Guia 4 Complejidad

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 funcion sea T.C debe ser **computable** en un tiempo, es decir debe haber una maquina de turing que la compute en ese tiempo.

Las maquinas 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 ele ejemplo voy a aprovechar que se que halt no es computable:

- $f(n) = n^2$ Si la n-esima maquina 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 pseudocodigo de las funciones:

```
# de 0 a i chequo, para todo x de 0/1^logn si la maquina i en i.|x|^i pasos da lo mismo que
H(n):
    for i de 0 a log log n: #O(loglogn)
        valido = True
        for x cada posible:  {0,1}^logn: #O(2^logn)=O(n)
            v1 = simular Mi(x) por i.|x|^i pasos #O(i.|x|^i) = O((log log n) . (log n ^ (lo
            v2 = SATh(x) #ver complejidad...
            if(v1 != v2):
                valido = falso
        if valido:
            return True
    return log log n
```

Hasta aca tendriamos: O(log log n . (n.(n+complejidad sat h))).

Analicemos Sath, de vuelta pseudocodigo 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
                                # 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 calc
                                      #porque k es a lo sumo log n, una vez que haya hecho
                                      #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 . (n.(n+n))) = O(log log n . n^2) = O(n^3)