Ahmet Yiğit
200104004066

CSE-321 Fall 2023
HW-1

1-)

a) $f(n) = 2^n$ $g(n) = 2^{2n}$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{2^n}{2^{2n}} = \lim_{n\to\infty} \frac{2^n}{2^n 2^n} = \lim_{n\to\infty} \frac{1}{2^n} = 0$$

- The result is 0 which means g(n) faster growth rate than f(n) so

$f(n) \in O(g(n))$

b) $f(n) = n^2$ $g(n) = n^3$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{n^2}{n^2 \cdot n} = \lim_{n\to\infty} \frac{1}{n} = 0$$

- Same as a.

$f(n) \in O(g(n))$

c) $f(n) = 3n+1$ $g(n) = 2n-5$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{3n+1}{2n-5} \overset{\text{L'Hôpital}}{=} \lim_{n\to\infty} \frac{3}{2} = \frac{3}{2}$$

- It is constant so their growth rate is same.

$f(n) \in \Theta(g(n))$

d) $f(n) = 4n^2$ $g(n) = n^2$

- O notation Big O: $n > n_0$, c is constant: $0 \le f(n) \le c\, g(n)$, $0 \le 4n^2 \le c\, n^2$

if choose "$c \ge 4$", it is correct so f(n) is $O(n^2)$

- Ω notation (Big Omega) $n > n_0$, c is constant: $0 \le c\, g(n) \le f(n)$, $0 \le c \times n^2 \le 4n^2$

There is no answer so f(n) is not $\Omega(n^2)$

$f(n) \in \Theta(g(n))$

e) $f(n) = \log_2(n)$ $g(n) = \log_{10}(n)$

- We does not care 2-10 in growth rate so they have same g. rate.

$f(n) \in \Theta(g(n))$

f) $f(n) = 2^n$ $g(n) = 3^n$

1-) Big O notation

$\exists n_0$ $0 \leq f(n) \leq c^* g(n)$

- this always true, so $f(n) \in O(g(n))$

2-) Big Omega notation

$\exists n_0$ $0 \leq c^* g(n) \leq f(n)$

- No C values so $f(n) \notin \Omega(g(n))$

3-) Big Theta Not

There is NO Ω so no Theta.

So

$f(n) \in O(g(n))$

g) $f(n) = n^3$ $g(n) = 1000n^2$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{n^3}{1000n^2} = \lim_{n\to\infty} \frac{n}{1000} = \infty$$

$f(n) \in \Omega(g(n))$

- So, $f(n)$ is not in $O(g(n))$, $\Omega(g(n))$ and $\Theta(g(n))$.

h) $f(n) = 5n+4$ $g(n) = 2n+2$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{5n+4}{2n+2} \overset{\text{Hosp}}{=} \lim_{n\to\infty} \frac{5}{2} = \frac{5}{2}$$

$f(n) \in \Theta(g(n))$

? i) $f(n) = \sqrt{n}$ $g(n) = \log_2(n)$

1-) Big O notation

$\exists n_0$ $0 \leq f(n) \leq c^* g(n)$

$0 \leq \sqrt{n} \leq c \log_2(n)$

$g(n)$ more slowly than $f(n)$ so

2-) Big Omega

$\exists n_0$ $0 \leq c^* g(n) \leq f(n)$

No C answer so there is no

$f(n)$ is not in $\Omega(g(n))$

3-) Big Theta

Without omega there is no Big Theta.

So $f(n) \in O(g(n))$

j) $f(n) = 2^n$ $g(n) = 2^{n+1}$ $\in O(g(n))$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{2^n}{2^n \cdot 2} = \lim_{n\to\infty} \frac{1}{2} = \frac{1}{2}$$

$f(n) \in \Theta(g(n))$

2-)

Pn order:

Function	Big O Notation	Explanation
$1/2n$	$O(1)$	for $n>0$ therefore $\frac{1}{2n} < 1$. When "n" increases funct. approaches zero. So it is constant function.
$\log(n)$	$O(\lg n)$	$\log(n)$ functin grows slowly, much slower than polynomial ($O(n)$).
$\sqrt{n+5}$	$O(\sqrt{n})$	Square root func. grows fastr than logarithmic but slower than poly. ↳ Explained in graph in the bottom up the page. (C)
$n+1$	$O(n)$	Linear function.
$n^2 \log(n)$	$O(n^2 \log(n))$	I showed in first comp
2^n	$O(\text{exponential})$	I showed in first and second comp.
10^n	$O(\text{exponential})$	Because 10^n contain 2^n.
$n!$	$O(\text{super-exp})$	Showed on 2. -3. comp.
n^{2^n}	$O(\text{tower of exp})$	Showed on 3.

1

Comprisin of $n^2 \log(n)$ and 2^n with limit.

$$\lim_{n\to\infty} \left(\frac{n^2 \log(n)}{2^n}\right) = \frac{(\log(2^n) + 2\log(n))}{2^n} \overset{l'hosp}{=} \frac{\frac{1}{\log 2} + \frac{2}{n}}{\ln 2 \cdot 2^n \infty} = 0$$

so 2^n grows faster than $n^2 \log(n)$

2

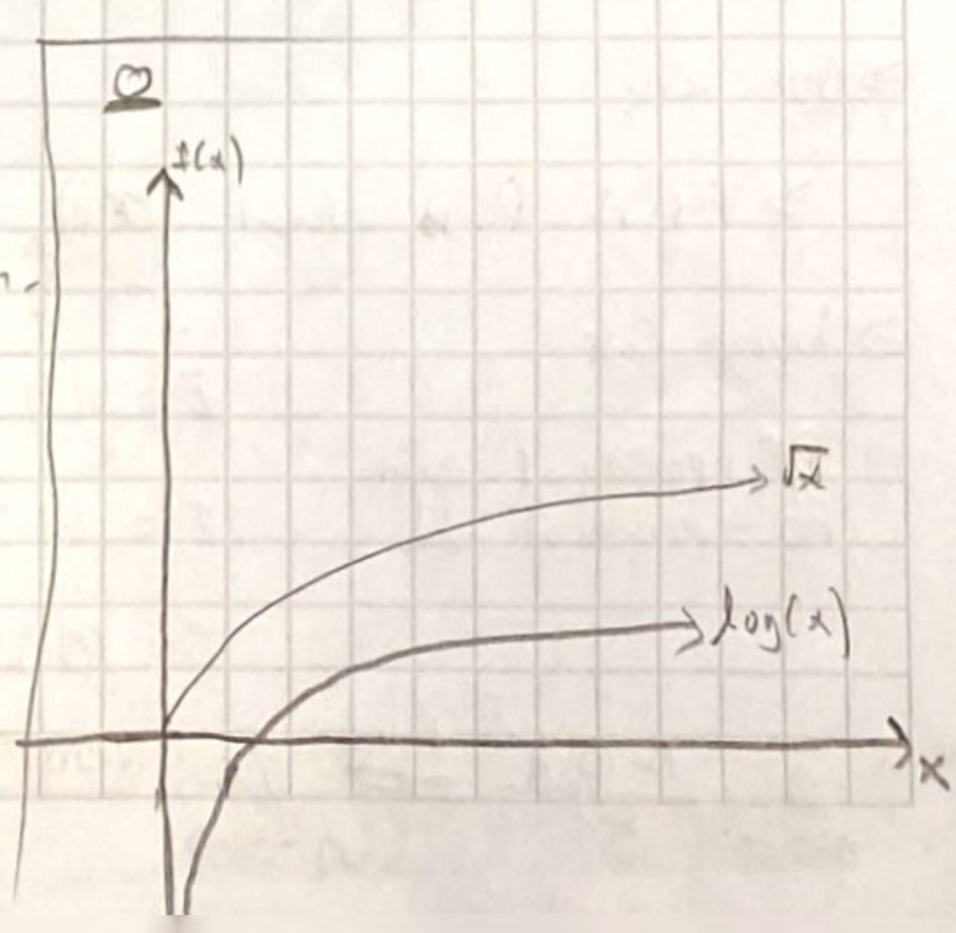
Comparision of 2^n and $n!$

$$\lim_{n\to\infty} \frac{2^n}{n!} = 0$$ use of convergence definition.

3

Comparison of $n!$ and n^{2^n}

n^{2^n} grows faster than $n!$.



4-)

```
i = 2                     O(1)
while i ≤ n:

    if i % 2 != 0:
        i = i - 1         O(1)

    else:
        i = i * i         O(1)
        i = i + i         O(1)

print(i)                  O(1)
```

• When i is even, it grows significally because it's squared in each iteration and also add 1. 3, 10, 901 so it grows exponatically.

• When i is odd, it decreases as 1. 3, 2, 1.

• In worst case, i grows exponentially and the loop continues until (i) becomes equal or greater than n.

To express i to exceed n, we can write

$2^k > n$ k = iteration number

$\log 2^k > \log(n)$

$k \log 2 > \log(n)$ ⇒ log2 is constant so k is proporsational with to log(n)

so time complexity of program is $O(\log(n))$

5-)

```
def findFirstEvenElement(array):
    """
        array: a list of elements
        return: first even number otherwise None
    """
    for element in array:

        if element % 2 == 0:
            return element
    return None
```

• This algorith will find first even element in array and returns it, if it can not find it returns None.

⇒ Worst Case

⇒ The algorithm will need to iterate through the entire array which is O(n).

⇒ Best Case

⇒ Find it first element O(1).

⇒ Average Case.

PE = probality of even
PO = probality of odd

$E = (1 \times PE) + (2 \times PO \times PE) + \dots + (n \times PO^{n-1} \times PE)$

$E = (1 \times 0.2) + (2 \times 0.8 \times 0.2) + \dots + (n \times 0.8^{n-1} \times 0.2)$

$E = (0.2)[1 + 2 \times 0.8 + \dots + n \times 0.8^{n-1}] = \frac{0.2 \times (1 - 0.8^n)}{1 - 0.8} = \frac{1 - 0.8^n}{5}$

$\lim_{n \to \infty} \frac{1 - 0.8^n}{5} \overset{\text{l'hos}}{=} \lim_{n \to \infty} \frac{(-0.8)^n \times n \times (0.8^{n-1})}{5} = 0$ from we get $O(n)$