) + \frac{n}{4}) + \frac{n}{2}) + n = 2^3 T(\frac{n}{8}) + \underbrace{n+n+n}_{3n}$

So the formula is $= 2^k \cdot T(\frac{n}{2^k}) + kn$

to get $T(1) = 1$, $\frac{n}{2^k} = 1 \rightarrow n = 2^k \rightarrow k = \log_2 n$

formula is $\rightarrow \underbrace{2^{\log_2 n}}_{n} \cdot \underbrace{T(1)}_{1} + \log_2 n \cdot n$

$n \cdot 1 + \log_2 n \cdot n = \log_2 n \cdot n = O(\log_2 n \cdot n)$

- $T(n) = 2T(n-1) + 1$, $T(0) = 0$

$T(n) = 2T(n-1) + 1$

$T(n-1) = 2T(n-2) + 1$

$T(n-2) = 2T(n-3) + 1$

$\rightarrow T(n) = 2(2T(n-2) + 1) + 1 = 2^2 T(n-2) + 2^1 + 2^0$

$\rightarrow T(n) = 2 \cdot (2 \cdot (2T(n-3) + 1) + 1) + 1 = 2^3 T(n-3) + 2^2 + 2^1 + 2^0$

So the formula is

$2^k \cdot T(n-k) + 2^{k-1} + 2^{k-2} + \dots + 1$

$2^k \cdot T(n-k) + 2^k - 1$

for $n = k$, formula is $\rightarrow 2^n \cdot \underbrace{T(0)}_{0} + 2^n - 1 = 2^n - 1 = O(2^n)$

Q6

```java
static void findSum(int array[], int sum) {
    for (int i = 0; i < array.length; ++i) {
        for (int j = i; j < array.length; ++j) {
            if(array[i] + array[j] == sum){

            }

        }
    }
}
```

6.

for i → array.length

for j = i → array.length (++j)

if(array[i] + array[j] = sum) $\Phi(1)$ $\Big\}$ $\Phi\left(\frac{n \cdot n+1}{2}\right) \cdot \Phi(1))$

$= \Phi(n^2)$

```
Iteration
First Array with 10 element : 0.0074 ns
Second Array with 100 element : 0.093 ns
Third Array with 1000 element : 3.2727 ns
```

Q7

```java
static void findSum(int array[],int sum, int n){
    if(n<0){
        return;
    }
    findSum(array,sum , n: n-1);
    for(int i =n ; i<array.length; ++i){
        if(array[n] + array[i] == sum){

        }
    }
}
```

7. if (n<0)
   return

findsum(n-1)
for i=n → array.length
  if (array [n] + array [i]) = sum

$$T(n) = \begin{cases} 1 & n<0 \\ 1+T(n-1)+n & n\geq 0 \end{cases}$$

$T(n) = T(n-1) + 1 + n$

...

$T(n-k+1) = T(n-k) + 1 + n$

$T(n) = T(n-k) + k + kn$

$n = k$

$T(n) = \underbrace{T(0)}_{1} + n + n^2 = O(n^2)$

$T(n) = O(n^2)$

```
Recursive
First Array with 10 element : 0.0061 ns
Second Array with 100 element : 0.1776 ns
Third Array with 1000 element : 2.2368 ns
```

Muhammed Sinan Pehlivanoğlu

1901042664

Computer Engineering