CSE 222 - SPRING 2022

HOMEWORK 2

Emircan Demirel
1901042674

1) a) $\log_2 n^2$'s growth rate is slower than $O(n)$. It is not best informative statement but it's true.

b) $\sqrt{n(n+1)} \Rightarrow \sqrt{n^2+\cancel{n}} \Rightarrow \sqrt{n^2} \Rightarrow n$

$n = \Omega(n)$

True. Because growth rates of two sides of this equality are same.

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

Growth rates of two sides of this equality are same so, we can say that is true.

2)

$$\lim_{n\to\infty} \frac{n^3}{n^2} = \frac{3n^2}{2n} \Rightarrow \infty$$ growth rate $n^3 > n^2$

$$\lim_{n\to\infty} \frac{n^2 \log n}{\log n} = \frac{2n \cdot \cancel{1/n}}{\cancel{1/n}} = 2n \Rightarrow \infty$$ growth rate $n^2 \log n > \log n$

$$\lim_{n\to\infty} \frac{8^{\log_2 n}}{2^n} = 0 \Rightarrow$$ growth rate $2^n > 8^{\log_2 n}$

$$\lim_{n\to\infty} \frac{n^2 \log n}{\sqrt{n}} = \infty \Rightarrow$$ growth rate $n^2 \log n > \sqrt{n}$

$$\underbrace{\log n < \sqrt{n} < n^2 \log n < n^2 < n^3 < 8^{\log_2 n} < 2^n < 10^n}_{\text{comparison of growth rates}}$$

3) a) for loop : $\Theta(n)$

if condition $\rightarrow \Theta(1)$

statement $\rightarrow \Theta(1)$

else statement $\rightarrow \Theta(1)$

} Time complexity $O(n)$ for p-1

b) for loop : $\Theta(n)$

other parts of function done its works in constant time: $\Theta(1)$

} Time complexity $O(n)$

c) function returns the multiplication of numbers in given indexes of array.

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

d) The function make same initialization n times.

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

e) This function contains nested loop. The growth rate of the inner loop is ($\log n$) and the outer loop's is n so; $T(n) = O(n * \log n)$

f) if condition: $\Theta(n)$

statement : $O(n * \log n)$

else statement; $\Theta(1) + \Theta(n)$

so, $T(n) = O(n * \log n)$

g) Inner loop's growth rate is n

Inner loop: $\Theta(n)$

Outer loop: $\Theta(\log n)$

> so, $T(n) = O(n * \log n)$

h) Inner loop's growth rate is depended to outer while loop and it's $\log n$.

So, Inner for loop : $\Theta(\log n)$
Outer while loop: $\Theta(\log n)$ $> T(n) = O(\log^2 n)$

i) best case $\Theta(1)$
worst case $\Theta(n) \Rightarrow$ recursive call n times

$$T(n) = O(n)$$

j) If statement: $\Theta(1)$

recursive call : $\Theta(n)$

while loop: $\Theta(n-1)$ and contains some constants.

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

4) a) Big Oh notation represents upper bounds for an algorithm, so this statement is false. Because it says $O(n^2)$ is lower bound. Correct statement should be "The running time of algorithm A is at most $O(n^2)$".

b) I. $f(n) = 2^n$,

$2^{n+1} = 2 \cdot 2^n$, this contant value '2' doesn't affect the growth rate of algorithm.

$\Theta(2^{n+1}) = \Theta(2^n)$ $\Rightarrow$ True

II. $f(n) = 2^n$

$\Theta(f(n)) = \Theta(2^n) \neq \Theta(2^{2n})$, the coefficient value of n's affects the growth rate. $\Rightarrow$ False

III. $f(n) = O(n^2)$ $g(n) = \Theta(n^2)$

If we multiply O and Θ result must be big Oh notation.

$f(n) \times g(n) \neq \Theta(n^4)$

$f(n) \times g(n) = O(n^4)$

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

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

$T(n) = 2 \cdot T(\frac{n}{2}) + n \Rightarrow$ if $n = n/2$
$T(\frac{n}{2}) = 2T(\frac{n}{4}) + \frac{n}{2}$

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

$= 2^2 T(\frac{n}{2^2}) + 2n \Rightarrow$ if $n = n/4$
$T(\frac{n}{4}) = 2T(\frac{n}{8}) + \frac{n}{4}$

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

Assume that $\frac{n}{2^k} = 1$

$T(n) = 2^k T(\frac{n}{2^k}) + kn \Rightarrow$ $\frac{n}{2^k} = 1$ $n = 2^k$

$\underline{k = \log n}$

$T(n) = 2^k T(1) + kn$
$= n + n\log n = \underline{O(n * \log n)}$

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

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

$T(n) = 2T(n-1)+1 \Rightarrow$ if $n=n-1 \Rightarrow T(n-1) = 2T(n-2)+1$

if $n=n-2 \Rightarrow T(n-2) = 2T(n-3)+1$

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

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

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

assume that $n-k=0$

$n=k$

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

$\underline{O(2^n)}$

```java
public static void Pair(int[] arr, int sum){
    for (int i = 0; i < arr.length-1; i++) {
        for (int j = i+1; j < arr.length; j++) {
            if(arr[i] + arr[j] == sum){
                System.out.println(arr[i] + ", " + arr[j]);
            }
        }
    }
}
```

iterative algorithm to find pairs of numbers with the given sum

6) Running time:

outer loop iteration $(n-1)$ times $\quad (n-1)$

inner loop iteration $(n-1)$ times $\quad \times\ (n-1)$

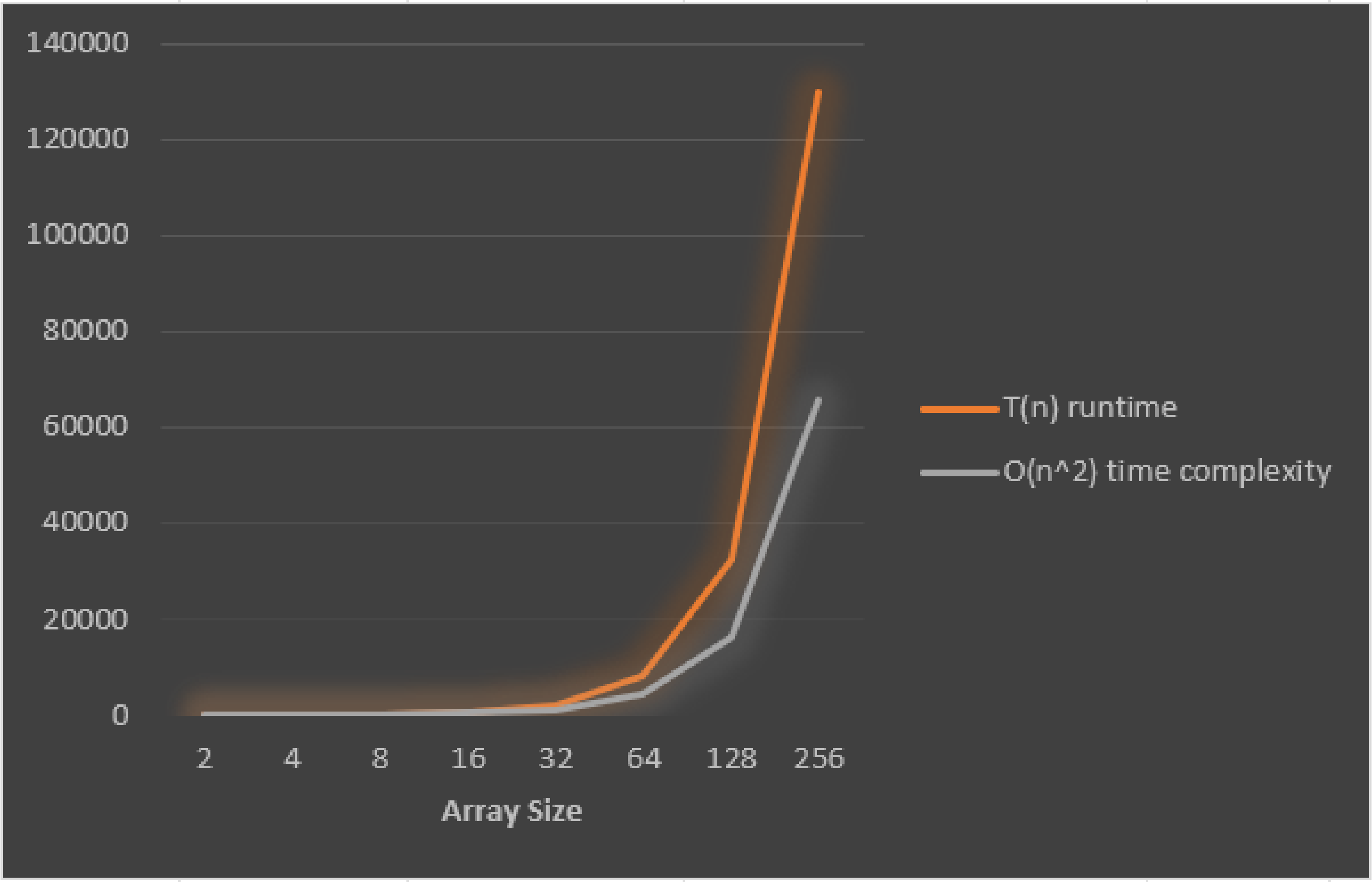
if condition at every step $\rightarrow n^2 - 2n + 1$

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

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

Array Size	T(n) runtime	O(n^2) time complexity
2	2	4
4	18	16
8	98	64
16	450	256
32	1922	1024
64	7938	4096
128	32258	16384
256	130050	65536

expected runtime tests with different size arrays



graphic of the expected runtime tests

```
----------------------------
array size: 2
runtime: 2800 ns
----------------------------
array size: 4
runtime: 1000 ns
----------------------------
array size: 8
runtime: 1600 ns
----------------------------
array size: 16
runtime: 2900 ns
----------------------------
array size: 64
runtime: 32600 ns
----------------------------
array size: 128
runtime: 119300 ns
```

actual runtime test results

7) firstIndex increases n-1 times

nextIndex increases n-1 times

we have 3 if conditions at every step

$$T(n) = 3.(n-1)^2$$

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

$$T(n) = 3n^2-6n+3$$

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

```
public static void Pair(int[] arr, int sum, int firstIndex, int next){
    if(firstIndex >= arr.length - 1)
        return;
    else if(next >= arr.length){
        Pair(arr, sum, firstIndex+1, firstIndex+2);
        return;
    }
    else if(arr[firstIndex] + arr[next] == sum){
        System.out.println(arr[firstIndex] + ", " + arr[next]);
    }
    Pair(arr, sum, firstIndex, next+1);
}
```

recursive algorithm to find pairs of numbers with the given sum