ADA - Tarea 1

Xavier Garzón López

Agosto 2019

1. Código de honor

Como miembro de la comunidad académica de la Pontificia Universidad Javeriana Cali, los valores éticos y la integridad son tan importantes como la excelencia académica. En este curso se espera que los estudiantes se comporten de manera ética y honestamente, con los más altos niveles de integridad escolar. En particular, se asume que cada estudiante adopta el siguiente *código de honor*:

Como miembro de la comunidad académica de la Pontificia Universidad Javeriana Cali me comprometo a seguir los más altos estándares de integridad académica.

Integridad académica se refiere a ser honesto, dar crédito a quien lo merece y respetar el trabajo de los demás. Por eso es importante evitar plagiar, engañar, 'hacer trampa', etc. En particular, el acto de entregar un programa de computador ajeno como propio constituye un acto de plagio; cambiar el nombre de las variables, agregar o eliminar comentarios y reorganizar comandos no cambia el hecho de que se está copiando el programa de alguien más. Para más detalles consultar *Reglamento de Estudiante*, sección VI.

2. Ejercicios 3 y 5 (Kleinberg & Tardos, páginas 67 y 68)

Orden asintótico

$$f_1(n) = n^{2.5}$$
$$f_2(n) = \sqrt{2n}$$
$$f_3(n) = n + 10$$
$$f_4(n) = 10^n$$
$$f_5(n) = 100^n$$
$$f_6(n) = n^2 \log n$$

Orden ascendente según tasa de crecimiento: *f2(n), f3(n), f6(n), f1(n), f4(n), f5(n)*

Eficiencia algorítmica

a)

$$\log_2 f(n) \text{ is } O(\log_2 g(n))$$

Sea *f(n)=2* y *g(n)=1*

$$log_2 2 \leq c \cdot log_2 1$$
$$1 \leq c \cdot 0$$
$$1 \leq 0$$

Se puede concluir que ningún ***c*** es capaz de satisfacer la definición

b)

$$2^{f(n)} \text{ is } O(2^{g(n)})$$

Sea *f(n)=2n y g(n)=n.*

$$2^{2n} \leq c \cdot 2^n$$
$$2n \cdot log_2 2 \leq log_2 c + n \cdot log_2 2$$
$$2n \leq log_2 c + n$$
$$n \leq log_2 c$$

Se puede concluir por contradicción que ***n*** no es menor o igual que ***ln(c)***, por tanto, la afirmación es falsa.

c)

$$f(n)^2 \text{ is } O(g(n)^2)$$

$$0 \leq f(n) \leq c \cdot g(n)$$
$$0^2 \leq f(n)^2 \leq (c \cdot g(n))^2$$
$$0 \leq f(n)^2 \leq c^2 \cdot g(n)^2$$

La definición de O grande se mantiene, por tanto es verdad

3. **Ejercicio 8 (Kleinberg & Tardos, páginas 68 y 69)**

   Safe Rungs

   a) Se deben marcar todos los escalones múltiplos de $\sqrt{k}$ (si no es exacto el resultado se debe tomar la parte entera) hasta *k.* Después se lanza una jarra desde la posición más alta, si esta no se rompe, se intenta arrojar otra desde un escalón más alto y verificar si resiste o no. En caso contrario, se debe bajar al siguiente escalón múltiplo de *k* y lanzar una jarra y verificar si se rompe o no.

4. **Ejercicio 9 (Erickson, páginas 49 y 50)**
   Pancakes

   a)

   Entradas: arreglo que almacena el tamaño de cada panqueque
   Saldida: arreglo que almacena los tamaños de los panqueques ordenados
   Invariantes: el arreglo no puede ser vacío

   Se debe buscar el panqueque más grande y llevarlo al extremo superior de la pila. Esto se hace poniendo la espátula en la base del panqueque más grande y darle vuelta junto a todos los panqueques encima de él. Una vez arriba, solo basta con voltear todos los panqueques de la pila (no voltear los que ya están ordenados).
   Una vez se tiene el panqueque más grande en la base, este se omite y se repite el proceso anterior.

   Peor de los casos: **n** vueltas

   b) Se debe realizar el mismo algoritmo mencionado anteriormente, solo con la diferencia de que cuando un panqueque grande esté en el extremo superior de la pila se debe revisar cuál lado está quemado, y de ser necesario, voltearlo antes de llevarlo a la parte inferior.

   Peor de los casos: **2n** vueltas

5. **Ejercicio 13 (Erickson, página 51)**
   Inversions

   Entradas: arreglo que indica el tamaño de cada panqueque
   Saldida: número de inversiones realizadas
   Invariantes: el arreglo no puede ser vacío

   Se parte del algoritmo Merge-Sort. Justo antes de que Merge() retorne los arreglos ordenados se debe saber cuántas inversiones son posibles en ese momento. Partimos de la definición del problema que menciona que si A[i]<A[j] no van a haber inversiones posibles. Como justo en este momento sabemos que los arreglos ordenados, podemos calcular la cantidad de inversiones posibles. La cantidad de inversiones son todas aquellas que cumplan la condición anterior (A[i]>A[j]). Al final, simplemente se retorna este resultado y se empieza a sumar recursivamente con los demás resultados.

```
while i<=mid and j<=hi:
        if arr[i]<=arr[j]:
            i+=1
        else:
            cnt+=(mid-i+1)
            j+=1
```