GIT Department of Computer Engineering

CSE 222/505 - Spring 2022

Homework #2 Report

Hasan Mutlu

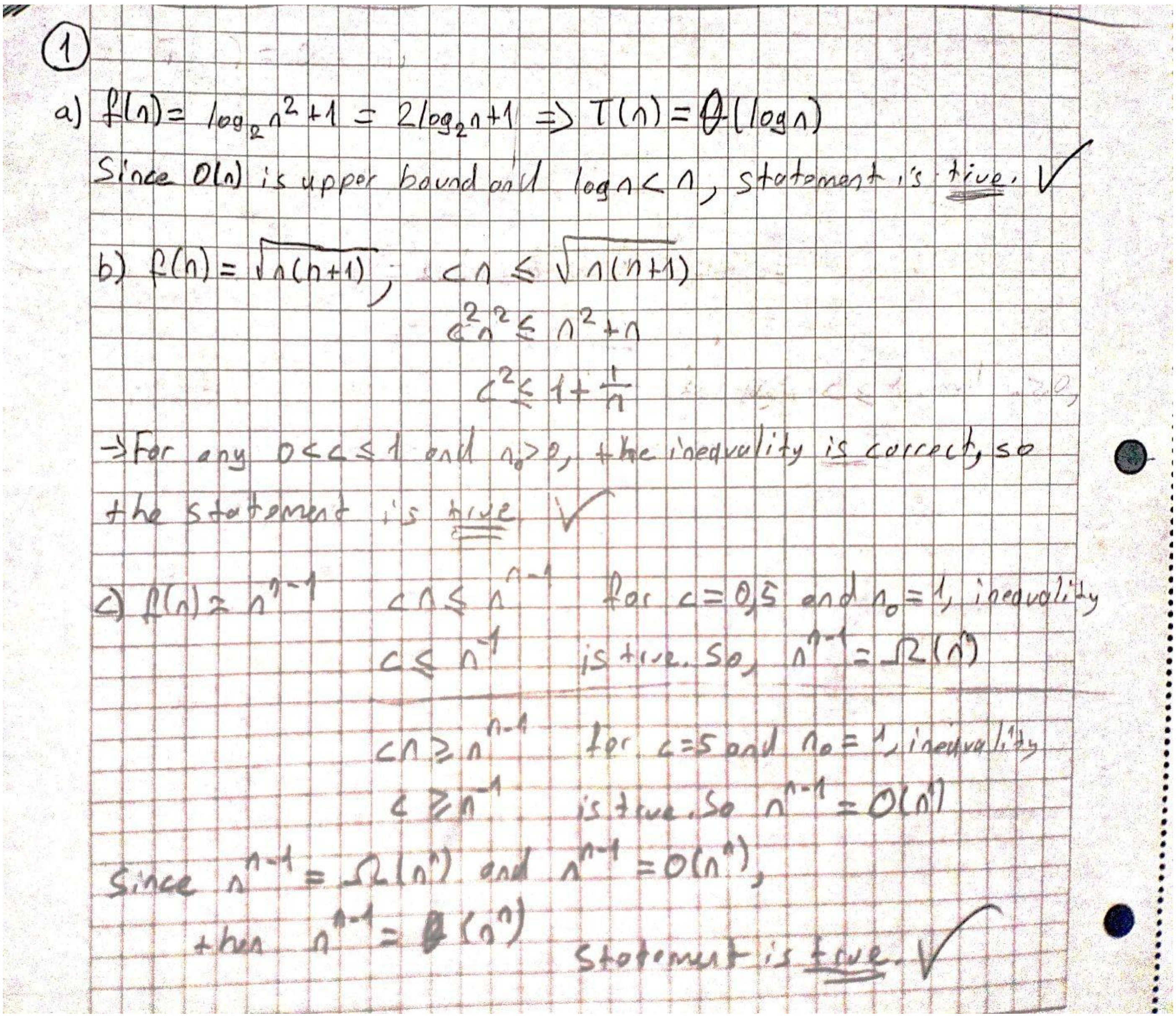
1801042673

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

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

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



①

a) $f(n) = \log_2 n^2 + 1 = 2\log_2 n + 1 \Rightarrow T(n) = \theta(\log n)$

Since $O(n)$ is upper bound and $\log n < n$, statement is <u>true</u>. ✓

b) $f(n) = \sqrt{n(n+1)}$, $\quad cn \leq \sqrt{n(n+1)}$

$c^2 n^2 \leq n^2 + n$

$c^2 \leq 1 + \frac{1}{n}$

→ For any $0 < c \leq 1$ and $n_0 > 0$, the inequality is correct, so the statement is <u>true</u> ✓

c) $f(n) = n^{n-1}$ $\quad cn \leq n^{n-1}$ $\quad$ for $c = 0,5$ and $n_0 = 1$, inequality

$c \leq n^{-1}$ $\quad$ is true. So, $n^{n-1} = \Omega(n^n)$

$cn \geq n^{n-1}$ $\quad$ for $c = 5$ and $n_0 = 1$, inequality

$c \geq n^{-1}$ $\quad$ is true. So $n^{n-1} = O(n^n)$

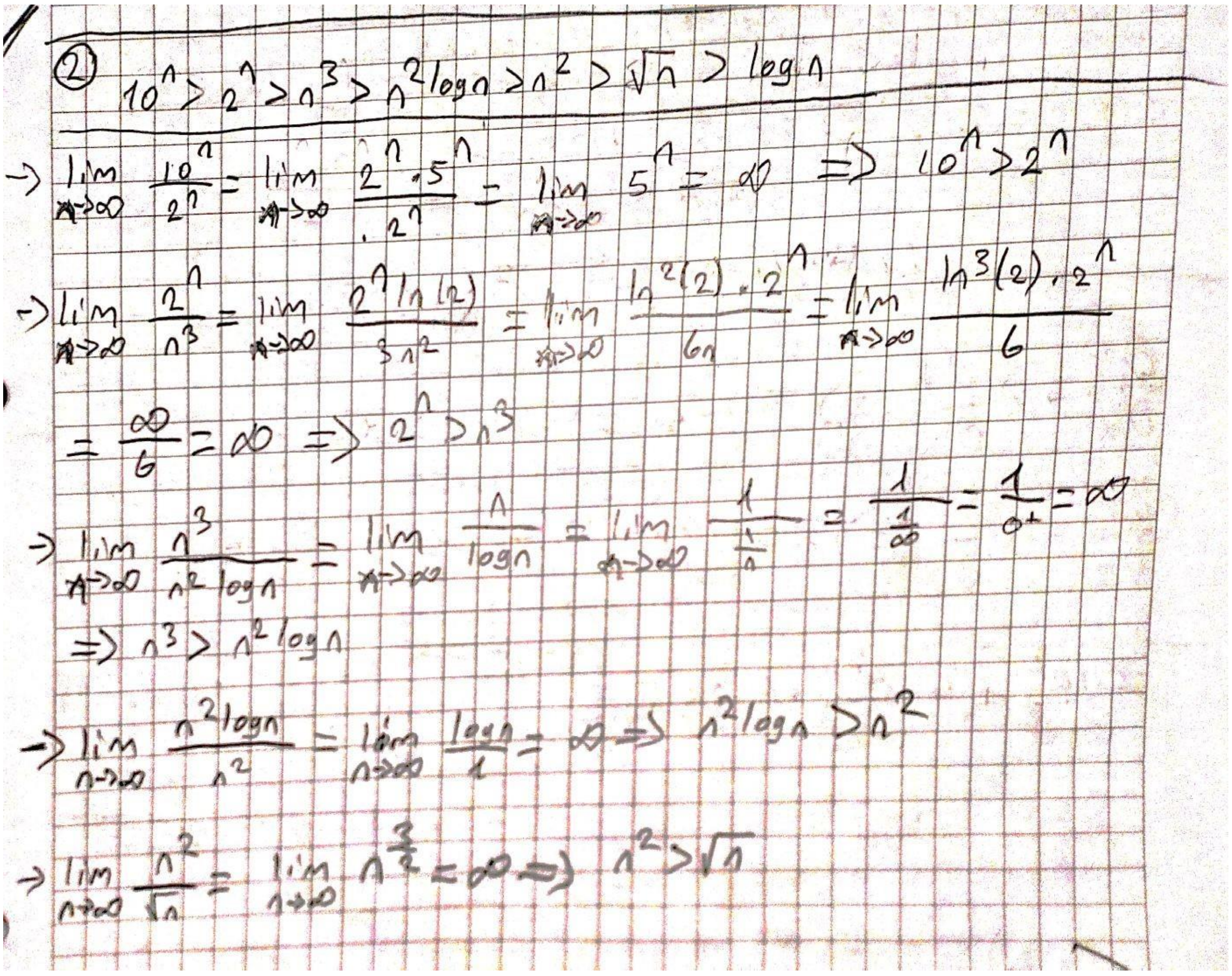
Since $n^{n-1} = \Omega(n^n)$ and $n^{n-1} = O(n^n)$,

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

Statement is <u>true</u>. ✓

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

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



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

$\to \lim\limits_{n\to\infty} \frac{10^n}{2^n} = \lim\limits_{n\to\infty} \frac{2^n \cdot 5^n}{2^n} = \lim\limits_{n\to\infty} 5^n = \infty \Rightarrow 10^n > 2^n$

$\to \lim\limits_{n\to\infty} \frac{2^n}{n^3} = \lim\limits_{n\to\infty} \frac{2^n \ln(2)}{3n^2} = \lim\limits_{n\to\infty} \frac{\ln^2(2) \cdot 2^n}{6n} = \lim\limits_{n\to\infty} \frac{\ln^3(2) \cdot 2^n}{6}$

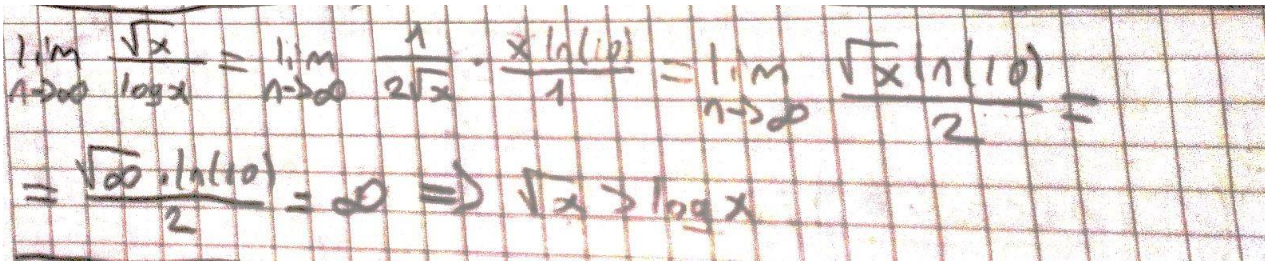
$= \frac{\infty}{6} = \infty \Rightarrow 2^n > n^3$

$\to \lim\limits_{n\to\infty} \frac{n^3}{n^2 \log n} = \lim\limits_{n\to\infty} \frac{n}{\log n} = \lim\limits_{n\to\infty} \frac{1}{\frac{1}{n}} = \frac{1}{\frac{1}{\infty}} = \frac{1}{0^+} = \infty$

$\Rightarrow n^3 > n^2 \log n$

$\to \lim\limits_{n\to\infty} \frac{n^2 \log n}{n^2} = \lim\limits_{n\to\infty} \frac{\log n}{1} = \infty \Rightarrow n^2 \log n > n^2$

$\to \lim\limits_{n\to\infty} \frac{n^2}{\sqrt{n}} = \lim\limits_{n\to\infty} n^{\frac{3}{2}} = \infty \Rightarrow n^2 > \sqrt{n}$



$\lim\limits_{n\to\infty} \frac{\sqrt{x}}{\log x} = \lim\limits_{n\to\infty} \frac{1}{2\sqrt{x}} \cdot \frac{x \ln(10)}{1} = \lim\limits_{n\to\infty} \frac{\sqrt{x} \ln(10)}{2} =$

$= \frac{\sqrt{\infty} \cdot \ln(10)}{2} = \infty \Rightarrow \sqrt{x} > \log x$

3) What is the time complexity of the following programs? Use most appropriate asymptotic notation. Explain by giving details.

p_1:

for(int i=2; i<=n; i++){	2	n-1	2n-2
if(i%2==0){	2	n-1	2n-2
count++;	1	n-1	n-1
i=(i-1)i;	3	n-1	3n-3

Every instruction in if-else blocks are executed θ(n) times so the answer is **θ(n).**

p_2:

first_element = my_array[0];	1	1	1
second_element = my_array[0];	1	1	1
for(int i=0; i<sizeofArray; i++){	2	n	2n
if(my_array[i]<first_element){	1	n	n
second_element=first_element;	1	n	n
first_element=my_array[i];	1	n	n
else if(my_array[i]<second_element){	1	n	n
if(my_array[i]!= first_element){	1	n	n
second_element= my_array[i];	1	n	n

Every instruction in if-else blocks are executed n times so that the answer is **θ(n).**

p_3: Answer: θ(n).

p_4:

Int sum = 0	1	1	1
for (int i = 0; i < n; i=i+5)	2	logn	2logn
sum += array[i] * array[i];	2	logn	2logn
return sum;	1	1	1

Answer: **θ(logn).**

p_5:

for (int i = 0; i < n; i++)	2	n	2n
for (int j = 1; j < i; j=j*2)	2	nlogn	2logn
printf("%d", array[i] * array[j]);	2	nlogn	2logn

Answer: **θ(nlog(n)).**

p_6:

if (p_4(array, n)) > 1000)	1+logn	1	1+logn
p_5(array, n)	nlogn	1	nlogn
printf("%d", p_3(array) * p_4(array, n))	2 + n+ logn	1	2 + n+ logn

Answer: Depending on the if conditions, **O(nlogn) and Ω(logn)**

p_7:

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

Answer: **θ(nlog(n)).**

p_8:

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

Answer: **θ(nlog(n)).**

p_9: This one is recursive and answer: **θ(n)**

4)

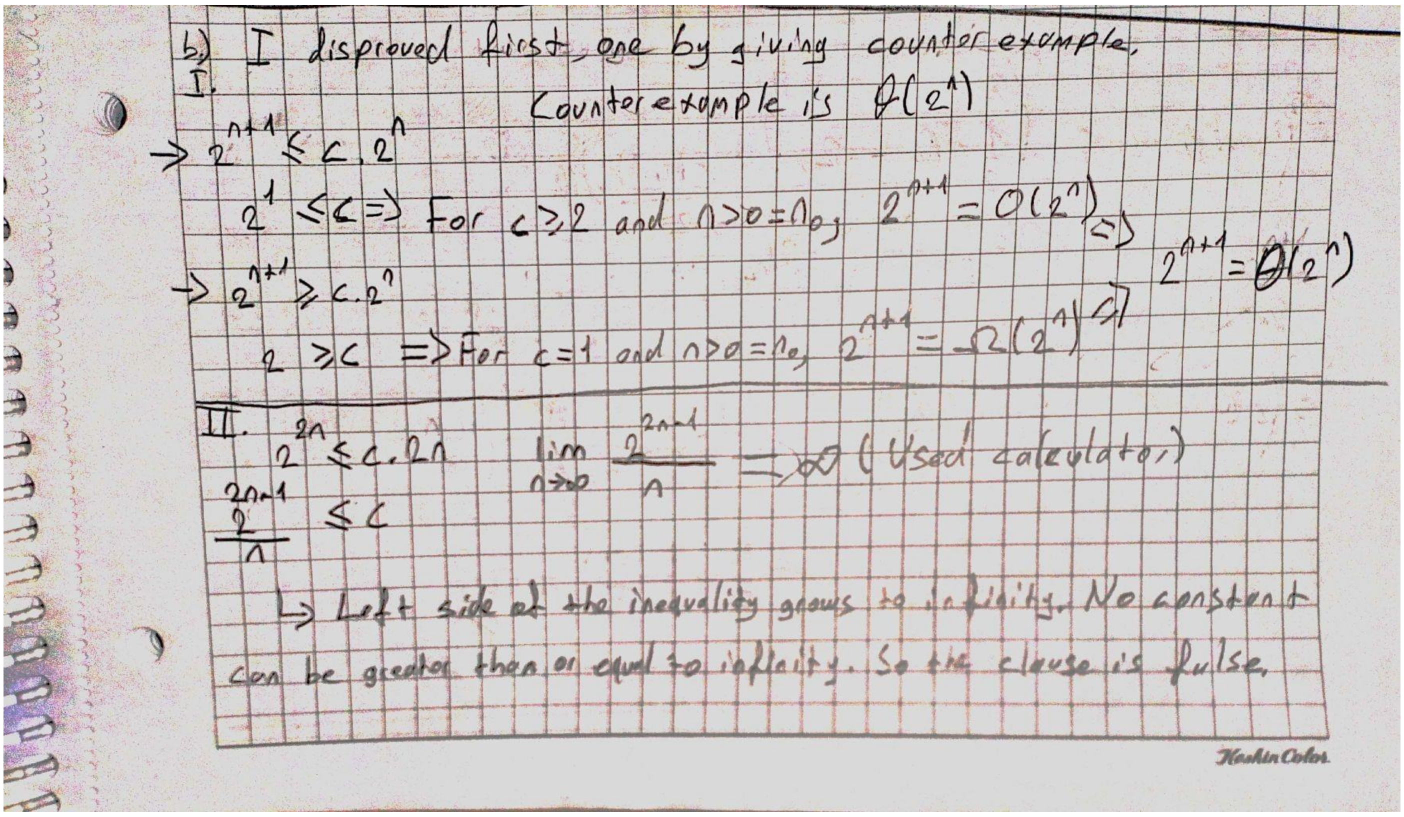
a) Explain what is wrong with the following statement. "The running time of algorithm A is at least $O(n^2)$".
Answer: Big O Notation provides an upper bound for an algorithm while the term "at least" provides a lower bound related information. It states the best case scenario so Omega notation should've been used instead of Big O.

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

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

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

I disproved first one by proving a counterexample, which is $\theta(2^n)$.



b) I disproved first, one by giving counter example.
I. Counterexample is $\theta(2^n)$

$\rightarrow 2^{n+1} \leq c.2^n$

$2^1 \leq c \Rightarrow$ For $c \geq 2$ and $n > 0 = n_0$, $2^{n+1} = O(2^n)$ $\Leftrightarrow$ $2^{n+1} = \theta(2^n)$

$\rightarrow 2^{n+1} \geq c.2^n$

$2 \geq c \Rightarrow$ For $c = 1$ and $n > 0 = n_0$, $2^{n+1} = \Omega(2^n)$ $\Leftrightarrow$

II. $2^{2n} \leq c.2n$ $\lim_{n\to\infty} \frac{2^{2n-1}}{n} = \infty$ (Used calculator)

$\frac{2^{2n-1}}{n} \leq c$

$\hookrightarrow$ Left side of the inequality grows to infinity. No constant can be greater than or equal to infinity. So the clause is false.

KaliKen Color

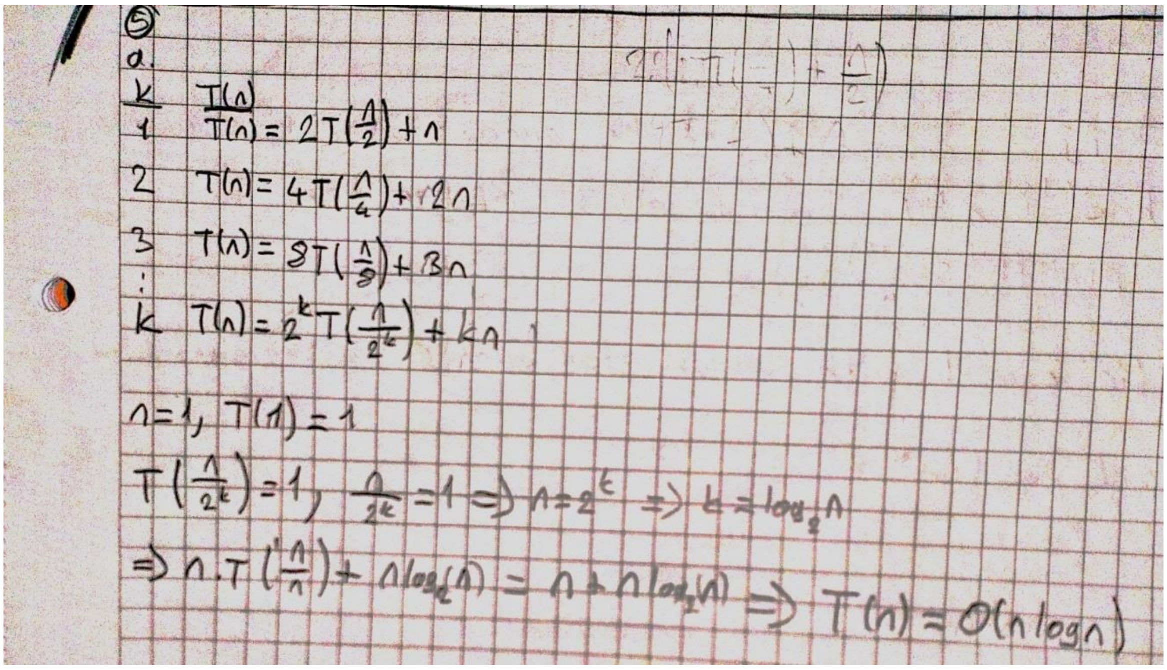
In second one, left side of the inequality grows to infinity. No constant can be greater than or equal to infinity so the clause is false.

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

Answer: Big O Notation doesn't provide tight bound like Theta notation does. This statement might be or not be true depending on the f(n)'s Omega value. If f(n) = Ω(n^2), then the statement is true as f(n) is also equal to Θ(n^2). But since it is not known, there is an ambiguity.

5) Solve the following recurrence relations. Express the result in most appropriate asymptotic notation. Show details of your work.

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



⑤

a.

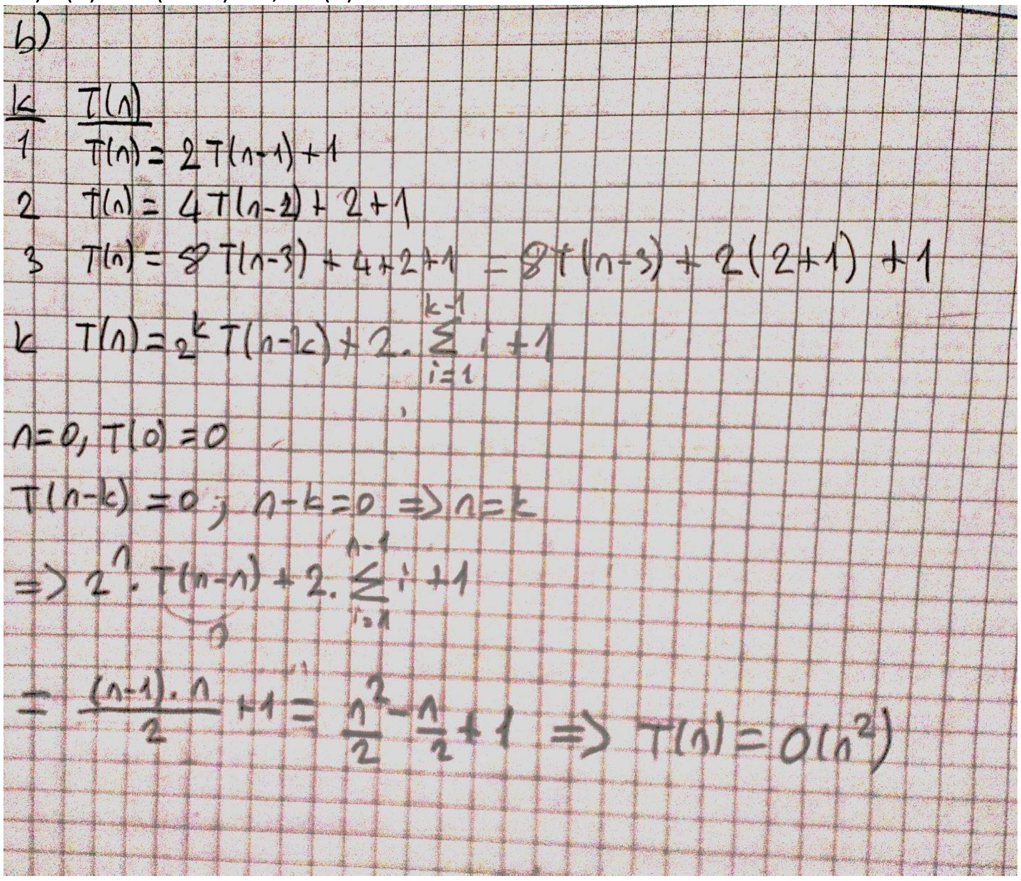
k	T(n)
1	$T(n) = 2T(\frac{n}{2}) + n$
2	$T(n) = 4T(\frac{n}{4}) + 2n$
3	$T(n) = 8T(\frac{n}{8}) + 3n$
$\vdots$	
k	$T(n) = 2^k T(\frac{n}{2^k}) + kn$

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

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

$\Rightarrow n.T(\frac{n}{n}) + n\log_2(n) = n + n\log_2(n) \Rightarrow T(n) = O(n \log n)$

b) T(n) = 2T(n − 1) + 1,   T(0)=0



b)

k	T(n)
1	$T(n) = 2T(n-1) + 1$
2	$T(n) = 4T(n-2) + 2 + 1$
3	$T(n) = 8T(n-3) + 4 + 2 + 1 = 8T(n-3) + 2(2+1) + 1$
k	$T(n) = 2^k T(n-k) + 2.\sum_{i=1}^{k-1} i + 1$

$n=0,\ T(0) = 0$

$T(n-k) = 0;\ n-k = 0 \Rightarrow n = k$

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

$= \frac{(n-1).n}{2} + 1 = \frac{n^2}{2} - \frac{n}{2} + 1 \Rightarrow T(n) = O(n^2)$

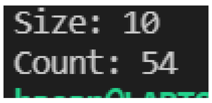
6) In an array of numbers (positive or negative), find pairs of numbers with the given sum. Design an iterative algorithm for the problem. Test the algorithm with different size arrays and record the running time. Calculate the resulting time complexity. Compare and interpret the test result with your theoretical result.

```
void func(int arr[], int size, int sum){
    for(int i = 0; i < size - 1; i++){
        //count++;
        for(int j = i + 1; j < size; j++){
            //count++;
            if(arr[i] + arr[j] == sum){
                printf("%d + %d = %d", arr[i], arr[j], sum);
            }
        }
    }
}
```

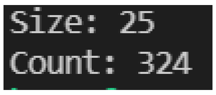
for(int i = 0; i < size - 1; i++){	2	n-1	2n-2
for(int j = i + 1; j < size; j++){	2	(n-1)(n/2)	n^2 – n
if(arr[i] + arr[j] == sum){	2	(n-1)(n/2)	n^2 – n
printf("%d + %d = %d", arr[i], arr[j], sum);	1	(n-1)(n/2)	n^2/2 – n/2

Theoretical time complexity: O(n^2)

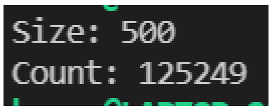
I added a counter and incremented it in each for loop iteration.
Test Results:



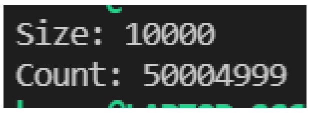
```
Size: 10
Count: 54
```



```
Size: 25
Count: 324
```



```
Size: 500
Count: 125249
```



```
Size: 10000
Count: 50004999
```

According to the test results, the count is a little over n^2/2. So the theoretical and real results are matching.

7) Write a recursive algorithm for the problem in 6 and calculate its time complexity. Write a recurrence relation and solve it.