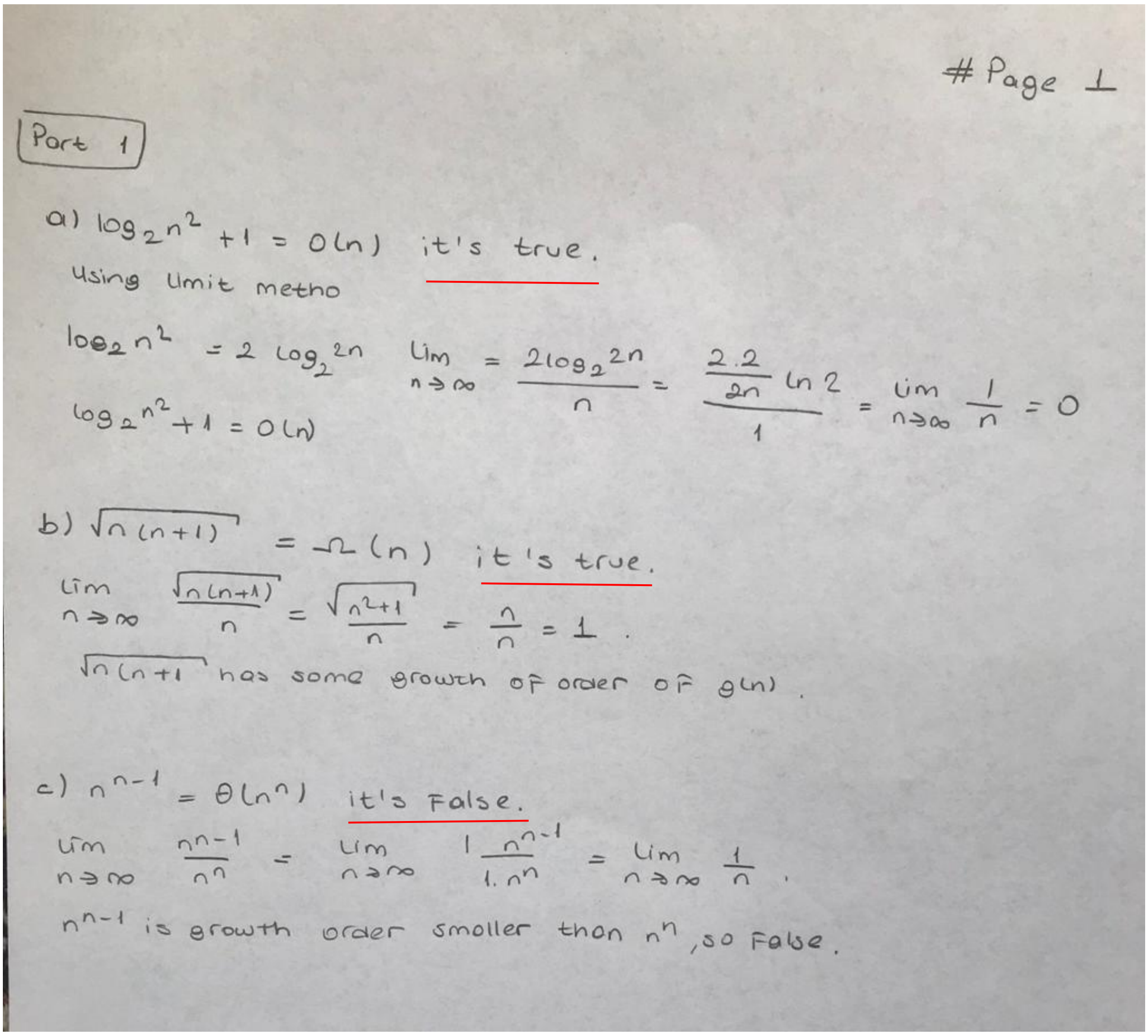
GTU Department of Computer Engineering
CSE 222/505 - Spring 2022
Homework 2 Report

Tuba TOPRAK
161044116



#Page 1

Part 1

a) $\log_2 n^2 + 1 = O(n)$ it's true.

Using limit metho

$\log_2 n^2 = 2 \log_2 2n$ $\quad \lim_{n \to \infty} = \frac{2\log_2 2n}{n} = \frac{\frac{2.2}{2n} \ln 2}{1} = \lim_{n \to \infty} \frac{1}{n} = 0$

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

b) $\sqrt{n(n+1)} = \Omega(n)$ it's true.

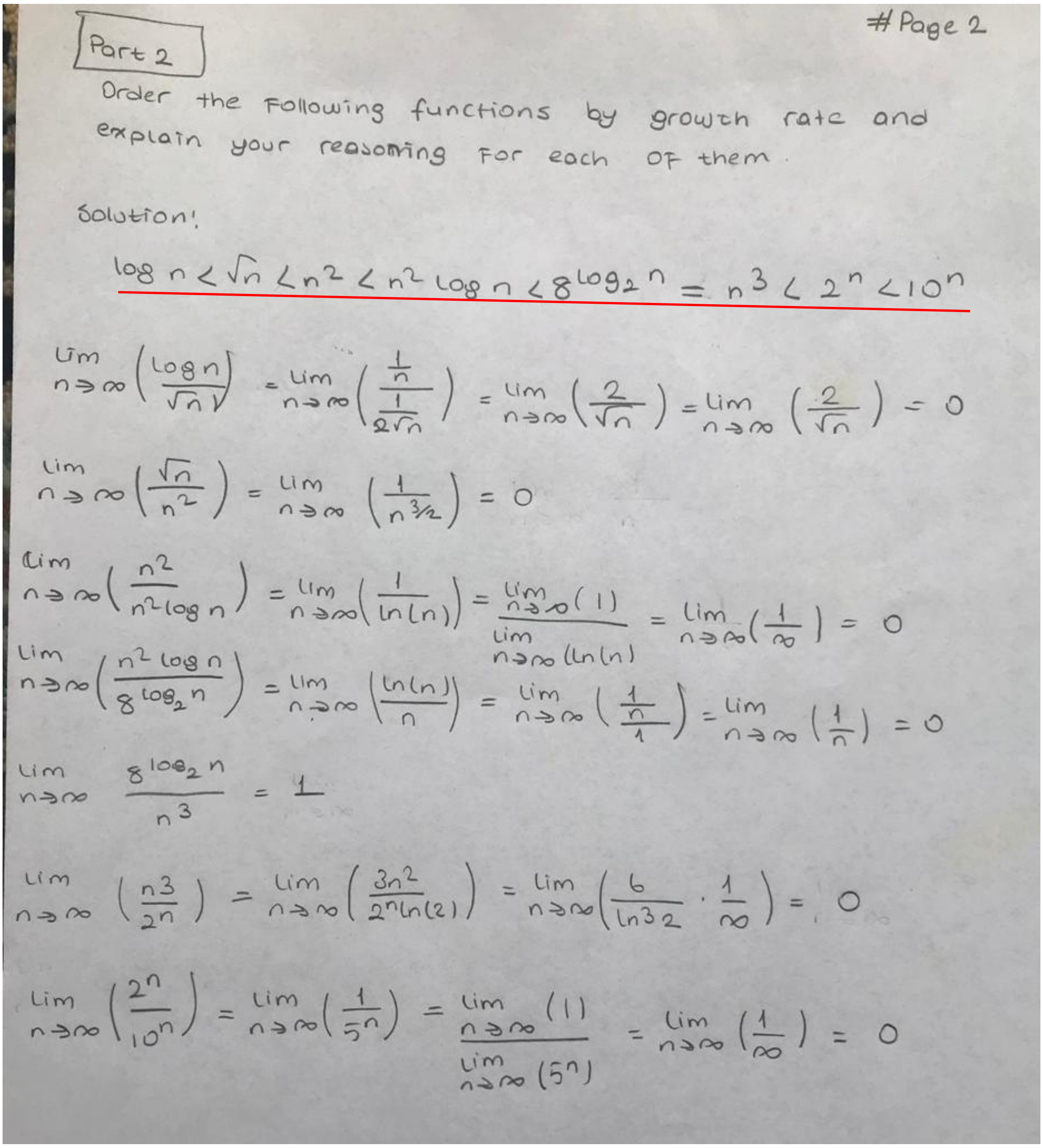
$\lim_{n \to \infty} \frac{\sqrt{n(n+1)}}{n} = \frac{\sqrt{n^2+1}}{n} = \frac{n}{n} = 1$.

$\sqrt{n(n+1}$ has some growth of order of $g(n)$.

c) $n^{n-1} = \Theta(n^n)$ it's False.

$\lim_{n \to \infty} \frac{n^{n-1}}{n^n} = \lim_{n \to \infty} \frac{1 \; n^{n-1}}{1.\, n^n} = \lim_{n \to \infty} \frac{1}{n}$.

n^{n-1} is growth order smoller than n^n ,so False.



#Page 2

Part 2

Order the Following functions by growth rate and explain your reasoning For each OF them.

Solution:

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

$$\lim_{n \to \infty} \left(\frac{\log n}{\sqrt{n}} \right) = \lim_{n \to \infty} \left(\frac{\frac{1}{n}}{\frac{1}{2\sqrt{n}}} \right) = \lim_{n \to \infty} \left(\frac{2}{\sqrt{n}} \right) = \lim_{n \to \infty} \left(\frac{2}{\sqrt{n}} \right) = 0$$

$$\lim_{n \to \infty} \left(\frac{\sqrt{n}}{n^2} \right) = \lim_{n \to \infty} \left(\frac{1}{n^{3/2}} \right) = 0$$

$$\lim_{n \to \infty} \left(\frac{n^2}{n^2 \log n} \right) = \lim_{n \to \infty} \left(\frac{1}{\ln(n)} \right) = \frac{\lim_{n \to \infty} (1)}{\lim_{n \to \infty} (\ln(n)} = \lim_{n \to \infty} \left(\frac{1}{\infty} \right) = 0$$

$$\lim_{n \to \infty} \left(\frac{n^2 \log n}{8^{\log_2 n}} \right) = \lim_{n \to \infty} \left(\frac{\ln(n)}{n} \right) = \lim_{n \to \infty} \left(\frac{\frac{1}{n}}{1} \right) = \lim_{n \to \infty} \left(\frac{1}{n} \right) = 0$$

$$\lim_{n \to \infty} \frac{8^{\log_2 n}}{n^3} = \underline{1}$$

$$\lim_{n \to \infty} \left(\frac{n^3}{2^n} \right) = \lim_{n \to \infty} \left(\frac{3n^2}{2^n \ln(2)} \right) = \lim_{n \to \infty} \left(\frac{6}{\ln 3\, 2} \cdot \frac{1}{\infty} \right) = 0$$

$$\lim_{n \to \infty} \left(\frac{2^n}{10^n} \right) = \lim_{n \to \infty} \left(\frac{1}{5^n} \right) = \frac{\lim_{n \to \infty} (1)}{\lim_{n \to \infty} (5^n)} = \lim_{n \to \infty} \left(\frac{1}{\infty} \right) = 0$$

a)

```
int p-1(int my-array[]){
    For(int i=2; i<=n, i++) {
        if (i % 2 == 0) {
            count++;
        }
        else {
            i=(i-1)i;
}}      }
```

Time Complexity
θ(log(log n))

b)

```
int p(int my-array[]){
  first-element=my-array[0]; → θ(1)
  second-element=my-array[0]; → θ(1)
  for(int i=0; i<sizeofarray, i++){ → runs n times. = θ(n)
    if(my-array[i] <first_element){
      second_element = first-element;
      first_element = my-array[i];
    }
    else if (my-array[i] <second-element){         runs 1 time
      if(my-array[i] != first-element){
        second_element =my-array[i];
    }}
}
```

Time Complexity
T(n) = θ(n)

c)

```
Int p-3(int array[]){
  return array[0] * array[2]; → θ(1)
}
```

Space Complexity S(n) = θ(n)

Time Complexity
T(n) = θ(1)

Because there is just one return, nothing else.

d)

```
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];
  return sum; → θ(1)
}
```

Time Complexity
T(n) = θ(1) + θ(n) + θ(1)
T(n) = θ(n)

Space Complexity

For "sum" and "n" they have space complexity as θ(1) because they have constant space complexity, but for array[n], it has θ(n) as space complexity.

S(n) = θ(n)

e)
```
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)
            print("%d", ar.....   ); Θ(1)
}   }  }
```

Time Complexity

$Tn = \Theta(n) * \Theta \log n * \Theta(1)$

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

Space complexity

$S(n) = \Theta(n)$

array

f)
```
void p_6 (int array[], int p){
    If(p-4(array,n)) >1000) Θ(n) because P_4
        p-5(array,n) Θ(n.log n because P_5
    else                                          P3      P-4
                                                   ↑       ↑
        print("%d",p-3(array)*p-4(array,n)) Θ(1) * Θ(n) = Θ(n)
}
```

Time Complexity

worst-case:

$T(n) = \Theta(n) + \Theta n + \Theta(n.\log n) = \Theta(n \log n)$

Best case:

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

Avorage Case: $O(n \log n)$

Space Complexity

for "n" variable, space complexity is constant, it means $\Theta(1)$ but for the "array" it has $\Theta(n)$ because of size, $S(n) = \Theta(n)$

g)
```
int p-7 (int n){
    int i=n;
    while (i>0) {Θ(log n)
        For (int j=0; j<n; j++)Θ(n)
            system.out.pritln("*");
        i=i/2;
    }
}
```

Time Complexity

$T(n) = O(n \log n)$

The two loops here are nested but the number of iterations the inner loop run is independent of the outer loop. Therefore, the total volume of statements can be taken

h)
```
int p-8(int n){
    while (n>0) {
        for (int j=0; j<n; j++)
            sout ("*");
        n=n/2;
    }
}
```

Time Complexity

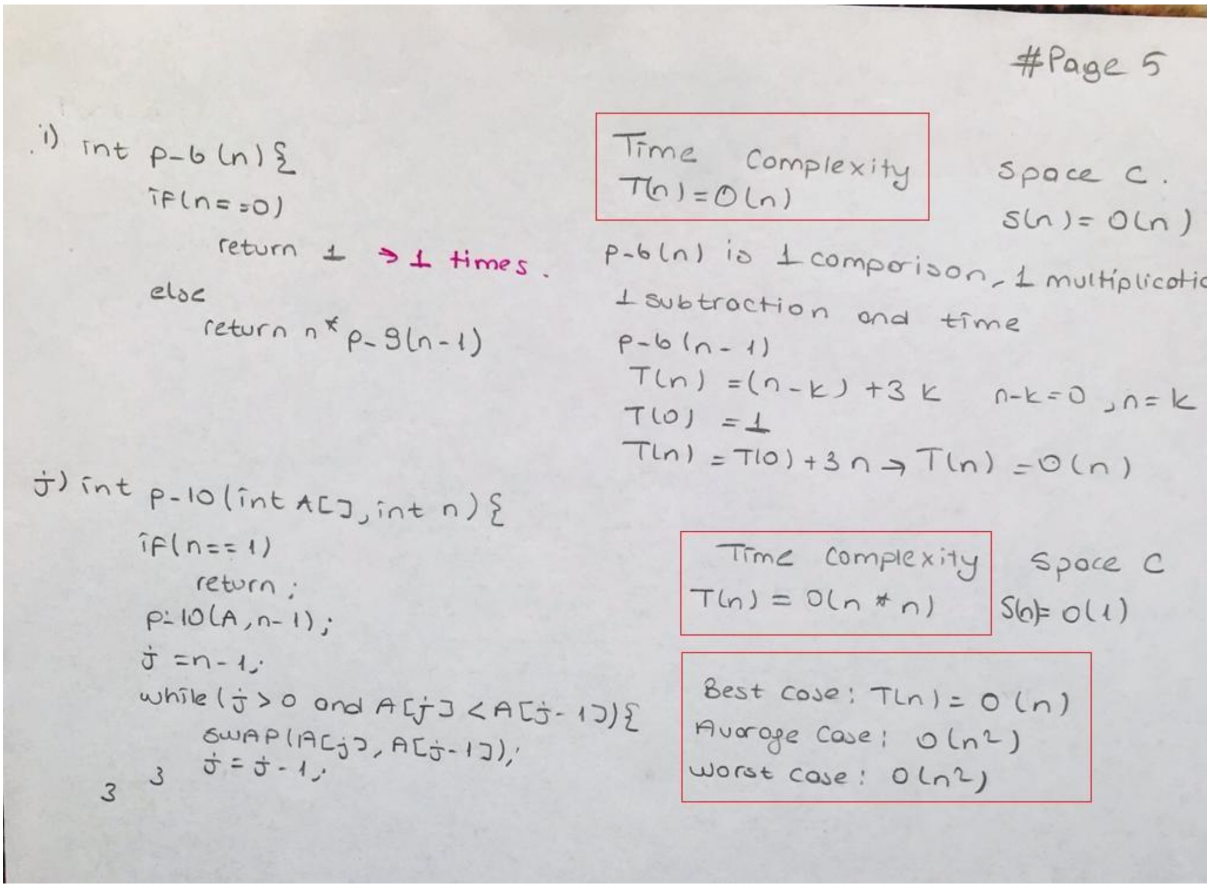
$T(n) = O(n)$

This series:

$1+2+3+4+ \ldots \frac{n}{4}+\frac{n}{2}+n = 2n-1$

$T(n) = 2n-1$

$T(n) = O(n)$



#Page 5

i)
```
int p-6(n){
    if(n==0)
        return 1  → 1 times.
    else
        return n*p-9(n-1)
```

Time Complexity
$T(n) = O(n)$

Space C.
$S(n) = O(n)$

p-6(n) is 1 comparison, 1 multiplicatio
1 subtraction and time
p-6(n-1)

$T(n) = (n-k) + 3k$ $n-k=0$, $n=k$

$T(0) = \underline{1}$

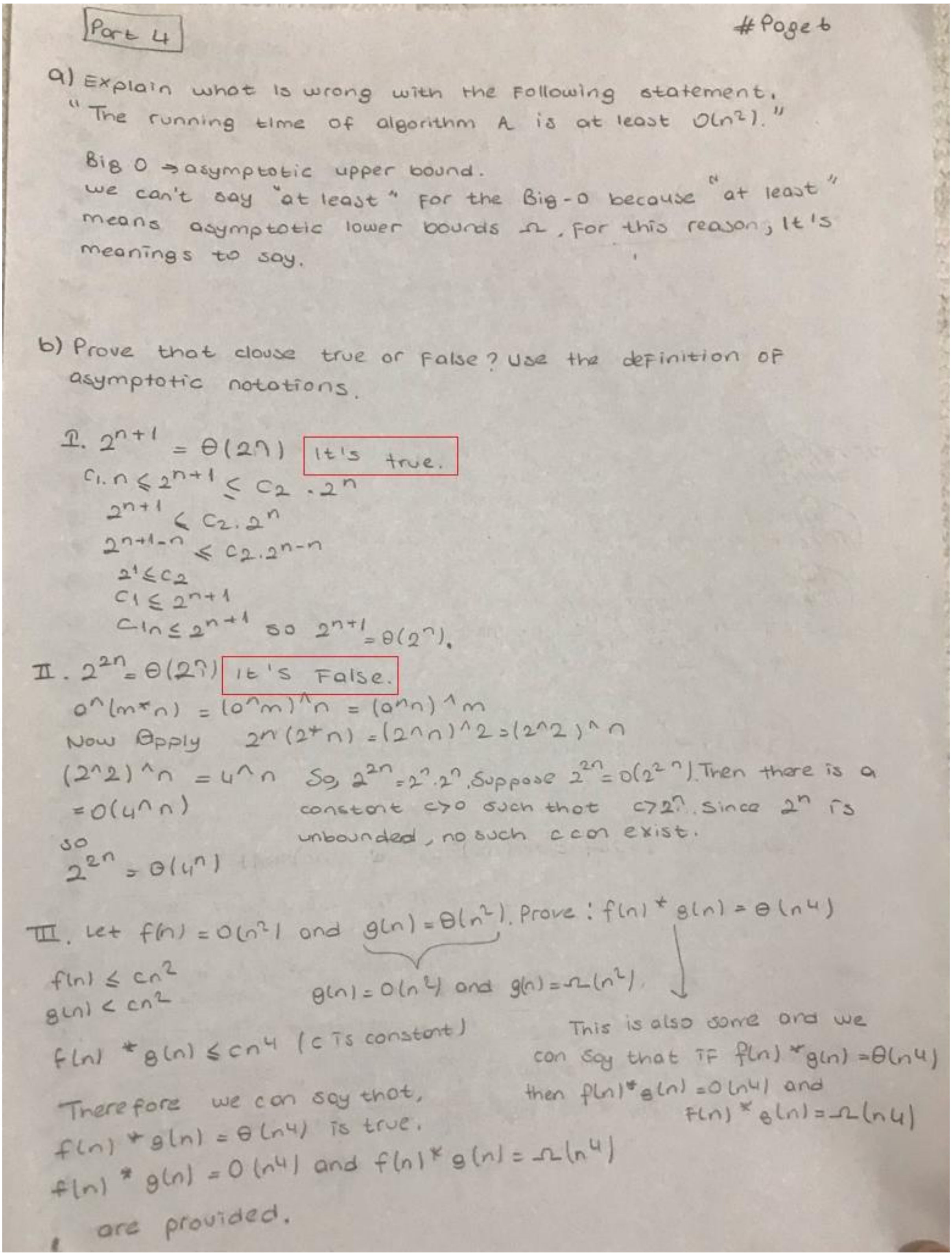
$T(n) = T(0) + 3n \rightarrow T(n) = O(n)$

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;
    }
}
```

Time Complexity
$T(n) = O(n * n)$

Space C
$S(n) = O(1)$

Best case: $T(n) = O(n)$
Average Case: $O(n^2)$
Worst case: $O(n^2)$



Part 4

#Page 6

a) Explain what is wrong with the following statement.
"The running time of algorithm A is at least $O(n^2)$."

Big O → asymptotic upper bound.
we can't say "at least" for the Big-O because "at least" means asymptotic lower bounds Ω, for this reason, It's meanings to say.

b) Prove that clouse true or False? Use the definition of asymptotic notations.

I. $2^{n+1} = \Theta(2^n)$ It's true.

$c_1.n \leq 2^{n+1} \leq c_2 . 2^n$

$2^{n+1} \leq c_2 . 2^n$

$2^{n+1-n} \leq c_2 . 2^{n-n}$

$2^1 \leq c_2$

$c_1 \leq 2^{n+1}$

$c_1 n \leq 2^{n+1}$ so $2^{n+1} = \Theta(2^n)$.

II. $2^{2n} = \Theta(2^n)$ It's False.

o^(m*n) = (o^m)^n = (o^n)^m

Now apply 2^(2*n) = (2^n)^2 = (2^2)^n

(2^2)^n = 4^n

= O(4^n)

so

$2^{2n} = \Theta(4^n)$

So, $2^{2n} = 2^n . 2^n$, Suppose $2^{2n} = O(2^n)$. Then there is a constant $c > 0$ such that $c > 2^n$. Since 2^n is unbounded, no such c can exist.

III. Let $f(n) = O(n^2)$ and $\underbrace{g(n) = \Theta(n^2)}$. Prove: $f(n) * g(n) = \Theta(n^4)$

$f(n) \leq cn^2$

$g(n) < cn^2$

$g(n) = O(n^2)$ and $g(n) = \Omega(n^2)$

$f(n) * g(n) \leq cn^4$ (c is constant)

This is also same and we can say that if $f(n) * g(n) = \Theta(n^4)$ then $f(n) * g(n) = O(n^4)$ and $f(n) * g(n) = \Omega(n^4)$

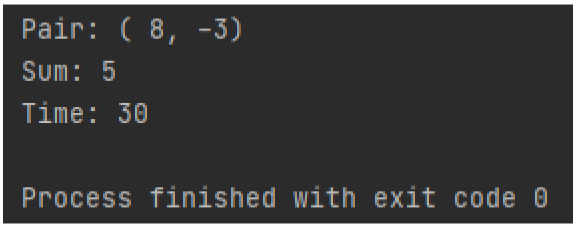
Therefore we can say that,
$f(n) * g(n) = \Theta(n^4)$ is true.
$f(n) * g(n) = O(n^4)$ and $f(n) * g(n) = \Omega(n^4)$ are provided.

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.

Code:

```java
public class Main {
    public static void main(String[] args) {
        long start = System.currentTimeMillis();
        int[] Array = {0, 7, 8, -3, 1};
        int x = 5;
        Pair(Array, Array.length, x);
        long elapsedTime = System.currentTimeMillis() - start;
        System.out.println("Time: "+ elapsedTime);

    }
    public static void Pair(int[] Array, int size, int x) {
        for (int i = 0; i < (size - 1); i++) {
            for (int j = (i + 1); j < size; j++) {
                if (Array[i] + Array[j] == x) {
                    System.out.println("Pair: ( "+ Array[i] +", "  + Array[j] + ")\n"+"Sum: "  + x  );
                    return;
                }
            }
        }
        System.out.println("There is no pair");
    }
}
```

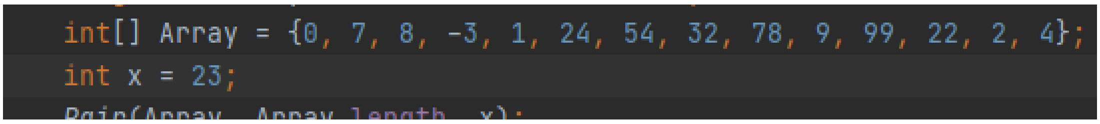
Result:



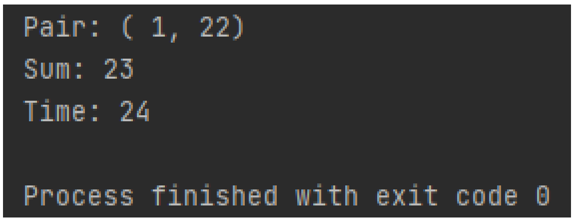
```
Pair: ( 8, -3)
Sum: 5
Time: 30

Process finished with exit code 0
```

The time complexity of the above solution **O(n^2).**

With Different Size Array:



```java
    int[] Array = {0, 7, 8, -3, 1, 24, 54, 32, 78, 9, 99, 22, 2, 4};
    int x = 23;
```



```
Pair: ( 1, 22)
Sum: 23
Time: 24

Process finished with exit code 0
```

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