Order of growth exercises

1. Algorithms A and B spend exactly TA(n) = 0.1n2 log10 n and TB(n) =2.5n2 microseconds, respectively, for a problem of size n. Which algorithm is better?

Taking into account the order of grow through the Big O notation, we will realize that algorithm B, on that scale, is better, because although A is of type n log n, it has a power of two, for which increases its complexity.

Find out a problem size n0 such that for any larger size n > n0 the chosen algorithm outperforms the other.

To check which size of n makes an algorithm perform better than the other we must find the cutoff point of both functions. After that, we substitute n with smaller and bigger values than the cutoff value.

$$0.1n^2 \log_{10}(n) = 2.5n^2$$

$$n^2 \log_{10}(n) - 25n^2 = 0$$

$$n^2 * (\log_{10}(n) - 25) = 0$$

$$\log_{10}(n) - 25 = 0$$

$$n = 10^{25}$$

We will obviate the result n=0 since it doesn't give us any information.

As we can see the cutoff point is in $n=10^{25}$, so now will check smaller and bigger values.

For $n=10^{24}$:

- $TA(10^{24}) = 2.4*10^{48}$ microseconds
- $TB(10^{24}) = 2.5*10^{48}$ microseconds

For $n=10^{26}$:

- $TA(10^{26}) = 2.6*10^{52}$ microseconds
- $TB(10^{26}) = 2.5*10^{52}$ microseconds

As we can see the algorithm A is better until n takes the value of 10^{25}.

After that, every value of n greater will make the algorithm B be better.

If your problems are of the size n ≤ 109, which algorithm Will you recommend using?

As we saw earlier algorithm A is faster than algorithm B until N reaches 10^{25}, so since 109 is a smaller number, I would recommend algorithm A.

2. Algorithms A and B spend exactly TA(n) = cA n log2 n and TB(n) = cB n2 microseconds, respectively, for a problem of size n. Find the best algorithm for processing n = 220 data items if the algorithm A spends 10 microseconds to process 1024 items and the algorithm B spends 1 microsecond to process 1024 items.

First of all we have to find the values of Ca and Cb of each function to be able to calculate the complexity of each one. To do this, we will substitute n = 1024 and the respective times of each function.

After doing the math we will get that:

cA = 1/1024

cB = $1/1024^2$

Now we just substitute n=220 and check which algorithm is faster.

Once the math is done we will realize that TB(202) is faster than TA(202) since:

TA(202) = 1.6718 microseconds

TB(202) = 0.0462 microseconds

3. Algorithms A and B spend exactly TA(n) = 5·n·log10 n and TB(n) = 25·n microseconds, respectively, for a problem of size n. Which algorithm is better?

Algorithm B is better for bigger n values because it’s linear and A is better for smaller values in this case because it’s a n log n type.

To check this, we will do the same than in exercise 1 and check for the cutoff point.

Once this is done, we will get the value $n=10^5$, that means that TA is better for n values smaller than 10^5 and TA is better for bigger values.

4. One of the two software packages, A or B, should be chosen to process very big databases, containing each up to 1012 records. Average processing time of the package A is TA(n) = 0.1 n log2 n microseconds, and the average processing time of the package B is TB(n) = 5 n microseconds. Which algorithm has better performance?

To solve this we just substitute the value of n with 1012. Doing this would give us the next results:

TA(1012) = 1010.279 microseconds.

TB(1012) = 5060 microseconds.

As we can see TB is almost twice as fast as TA for a problem with an n of size 1012.

Work out exact conditions when these packages outperform each other.

To solve these we do exactly as exercises before and find the cutoff point for n.

Once we have balanced both equation we get that the cutoff point is $n=2^{50}$.

Now we check for smaller and bigger values.

For $n=2^{49}$:

- $TA(2^{49}) = 2.75*10^{48}$ microseconds
- $TB(2^{49}) = 2.81*10^{48}$ microseconds

For $n=2^{51}$:

- $TA(2^{51}) = 1.14*10^{52}$ microseconds
- $TB(2^{51}) = 1.12*10^{52}$ microseconds

As we can see TA is better until $n = 2^{49}$, for values bigger than that, TA is better.

5. One of the two software packages, A or B, should be chosen to process data collections, containing each up to 109 records. Average processing time of the package A is TA(n) = 0.001 n milliseconds, and the average processing time of the package B is TB(n) = 500 √n milliseconds. Which algorithm has better performance?

First, we calculate the cutoff point, which is $n=2.5*10^7$.Since this value is way bigger than the max value of n for our problem, we now check which algorithm is faster for smaller values.

For $n= 2.5 * 10^{10}$:

- $TA(2.5 * 10^{10}) = 2.5*10^7$microseconds
- $TB(2.5 * 10^{10}) \approx 7.9*10^7$ microseconds

TB is better for greater values of n, however as we just checked, TA is better for n values smaller than $2.5*10^{10}$, so this algorithm would better for our problem since each collection of data contains up to n=109.

6. Software packages A and B of complexity O(n log n) and O(n), respectively, spend exactly TA(n) = cA n log10 n and TB(n) = cB n milliseconds to process n data items. During a test, the average time of processing n = 104 data items with the package A and B is 100 milliseconds and 500 milliseconds, respectively. Work out exact conditions when one package actually outperforms the other and recommend the best choice if up to n = 109 items should be processed.

First we substitute n with 104 in both equations to get cA and cB.

TA(104) = 100; cA = $25/(26\log_{10}(104))$

TB(104) = 500; cB = 125/26

Now we just subsitute n with 109:

- TA(109) ≈ 105.86 microseconds
- TB(109) ≈ 524.04 microseconds

As we can see for n=109 TA is over 4 times faster than TB.

If we check the cutoff point we will realize that it is $n=104^5$, which means that TA is better for n values lower than 104^5 as we just checked with n=109, and TB is faster for greater values.

7. For each of the following operations, develop the algorithm, execute it with different parameters and indicate the order of growth:

a)

```
public int sumNumbers (int[] numbers) {
    int sum_n = 0;
    for (int i =0; i<numbers.length; i++){
        sum_n = sum_n + numbers[i];
    }
    return sum_n;
}
```

Order of growth: Linear
100 -> 0
10.000 -> 0
10.00.000 -> 11

b)

```java
public static int factorial(int val) {
    if (val==0)
        return 1;
    else
        return val * factorial( val: val-1);
}
```

Order of growth: Quadratic

1.000 -> 0

1.000.000 -> 2

c)

```java
public int maxArray (int[] values) {
    int max = values[0];
    for (int i =0; i<values.length; i++){
        if (values[i] > max) {
            max = values[i];
        }
    }
    return max;
}
```

Order of growth: Logarithmic

100 -> 0

10.000 -> 0

10.00.000 -> 11

d)

```java
public int euclidA (int x, int y) {
    if (x == y){
        return x;
    } else if (x > y) {
        return euclidA( x: x-y, y);
    }else {
        return euclidA(x,  y: y-x);
    }
}
```

A:

B:

```
public int euclidB (int x, int y) {
    while (y != 0){
        int r = x % y;
        x = y;
        y = r;
    }
    return x;
}
```