2022

GTU Department of Computer Engineering

CSE 222/505

HOMEWORK 2

Abdurrahman BULUT

1901042258

GEBZE TEKNİK ÜNİVERSİTESİ

Q1) For each of the following statements, specify whether it is true or not, and prove your claim. Use the definition of asymptotic notations.

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

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

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

a

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

$\log_2 n^2 + 1 \leq c.n$

$\log_2 n^2 \leq c.n - 1$

$2.\log_2 n^2 \leq c.n - 1$

$\log_2 n \leq \frac{c.n-1}{2}$

$n \leq 2^{\frac{c.n-1}{2}}$, the definition of big-oh holds for $c = 3$ and $n_0 = 1$, $n \geq n_0$

it is TRUE

b

$\sqrt{n.(n+1)} \geq c.n$

$n^2 + n \geq c^2.n^2$

$(c^2-1).n^2 \leq n$, $c = 1$, $n_0 = 1$
$N > n_0$

≤ 1 ✓

the definition of Omega notation holds for $c = 1$ and $n_0 = 1$, so its True.
$n_0 \geq 1$ also True.

$\sqrt{n^2 + \cancel{n}}$ → it dosnt matter actually.

$= n$ //

C

Theta Notation

$T(N) = O(h(N))$ and $T(N) = \Omega(h(N))$

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

$n^{n-1} \leq cn^n$ and $n^{n-1} \geq cn^n$

$\frac{n^n}{n} \leq cn^n$ $\quad\quad$ $\frac{n^n}{n} \geq cn^n$

$\frac{1}{n} \leq c$ $\quad\quad$ $\frac{1}{n} \geq c$

for $c=1$, $n_0=1$ $\forall N \geq n_0$.

for $c=1$, $n_0=1$ Not $\forall N \geq n_0$ $\longrightarrow$ true but Not all $N > n_0$

So, it is <u>False</u>

Q2)

Limit Method

$$\lim_{N\to\infty} \frac{f(N)}{g(N)} = \begin{cases} \text{if } 0 \Rightarrow f(N) \text{ slower} \\ \text{if } c \neq 0 \Rightarrow \text{same} \\ \text{if } \infty \Rightarrow f(n) \text{ faster} \end{cases}$$

$\lim_{N\to\infty} \frac{n^2}{n^3} = 0$, so $\boxed{n^2 < n^3}$

$\lim_{N\to\infty} \frac{n^2}{n^2 \log n} = \lim \frac{1}{\log N}$, so $\boxed{n^2 < n^2 \log n}$

$\lim_{N\to\infty} \frac{n^2}{\sqrt{n}} = \lim_{N\to\infty} n^{3/2} = \infty$, so $\boxed{n^2 > \sqrt{n}}$

$\lim_{N\to\infty} \frac{n^2}{\log n} = \lim_{N\to\infty} \frac{2n}{1/n} = \lim_{n\to\infty} 2n^2 = \infty$, so $\boxed{n^2 > \log n}$

$\lim_{N\to\infty} \frac{n^2}{10^n}$, n^2 grows asymptotically slower, $= 0$, so $\boxed{10^n > n^2}$

$\lim_{N\to\infty} \frac{n^2}{2^n}$, same result, $= 0$, so $\boxed{2^n > n^2}$

b) $\checkmark$

$\lim_{n\to\infty} \frac{n^2}{10^n} = \frac{\infty}{\infty} \Rightarrow$ Lhospital $\Rightarrow \lim_{n\to\infty} \frac{2n}{10^n . \ln 10} = \frac{\infty}{\infty} \Rightarrow$ Lhospital

$\lim_{n\to\infty} \frac{2n}{10^n . \ln 10} = \lim_{n\to\infty} \frac{2}{10^n . \ln^2 10} = 0$ //

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

$\Rightarrow \lim_{N\to\infty} \frac{n^2}{n^3} = \lim_{N\to\infty} \frac{1}{n} = 0$, so $\boxed{n^2 < 8^{\log_2 n}}$

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

$\lim_{N\to\infty} \frac{(\log n)'}{(\sqrt{n})'} = \lim_{n\to\infty} \frac{1/n}{\frac{1}{2\sqrt{n}}} = \lim_{n\to\infty} \frac{2\sqrt{n}}{n} = \lim_{n\to\infty} \frac{2}{\sqrt{n}} = 0$, so $\boxed{\sqrt{n} > \log n}$

$\Rightarrow \underline{\log n < \sqrt{n} < n^2 < n^3}, n^2 \log n, 10^n, 2^n, 8^{\log_2 n}$

$\lim_{n\to\infty} \frac{n^3}{n^2\log n} = \lim_{n\to\infty} \frac{n}{\log n} = \lim_{n\to\infty} \frac{1}{1/n} = \lim_{n\to\infty} n = \infty$, so $n^3 > n^2\log n$

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

$\lim_{n\to\infty} \frac{n^3}{10^n} = 0$, so $n^3 < 10^n$ Same result with $n^2/10^n$ calculations.

Same way, $n^3 < 2^n$ also

$8^{\log_2 n} = 2^{\log_2 n^3} = n^3$

So,

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

$\lim_{n\to\infty} \frac{10^n}{2^n}$ = obviously ∞, so $10^n > 2^n$

Final result:

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

Q3) What is the time complexity of the following programs?

a and b:

```
int p_1(int myArray[]){
    for(int i=2; i<=n; i++){ → O(√n)
        if(i%2==0){ — 1
            count++; — 1
        }else{
            i=(i-1).i; — 1 → Since, i will be even
                             number every time, i+1
                             should be odd, So
                             'if' part works just once time.
        }
    }
}
```

i
2
3 → k.(i-1)+1
7 → k.(i-1)+1
43
⋮
k.(k-1)+1

⇒ Terminate:

$i > n$

$i = k.(k-1)+1$

$k.(k-1)+1 > n$

Constant

$k^2 > n$

$k^2 = n$

$k = \sqrt{n}$

$O(\sqrt{n})$


```
int p_2 (int my-array[]) {
    first_element = my-array[0];  → O(1)  } → O(1)
    second-element = my-array[0];  → O(1) }
    for (int i=0; i<sizeOfArray; i++){ → O(n) → O(n)
        if (my-Array[i]< first_element){ → O(1)
            second_element=first element;  } O(1)
            first_element= my-Array[i];    }
        }elseif (my-array[i] < second_element){ → O(1)
            if(my-array[i] != first_element){ → O(1)     } O(1)
                second_element = my-array[i]; → O(1)     }
            }
        }
}
```

$O(1).P(T) + O(1).P(F) + O(1) = O(1)$

$T(n) = O(n)$

c, d and e:

c) int p_3 (int array[]){
    return array[0]* array[2]; } $\Theta(1) = O(1)$
}

$T(n) = \Theta(1) = O(1) =$ constant time , most appropriate $= \underline{\Theta(1)}$

d) int p_4 (int array[], int n){
    int sum=0 } $\Theta(1)$
    for(int i=0; i<n; i=i+5) $\longrightarrow \Theta(\frac{n}{5}) = \Theta(n)$
        sum += array[i]* array[i]; } $\Theta(1)$ constant time
    return sum; } $\Theta(1)$
}

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

$T(n) = \underline{\Theta(n)}$

e) void p_5 (int array[], int n){
    for(int i=0 ; i<n; i++) $\rightarrow \Theta(n)$
        for(int j=1; j<i ; j=j*2) $\rightarrow \Theta(\log n)$
            Print ("%d", array[i]* array[j]); $\rightarrow \Theta(1)$
}

$T(n) = \Theta(n) \cdot \Theta(\log n) = \underline{\Theta(n \log n)}$

CamScanner ile tarandı

f and g:

f) int p_6 (int array[], int n){

if (p_4(array, n)) >1000) → $O(n)$

p_5 (array, n) → $O(n\log_2 n)$

else printf("%d", $\underbrace{\text{p\_3(array)}}_{O(1)}$ * $\underbrace{\text{p\_4(array, n)}}_{O(n)}$)

}

if → $O(n\log_2 n) + O(n)$

else → $O(n)$

best $T_b(n) = \underline{O(n)}$

worst $T_w(n) = O(n) + O(n\log_2 n) = \underline{O(n\log_2 n)}$

because of if-else statement, Big-oh notation is most appropriate

g) int p_7 (int n){

int i=n; → $\Theta(1)$

while(i>0){ → i/2 → $\Theta(\log_2 n)$

for(int j=0;j<n;j++) → $\Theta(n)$

System.out.println("*"); → $\Theta(1)$

i=i/2;

}

}

$T(n) = \Theta(1) + \Theta(\log_2 n) * \Theta(n) + \Theta(1)$

$= \Theta(n\log_2 n)$

h and I:

h)

```
int p_8(int n){
    while(n>0){          -> Θ(log_2 n)
        for(int j=0; j<n; j++)   -> Θ(log_2 n)
            System.out.println("*");   -> Θ(log_2 n)
        n=n/2;
    }
}
```

$T(n) = \Theta(\log_2 n) * \Theta(\log_2 n) = \Theta(\log_2^2 n)$

i)

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

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

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

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

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

$\Rightarrow$ So,

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

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

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

⋮ for k times.

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

assume $n = k$, so

$T(n) = T(0) + n$

$T(n) = 1 + n$

$\Theta(n)$

CamScanner ile tarandı

j:

j)

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

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

$= \text{simply} = T(n-1) + n$

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

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

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

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

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

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

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

relation

$T(n) = T(n-k) + (n-k+1) + (n-k+2) + \dots + (n-1) + n$

Assume $n-k = \phi$

$n = k+1$

$T(n) = T(\phi) + 2 + 3 + \dots + (n-1) + n$

$T(n) = 1 + 2 + 3 + \dots + n$

$T(n) = \dfrac{n.(n+1)}{2} = O(n^2)$

Q4)

a

Statement says T(n) is at least (n^2). It means that T(n) is upper bound of f(n). Since f(n) could be any function smaller than n^2.

If T(n) >= O(n^2) then n^2 > f(n) >= 0. Running time always non-negative. There is no information about upper bound of T(n) and lower bound of f(n) to. O (n2) = It is a worst-case scenario of running time so running time of algorithm A will be n 2 or faster. But algorithm A could be anything that is smaller. Example: Constant 1 or n…

This statement is True but uninformative and redundant.

b

1 and 2

I. $2^{n+1} = \Theta(2n)$

theta Notation => $f(n) = O(g(n)) \iff g(n) = \Omega(f(n))$

$c_1 \frac{2^n}{2^n} \le \frac{2^{n+1}}{2^n} \le c_2 \frac{2^n}{2^n}$

$c_1 \le 2 \le c_2$ for all $n \ge 0$

when $c_1 = c_2 = 2$, this statement is right. for all $n \ge n_0$

so, $2^{n+1} = \Theta(2^n)$ is true.

II. $2^{2n} = \Theta(2^n)$

$T(n) = O(T_w(n)) = \Omega T_b(n))$

$c_1 . 2^n \le 2^{2n} \le 2^n . c_2$

$c_1 . \cancel{2^n} \le \cancel{2^n} . 2^n \le c_2 . \cancel{2^n}$ for all $n \ge n_0$

$c_1 \le 2^n \le c_2$ for all $n \ge 0$

This statement is false, because 2^n grows exponentially. c_1 and c_2 are two constant, $c_1 \le 2^n$ can be provided but $2^n \le c_2$ cannot be provided so, it is FALSE b.I.

$f(n) = O(n^2)$, $g(n) = \Theta(n^2)$, $f(n) * g(n) = \Theta(n^4)$ =? true or false?

It is false, $O(n^2)$ is worst case scenario of $f(n)$

This kind of things may occur:

$$f(n) = \Theta(n) \rightarrow f(n) * g(n) = \Theta(3)$$

$$f(n) = \Theta(1) \rightarrow f(n) * g(n) = \Theta(2)$$

If we do not have lower bound, we could not say $f(n) * g(n) = \Theta(n^4)$, so it is false.

Q5)

a)

a) $T(n) = 2T(n/2) + n \;,\; T(1) = 1$

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

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

$T(n/4) = 2T(n/8) + n/4$

so,

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

$T(n) = 4T(n/4) + \cancel{2} \cdot n/\cancel{2} + n$

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

$T(n) = 8T(n/8) + n + n + n$

$T(n) = 2^k T(n/2^k) + \underbrace{n + n + n}_{k}$

Assume;

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

$\frac{n}{2^k} = 1 \;,\; n = 2^k$

$k = \log_2 n$

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

$T(n) = n \times 1 + n \cdot \log n$

$O(n \log n)$

b)

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

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

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

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

$\vdots \qquad \vdots \quad \vdots$

$\Rightarrow$

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

$T(n) = 4 \cdot T(n-2) + 2 + 1$

$T(n) = 8T(n-3) + 4 + 2 + 1$

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

$\therefore$ Assume $n = k$

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

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

$T(n) = \dfrac{1-2^n}{1-2} = 2^n - 1$

$T(n) = \Theta(2^n - 1) = \underline{\Theta(2^n)}$

Q6)

```java
// Iterative algorithm
public static void findSumPair(int[] array, int target)
{
    for (int i = 0; i < array.length - 1; i++)   -Θ(n)
    {
        for (int j = i + 1; j < array.length; j++)   -Θ(n)
        {
            if (array[i] + array[j] == target)   -1
            {
                System.out.println("(" + array[i] + "," + array[j] + ")");   -Θ(1)
            }
        }
    }
}
```

6)

Outer loop: $\Theta(n)$ – lineer

Inner loop : $\Theta(n)$ – lineer

If : $\Theta(1)$ – Constant , so

$T(n) = \Theta(n.n) = \Theta(n^2)$

Test on IDE: Intellij, jdk 11

```java
System.out.println("iterative func, test: size of array: 10");
start = System.currentTimeMillis();
findSumPair(array, target);
finish = System.currentTimeMillis();
timeElapsed = finish - start;
System.out.println();
System.out.println("Running time: " + timeElapsed + " ms ");
```

```
/home/alex/.jdks/corretto-11.0.14.1/bin/java -javaagent:/app/extra/idea-IU/lib/idea_rt.jar=44425:/app/extra/idea-IU/bin -Dfile.
iterative func, test: size of array: 10
(5,5)(5,5)(3,7)(5,5)
Running time: 45 ms
iterative func, test: size of array: 20
(5,5)(5,5)(5,5)(5,5)(5,5)(3,7)(3,7)(5,5)(5,5)(5,5)(5,5)(7,3)(5,5)(5,5)(5,5)(5,5)(5,5)(3,7)(5,5)
Running time: 0 ms
iterative func, test: size of array: 40
(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(3,7)(3,7)(3,7)(3,7)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(7
Running time: 3 ms
iterative func, test: size of array: 80
(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(3,7)(3,7)(3
Running time: 10 ms
iterative func, test: size of array: 160
(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5
Running time: 21 ms

Process finished with exit code 0
```

First function works very long every test. Other functions show a relation between run time results. Array sizes doubled every time. When we look at the test results, we see that we got the right results. Q(2^n)

Q7)

Test on IDE: Intellij, jdk 11

```java
    // Recursive algorithm
    public static void findSumPairRec(int[] array, int target, int firstIndex, int nextIndex){

        // first element to check is the last element in the array.
        // base case
        if (firstIndex >= array.length - 1) {
            return;
        }

        // Advance the first element to the
        // next index and compare it to the rest of the elements left in the array.
        if (nextIndex >= array.length) {
            findSumPairRec(array, target, firstIndex: firstIndex + 1, nextIndex: firstIndex + 2);
            return;
        }

        if (array[firstIndex] + array[nextIndex] == target) {
            System.out.println("(" + array[firstIndex] + "," + array[nextIndex] + ")");
        }

        // Compare first element to the next element in the array.
        findSumPairRec(array, target, firstIndex, nextIndex: nextIndex + 1);

    }
}
```

Q(7)

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

$$T(n) = \begin{cases} 1, & n = 0 \\ 2T(n-1) + 2, & n > 0 \end{cases} \longrightarrow$$

In function, Recursion goes from first_index which is 0, to the array.length-1. But it can wrote as (0 - length) or (length - 0)

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

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

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

k time k times

$\Rightarrow$ So,

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

$$T(n) = 4T(n-2) + 4 + 2$$

$$T(n) = 8T(n-3) + 8 + 4 + 2$$

k times

$$T(n) = 2^k T(n-k) + \underbrace{\dots + 4 + 2}_{k}$$

!: Assume $n = k$

$$T(n) = 2^n \underbrace{T(0)}_{1} + 2 + 4 + \dots + 2^k$$

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

$$T(n) = 2^n + 2.(2^k - 1)$$

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

$$T(n) = 2^n + 2.2^n - 2$$

$$T(n) = 3.2^n - 2 \rightarrow O(2^n)$$

Test on IDE: Intellij, jdk 11

```
System.out.println("recursion");
start = System.nanoTime();
findSumPairRec(array, target, firstIndex: 0, nextIndex: 1);
finish = System.nanoTime();
timeElapsed = finish - start;
System.out.println();
System.out.println("Running time: " + timeElapsed + " ns ");
```

```
/home/alex/.jdks/corretto-11.0.14.1/bin/java -javaagent:/app/extra/idea-IU/lib/idea_rt.jar=41973:/app/extra/idea-IU/bin -Dfile.encoding=UTF-8 -c
recursion
(5,5)(5,5)
Running time: 45163690 ns
recursion
(5,5)(5,5)(5,5)(5,5)(5,5)
Running time: 207011 ns
recursion
(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)
Running time: 491618 ns
recursion
(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)
Running time: 939654 ns
recursion
(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5)(5,5
Running time: 1818546 ns

Process finished with exit code 0
```

First function works very long every test. Other functions show a relation between run time results. Array sizes doubled every time.