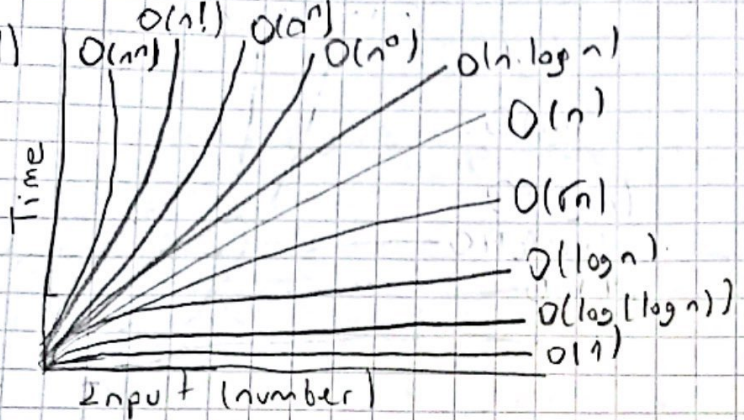
171044052
Muhammed Alperen Karasele

CSE 321 - HOMEWORK 1

1)



$T_1(n) = 3\log n + 3 \rightarrow O(\log(\log n)) < O(3\log n + 3) < O(\sqrt{n})$ +

$T_2(n) = 4\log(\log n) \rightarrow O(1) < O(4\log(\log n)) < O(\log n)$ +

$T_3(n) = n^5 + 8n^4 \rightarrow O(n \cdot \log n) < O(n^5 + 8n^4) < O(a^n)$ +

$T_4(n) = 2000n + 1 \rightarrow O(\sqrt{n}) < O(2000n + 1) < O(n \cdot \log n)$ +

$T_5(n) = \left(\frac{n}{6}\right)^2 \rightarrow O(n \cdot \log n) < O\left(\frac{n}{6}\right)^2 < O(a^n)$ +

$T_6(n) = 3^n + n^2 \rightarrow O(n^2) < O(3^n + n^2) < O(n!)$

$T_7(n) = n^n + 1000n \rightarrow On! < n^n + 1000n$

$T_8(n) = 2^n + n^3 \rightarrow O(n^3) < O(2^n + n^3) < O(n!)$

$T_2(n) < T_1(n) < T_4(n) < T_5(n) < T_3(n) < T_8(n) < T_6(n) < T_7(n)$

2) a) $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{99n}{n}$ = 99 → constant so $f(n) \in \Theta(g(n))$

L'hospital rule

b) $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{2n^4+n^2}{(\log n)^6}$ → We must approve L'hôspital 6 times

$\frac{8n^3+2n}{6(\frac{1}{n})^5}$ → it's not important take again, $\frac{32n^3}{30(\frac{1}{n})}$ ... $\frac{2^3 \cdot 2^{12} \cdot n^3}{6!} = \infty$ so $f(n) \in \Omega(g(n))$

c) $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{n^2+n}{2(4n+\log n)}$ → It's not important because small degree

$\sum_{x=1}^{n} x = \frac{n \cdot (n+1)}{2}$

$= \lim_{n\to\infty} \frac{n(n+1)}{8n} = \infty$ so $f(n) \in \Omega(g(n))$

d) $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \frac{3^n}{5^{\sqrt{n}}} = \left(\frac{3^{\sqrt{n}}}{5}\right)^{\sqrt{n}} = \infty$ so $f(n) \in \Omega(g(n))$

1 2

3) a) This algorithm is finding the occurrences of numbers in an array. If the occurence of the number is bigger the array's half size, then returns that number. If it can not found, returns -1.

Lets consider that array (1, 2, 3, 1, 1, 1). Algorithm counts occurences of 1. It founds 4. It's bigger than half size of array so it returns 1.

b) In best case, like the example above, our number is in the first element, and the algoritm works n time. So its $\Theta(n)$.

In worst case, our array will be like (2,3,4,1,1,1,1) and it works with two for loop. $\Theta(n) \cdot \Theta(\frac{n}{2})$ $= \underline{\Theta(n^2)}$

4) a) This algorithm finds max number and allocates a pointer space with max+1. Then every space in pointer represents a different number's count. For example if array is (1,2,3,1,1,1) → map[1] → represents count of 1. so its 4, map[2] represents count of 2, and it's 1. Then it returns the number which's count is bigger than half count of array. If It can not found, returns -1.

b) This algorithm work on $\Theta(n)$ time for both best case and worst case because one for loop always works for n times.

5) for time complexity, in best case both algorithm works same, but in worst case second algorithm is better. Second algorithm takes more space than first algorithm because it creates a pointer too And in some cases, pointers some elements never use. for example, (1,2,3,1,1,1) case, 1,2,3 elements useable, but the rest of the elements don't use and requires more space. There is 3 different number, but we use 3 number, rest of them is junk.

9b)

a)

```
for i=0 to n do {
    for j=0 to m do {
        multiply a[i] . b[j]
        find and store max a[i].b[i]
        }
}
```

Both best case and wors case take $\Theta(n^2)$ time. because this is nested loop

b)

```
for j=0 to j=n do {
    copy all elements of a[j]arroy to new_arroy[i]
    increase i
}

for j=0 to j=m do {
    copy all elements of b[j] arroy to new_arroy[i]
    increase i
}

for i=0 to n+m do {
    for j=i to n+m do {
        control if new_arroy[i] smaller than new_arroy[j]{
            swap new-arroy[i] and , new_arroy[j]
        }
    }
}
```

This algorithm takes $\Theta(n^2)$ time for both best and worst cases. Because it's insertion sort

c) if adding element to last {

a[n+1] = new-element.

}

if odding element to middle {

copy a[n/2] to temp
copy new-element to a[n/2]
slide after a[n/2] of arroy to right

}

if odding first {

copy a[0] to temp
copy new-element to a[0]
slide after a[0] of arroy to right

}

best case is $\Theta(1)$ if adding to last element.
worst case is $\Theta(n)$ if odding to first element.

d) Some as in C, best case is delete from last. and it takes $\Theta(1)$ time

delete a[n]

if its in middle;

delete a[n/2]
slide arroy to left

if its first element;

delete a[0]
slide arroy to left.

worst case is deleting from first element. and it takes $\Theta(n)$ time.