

# **Guia 4 Complejidad Computacional 1 Cuatrimestre 2025**

**Juan DElia**

**Ejercicio 2. Argumentar por cardinalidad que hay funciones que no son time-constructible. Dar un ejemplo de una función  $f$  con  $f(n) \geq n$  que no sea time-constructible.**

Para que una función sea T.C debe ser **computable** en un tiempo, es decir debe haber una máquina de Turing que la compute en ese tiempo.

Las máquinas de Turing se pueden codificar con cadenas de 1's, es decir, son numerables.  $\Rightarrow$  Hay a lo sumo numerables funciones T.C

Pero el conjunto de funciones  $f : N \rightarrow N$  es no numerable  $\Rightarrow$  Hay funciones que no son time-constructible.

Para el ejemplo voy a aprovechar que  $se$  que halt no es computable:

- $f(n) = n^2$  Si la  $n$ -ésima máquina con entrada  $n$  termina
- $f(n) = 2n$  Caso contrario

**Ejercicio 3. Padding** Probar que si  $P = NP$  entonces  $EXP = NEXP$ . Ayuda: Para ver que  $NEXP \subseteq EXP$  tomar  $\Pi \in NTIME(2^{n^c})$  y considerar el lenguaje

$$\Pi_{pad} = \{\langle x, 01^{2^{|x|^c}} \rangle : x \in \Pi\}$$

Esta la demostracion en el apunte de Santi en la pagina 36.

## Ejercicio 5 . Probar que la función H en la demostración del Teorema de Ladner es computable en tiempo $O(n^3)$

Para analizar la complejidad voy a escribir un pseudocódigo de las funciones:

*# de 0 a i chequo, para todo x de  $0/1^{\log n}$  si la maquina i en  $i \cdot |x|^i$  pasos da lo mismo que H(n):*

```

for i de 0 a log log n: # $O(\log \log n)$ 
    valido = True
    for x cada posible: {0,1}^{\log n}: # $O(2^{\log n})=O(n)$ 
        v1 = simular Mi(x) por  $i \cdot |x|^i$  pasos # $O(i \cdot |x|^i) = O((\log \log n) \cdot (\log n)^{\log \log n})$ 
        v2 = SATH(x) #ver complejidad...
        if(v1 != v2):
            valido = falso
    if valido:
        return True
return log log n

```

Hasta aca tendríamos:  $O(\log \log n \cdot (n \cdot (n + \text{complejidad sat h})))$ .

Analicemos Sath, de vuelta pseudocódigo dudoso: (sath es  $\phi 01^{n^{H(n)}}$  phi tiene que ser satisfacible y n es la longitud de phi) En la funcion llamo k a ese n para no mezclarme con la complejidad de arriba que esta respecto de un n

```

Sath(x):
    k = calcular_longitud_phi() #  $O(|x|) = O(\log n)$ , es recorrer la palabra hasta llegar a phi
                                # y la palabra es de longitud log n a lo sumo
    phi = x desde 0 hasta k #  $O(k) = O(\log n)$ 
    es_sat = SAT(phi) #  $O(2^{|phi|}) = O(2^{\log n}) = O(n)$ 
    #queda ver que al la longitud de los 1's del final es  $k^{H(n)}$ 
    m = cantidad_de_unos_al_final(x) # se ve que es a lo sumo  $O(\log n)$ 
    return es_sat and (m == k^{H(k)}) #  $H(k)$  puedo asumir que es  $O(1)$ ? Ya voy a haber calculado  $H(k)$ 
                                     #porque k es a lo sumo log n, una vez que haya hecho  $H(k)$ 
                                     #puede estar memoizada asi que asintoticamente es  $O(1)$ 

```

Si mi asuncion del final con  $H(k)$  es cierta  $SAT_h$  es  $O(n)$

Entonces la complejidad final de H(n) es:

$$O(\log \log n \cdot (n \cdot (n + n))) = O(\log \log n \cdot n^2) = O(n^3)$$