GTU Department of Computer Engineering
CSE 222/505- Spring 2022
HOMEWORK 2

Şiyar Tarık Özcaner 200104004021

1)

1) For each of the following statements, specify whether it is true or not, and prove your claim. Use the definition of asymptotic notations.

a) $\log_2 n^2 + 1 = O(n)$ $\quad 2\log_2 n + 1 \leq cn \quad n \geq 1$ and $c = 2$ this holds so True

b) $\sqrt{n(n+1)} = \Omega(n)$ $\quad \sqrt{n^2+n} \geq cn \quad n \geq 1$ and $c = 1$ holds for $\forall n$ so True

c) $n^{n-1} = \theta(n^n)$; $n^{n-1} \leq cn^n \quad c = 1 \quad n \geq 1$ holds FALSE

; $n^{n-1} \geq cn^n$ there is no appropriate c value so doesn't hold.

2) Order the following functions by growth rate and explain your reasoning for each of them. Use the

2)

2) $\lim_{n\to\infty} \frac{\log n}{\sqrt{n}} = 0 \Rightarrow \log n = o(\sqrt{n})$

$\lim_{n\to\infty} \frac{\sqrt{n}}{n^2} = 0 \Rightarrow \sqrt{n} = o(n^2)$

$\lim_{n\to\infty} \frac{n^2}{n^2 \log n} = 0 \rightarrow n^2 = o(n^2 \log n)$

$\lim_{n\to\infty} \frac{n^2 \log n}{n^3} = 0 \rightarrow n^2 \log n = o(n^3)$

$\lim_{n\to\infty} \frac{n^3}{2^n} = 0 \rightarrow n^3 = o(2^n)$

$\lim_{n\to 8} \frac{2^n}{10^n} = 0 \rightarrow 2^n = o(10^n)$

$n^{\log_2 8} \rightarrow n^3$

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

growth rate comparison

3)

a)

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

O(log n)

b)

```
int p_2 (int my_array[]){
        first_element = my_array[0]; Θ(1)
        second_element = my_array[0]; Θ(1)
        for(int i=0; i<sizeofArray; i++){ Θ(n)
                if(my_array[i]<first_element){ Θ(1)
                        second_element=first_element; Θ(1)
                        first_element=my_array[i]; Θ(1)
                }else if(my_array[i]<second_element){ Θ(1)
                        if(my_array[i]!= first_element){ Θ(1)
                                second_element= my_array[i]; Θ(1)
                        }
                }
}
```

Θ(n)

c)

$\Theta(1)$

```
int p_3 (int array[]) {
        return array[0] * array[2]; Θ(1)
}
```

d)

$\Theta(n)$

```
int p_4(int array[], int n) {
        Int sum = 0  Θ(1)
        for (int i = 0; i < n; i=i+5)  Θ(n)
                sum += array[i] * array[i]; Θ(1)
        return sum; Θ(1)
}
```

e)

$\Theta(n\log n)$

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

f)

Best: $\Theta(n)$
Worse: $\Theta(n\log n)$
Avg: $O(n\log n)$

```
int p_6(int array[], int n) {
        If (p_4(array, n)) > 1000) Θ(n)+Θ(1) = Θ(n)
                p_5(array, n)  Θ(nlogn)
        else printf("%d", p_3(array) * p_4(array, n))  Θ(n) + Θ(1) = Θ(n)
}
```

g)

```
int p_7( int n ){
        int i = n;  Θ(1)
        while (i > 0) {  Θ(logn)
                for (int j = 0; j < n; j++)  Θ(n)
                        System.out.println("*");  Θ(n)
                i = i / 2;
        }
}
```

$\Theta(\log n) \cdot \Theta(n) = \Theta(n \log n)$

h)

```
int p_8( int n ){
        while (n > 0) {  Θ(log n)
                for (int j = 0; j < n; j++)  Θ(logn)
                        System.out.println("*");  Θ(1)
                n = n / 2;
        }
}
```

$\Theta(\log n) \cdot \Theta(\log n) = \Theta(\log^2 n)$

Runs for n times.

i)

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

Best: $\Theta(1)$

Worst: $\Theta(n)$

Average: $O(n)$

j)

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

Best: $\Theta(1)$

Worst: $\Theta(n^2)$

Average: $O(n^2)$

4)

4)

a) Explain what is wrong with the following statement. "The running time of algorithm A is at least $O(n^2)$". big-O notation informs us about upper bounds so $O(n^2)$ doesn't give any info. on "lowest running time".

b) Prove that clause true or false? Use the definition of asymptotic notations.

I. $2^{n+1} = \Theta$[redacted](2^n) $\quad 2^{n+1} = c \cdot 2^n \quad c=2 \quad n \geq 1$ holds (True)

II. $2^{2n} = \Theta$[redacted](2^n) $\quad 2^{2n} = c\, 2^n$ there is no appropriate c for this equation so doesn't hold. (FALSE)

III. Let $f(n)=O(n^2)$ and $g(n)=\Theta(n^2)$. Prove or disprove that: $f(n) * g(n) = \Theta(n^4)$

a possible $f(n)$ is n for $g(n)$ $2n^2$, in this case it is obvious for $f(n) \cdot g(n) = n \cdot 2n^2 = 2n^3 \neq \Theta(n^4)$ doesn't hold hence disproval.

5) Solve the following recurrence relations. Express the result in most appropriate asymptotic

5.a)

$$T(n) = 2T(n/2) + n \Rightarrow = 2\left(2T(n/4) + \frac{n}{2}\right) + n$$

$$T\left(\frac{n}{2}\right) = 2T(n/4) + n/2 \qquad = 4\,T(n/4) + n + n$$

$$T\left(\frac{n}{4}\right) = 2T(n/8) + n/4 \qquad = 4\left(2T(n/8) + \frac{n}{4}\right) + 2n$$

$$= 2^3\,T\left(\frac{n}{2^3}\right) + 3n$$

So for kth step

$$T(n) = 2^k\,T(n/2^k) + kn$$

assume is $T(1)$

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

$$n/2^k = 1$$

$$n = 2^k$$

$$\log_2 n = k$$

$$T(n) = 2^{\log_2 n} \cdot T(1) + n\log_2 n$$

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

$$\boxed{O(n\log n)}$$

5.b)

$$T(n) = 2T(n-1)+1 \longrightarrow = 2\left(2T(n-2)+1\right)+1 = 2^2 T(n-2)+2+1 \quad ②$$

$$T(n-1) = 2T(n-2)+1 \qquad ③$$

$$= 2\left[2\left(2T(n-3)+1\right)+1\right]+1 = 2^3 T(n-3)+4+2+1$$

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

$$\vdots$$

for kth element

$$T(n) = 2^k \cdot T(n-k) + 2^{k-1} + 2^{k-2} + 2^{k+3} \ldots + 2 + 1$$

$$T(n-k) = 0$$

$$\hookrightarrow n = k$$

$$= 2^n \cdot 0 + 2^n - 1$$

$$T(n) = 2^n - 1 \qquad \Theta(2^n)$$

6)

```
for (int i = 0; i < N; ++i) Θ(n)
    for (int j = i + 1; j < N; ++j) Θ(n)
        if (array[i] + array[j] == find) Θ(1)
            ++count; Θ(1)
```

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

```
bombom65@DESKTOP-HA617JD:/mnt/d/dersler/222/hw02$ gcc array.c
bombom65@DESKTOP-HA617JD:/mnt/d/dersler/222/hw02$ ./a.out
element number: 500 time: 0.000408
bombom65@DESKTOP-HA617JD:/mnt/d/dersler/222/hw02$ gcc array.c
bombom65@DESKTOP-HA617JD:/mnt/d/dersler/222/hw02$ ./a.out
element number: 5000 time: 0.042628
bombom65@DESKTOP-HA617JD:/mnt/d/dersler/222/hw02$ gcc array.c
bombom65@DESKTOP-HA617JD:/mnt/d/dersler/222/hw02$ ./a.out
element number: 50000 time: 3.968978
```

Results as expected give shows us the time complexity of n^2

7)

```c
int rec(int array[], int find, int element, int inner){

    if (element >= N)
        return 0;

    if (inner >= N)
        return rec(array, find, element + 1, element + 2);

    if (array[element] + array[inner] == find)
        return 1 + rec(array, find, element, inner + 1);

    return rec(array, find, element, inner + 1);
}
```

```
bombom65@DESKTOP-HA617JD:/mnt/d/dersler/222/hw02$ gcc rec.c
bombom65@DESKTOP-HA617JD:/mnt/d/dersler/222/hw02$ ./a.out
element number: 5 time: 0.000001
bombom65@DESKTOP-HA617JD:/mnt/d/dersler/222/hw02$ gcc rec.c
bombom65@DESKTOP-HA617JD:/mnt/d/dersler/222/hw02$ ./a.out
element number: 50 time: 0.000039
bombom65@DESKTOP-HA617JD:/mnt/d/dersler/222/hw02$ gcc rec.c
bombom65@DESKTOP-HA617JD:/mnt/d/dersler/222/hw02$ ./a.out
element number: 500 time: 0.004456
```

Roughly the same complexity as before, but notably with more elapsed time.