Ertuğrul Koşıkçı 200104004097

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

Due Date: March 14, 2022 – 09:00 AM

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) $\log_2 n^2 + 1 = O(n)$ $\log_2 n^2 + 1 \leq c.n$ $\forall n \geq n_0$, if $c \geq 2$ and $n_0 = 1$ $\log_2 n^2 + 1 = O(n)$ is true.

√b) $\sqrt{n(n+1)} = \Omega(n)$ $\sqrt{n(n+1)} \geq c.n$ $\forall n \geq n_0$, if $c \leq 1$ and $n_0 = 1$ $\sqrt{n(n+1)} = \Omega(n)$ is true.

X c) $n^{n-1} = \theta(n^n)$ must be $n^{n-1} = O(n^n)$ and $\underbrace{n^{n-1} = \Omega(n^n)}_{\text{this is false}}$ $n^{n-1} \geq c.n^n$ c can't be smaller than $\frac{1}{n}$ statement is false

2) Order the following functions by growth rate and explain your reasoning for each of them. Use the limit method. $\lim_{N \to \infty} \frac{f(N)}{g(N)} \rightarrow$ if equals 0 $f(N) = o(g(N))$, else if equal,

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

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

a) $\theta(\log n)$

$\theta(\log n)$

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

b) $\theta(n)$

$\theta(n)$

```
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(1)
                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)
                        }
                }
        }
}
```

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

e) $\Theta(n.\log n)$

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

f) $O(n\log n)$, $\Omega(n)$

best case : $\Theta(n)$

worst case : $O(n\log n)$

average case : $O(n\log n)$, $\Omega(n)$

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

g) $\Theta(n\log n)$

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

h) $\Theta(\log n.\log n)$

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

i) $O(n)$

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

best case: $\Theta(1)$

worst case: $\Theta(n)$

average case: $O(n)$

j) $O(n^2)$

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

best case: $\Theta(1)$

worst case: $\Theta(n^2)$

average case: $O(n^2)$

4)

a) Explain what is wrong with the following statement. "The running time of algorithm A is at least $O(n^2)$". Big oh notation (O) provides upper bound. Algorith A can't be bigger than $c.n^2$ so the statement is false (if $\Omega(n^2)$ was used the statement could be true).

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

✓ I. $2^{n+1} = \Theta(\cancel{2n}\ 2^n)$ $c_1.2^n \leq 2^n.2 \leq c_2.2^n$ $\forall n \geq n_0$, if $c_1 = c_2 = 2$ and $n_0 = 1$ $2^{n+1} = \Theta(2^n)$ is true.

X II. $2^{2n} = \Theta(\cancel{2n}\ 2^n)$ $c_1.2^n \leq 2^n.2^n \leq c_2.2^n$ c_1 or c_2 can't be 2^n so the statement is false.

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

$1 \leq f(n) \leq c.n^2$ * $c_1.n^2 \leq g(n) \leq c_2 n^2$ $c_1.n^2 \leq f(n).g(n) \leq c.c_2.n^4$ we can't say $f(n)*g(n)=\Theta(n^4)$ $f(n)*g(n) = O(n^4)$ so the statement is false.

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

a) $T(n) = 2T(n/2) + n$, $T(1) = 1$ → $\Theta(n\log n)$ There is a detailed study on the following page.

b) $T(n) = 2T(n – 1) + 1$, $T(0)=0$ → $\Theta(2^n)$

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. Theoretically running time comlexity must be $\Theta(n^2)$
Test results holded with the theoretical time comlexity. Results and the function is on the following pages.

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

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

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

2. $\lim_{N\to\infty} \frac{f(N)}{g(N)} = 0 \rightarrow f(N) = o(g(N))$

$= c \neq 0 \rightarrow f(N) = \Theta(g(N))$

$= \infty \rightarrow g(N) = o(f(N))$

$\lim_{n\to\infty} \frac{10^n}{2^n} = \infty$, $\lim_{n\to\infty} \frac{2^n}{n^3} = \infty$, $\lim_{n\to\infty} \frac{n^3}{8^{\log_2 n}} = 1$ $\lim_{n\to\infty} \frac{8^{\log_2 n}}{n^2 \log n} = \infty$

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

$\lim_{n\to\infty} \frac{n^2 \log n}{n^2} = \infty$ $\lim_{n\to\infty} \frac{n^2}{\sqrt{n}} = \infty$ $\lim_{n\to\infty} \frac{\sqrt{n}}{\log n} = \infty$

then the order is: $10^n > 2^n > n^3 = 8^{\log_2 n} > n^2 \log n > n^2 > \sqrt{n} > \log n$

5. a. $T(n) = 2.T(n/2) + n$

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

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

$T(n) = 2^3.(2.T(n/2^4) + \frac{n}{2^3}) + n + n + n$

$\vdots$

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

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

$T(1) = 1$

$n.1 + \log n.n$

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

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

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

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

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

$\vdots$

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

$n = k$

$T(n) = 2^n.T(0) + 2^{n-1} \ldots 2 + 1$

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

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

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

CamScanner ile tarandı

```java
static void FindPairsOfNumbers(int numbers[], int askedSum)
{
    long start = System.currentTimeMillis();
    int foundPairs = 0;

    for (int i = 0; i < numbers.length; ++i)
    {
        for (int j = i + 1; j < numbers.length; ++j)
        {
            if(numbers[i] + numbers[j] == askedSum)
                ++foundPairs;
        }
    }
    long end = System.currentTimeMillis();
    System.out.println("For " + numbers.length + " size array " + foundPairs + " pairs found.");
    System.out.println("Elapsed Time in milli seconds: "+ (end - start));
    System.out.println();
}
```

```
For 100 size array 8 pairs found.
Elapsed Time in milli seconds: 0

For 1000 size array 435 pairs found.
Elapsed Time in milli seconds: 4

For 10000 size array 49551 pairs found.
Elapsed Time in milli seconds: 51

For 100000 size array 5505738 pairs found.
Elapsed Time in milli seconds: 4534
```

```java
static int FindPairsOfNumbersRecursion(int numbers[], int askedSum, int index1, int index2)
{
    if (index1 >= numbers.length)
        return 0;

    if (index2 >= numbers.length)
        return FindPairsOfNumbersRecursion(numbers, askedSum, index1 + 1, index1 + 2);

    if (numbers[index1] + numbers[index2] == askedSum){
            return 1 + FindPairsOfNumbersRecursion(numbers, askedSum, index1, index2 + 1);
    }

    return FindPairsOfNumbersRecursion(numbers, askedSum, index1, index2 + 1);
}
```

```
For 10 size array 0 pairs found.
Elapsed Time in micro seconds: 8

For 100 size array 7 pairs found.
Elapsed Time in micro seconds: 893
```