Introducción a la Programación

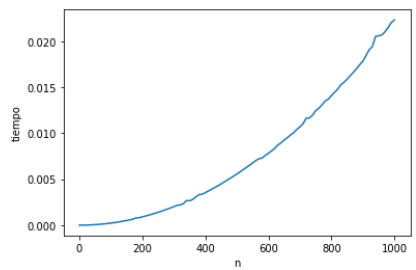
Prof. Agustín Gravano

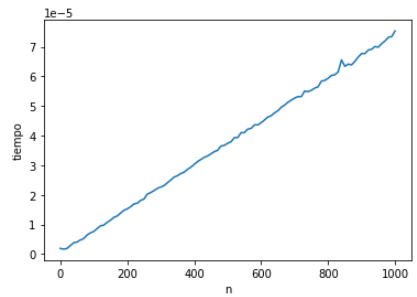
Primer semestre de 2022

Clase teórica 15: Complejidad algorítmica

¿Cuánto tiempo tarda la ejecución de un programa?

```
1   def sumatoria(n:int) -> int:
2       ''' Requiere: n>0
3           Devuelve: la suma de los enteros entre 1 y n (inclusive).
4       '''
5       vr:int = 0
6       i:int = 1
7       while i <= n:
8           vr = vr + i
9           i = i + 1
10      return vr
11
12  def lista_sumatorias_v1(n:int) -> List[int]:
13      ''' Requiere: n>=0
14          Devuelve: lista de longitud n, que en cada posición i tenga la sumatoria entre 1 e i+1
15      '''
16      vr:List[int] = []
17      i:int = 1
18      while i<=n:
19          vr.append(sumatoria(i))
20          i = i + 1
21      return vr
22
23  def lista_sumatorias_v2(n:int) -> List[int]:
24      ''' Requiere: n>=0.
25          Devuelve: lista de longitud n, que en cada posición i tenga la sumatoria entre 1 e i+1
26      '''
27      vr:List[int] = []
28      i:int = 1
29      s:int = 1
30      while i<=n:
31          vr.append(s)
32          i = i + 1
33          s = s + i
34      return vr
```





Problema 1: Contar ases

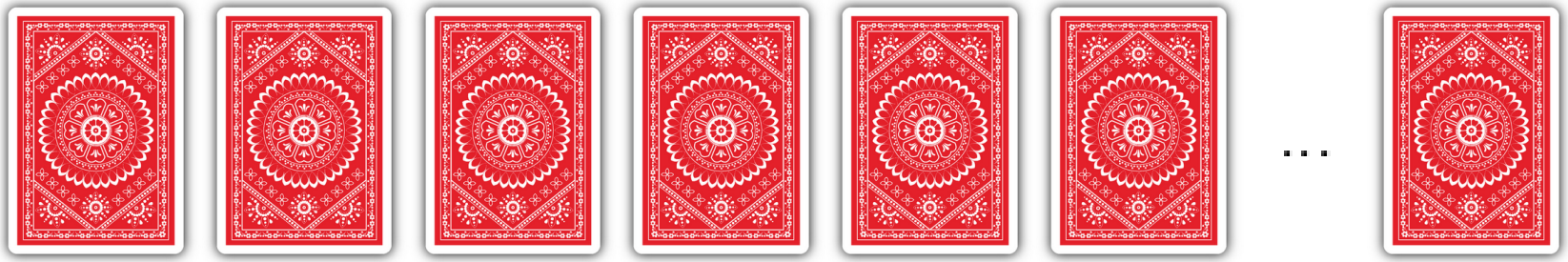
¿Cuántos pasos demandará contar los ases en estos 5 naipes?
(Supongamos que los naipes provienen de varios mazos mezclados.)

¿Y si tenemos que contar los ases en 10 naipes?

¿Y en 50 naipes?

Problema 1: Contar ases

En general, ¿cuántos pasos demanda contar ases en N naipes?



La cantidad de pasos necesarios crece en forma lineal respecto del tamaño de la entrada del problema (la cantidad de naipes).

Problema 1: Contar ases - Posible algoritmo

```
1  Contar ases en N naipes:
2      cant = 0
3      i = 0
4      mientras i < N:
5          dar vuelta el naipe i
6          si el naipe es un as:
7              cant = cant + 1
8          i = i + 1
9      devolver cant
```

Las líneas 5-8 son lo que veníamos llamando un "paso".

La ejecución individual de cada "paso" no depende de N.

El ciclo repite esas operaciones N veces.

Decimos que el algoritmo tiene $O(N)$, o bien "orden lineal".

Contar las apariciones de un elemento en una lista

```
1  def contar(elem:int, lista:List[int]) -> int:
2      ''' Cuenta las apariciones de elem en la lista. '''
3      cant:int = 0                          O(1)
4      i:int = 0                             O(1)
5      while i < len(lista):                 len(lista) iteraciones
6          if lista[i]==elem:                O(1)
7              cant = cant + 1               O(1)
8          i = i + 1                         O(1)
9      return cant                           O(1)
```

Importante: Cada operación de las líneas 6-8 va a ejecutarse en cierto tiempo fijo, que no depende de `len(lista)`.

Son operaciones simples: expresiones de tipos básicos, accesos a variables, etc. Decimos que tienen $O(1)$, u "orden constante".

El `while` repite `len(lista)` veces ese bloque de operaciones constantes. Entonces, el algoritmo tiene $O(\texttt{len(lista)})$.

Contar las apariciones de un elemento en una lista

El mismo razonamiento vale para este código parecido:

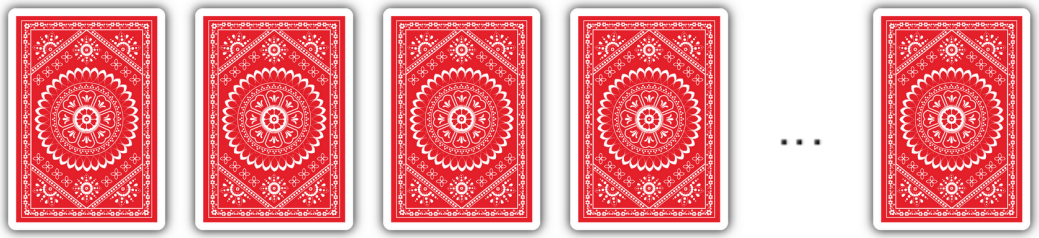
```
1  def contar(elem:int, lista:List[int]) -> int:
2      ''' Cuenta las apariciones de elem en la lista. '''
3      cant:int = 0                          O(1)
4      for x in lista:                       len(lista) iteraciones
5          if x==elem:                       O(1)
6              cant = cant + 1               O(1)
7      return cant                           O(1)
```

El `for` ejecuta `len(lista)` veces las líneas 5-6, que consisten en operaciones simples (todas O(1)).

Por lo tanto, este algoritmo también tiene $O(\texttt{len(lista)})$.

Ejercicio

¿Cuántos pasos demandará determinar, en el peor caso, si el as de corazones (A♡) está presente entre N naipes?



1. Escribir un algoritmo en pseudocódigo que busque un A♡ en N naipes. Analizar su complejidad algorítmica en el peor caso.
2. Si se corta la ejecución al encontrar un A♡, ¿cambia la complejidad en el peor caso?
3. Escribir en Python la siguiente función:

   ```
   buscar(elem:int, lista:List[int]) -> bool
   ```

   que devuelve `True` si `elem` está en la `lista`, y `False` en caso contrario.

Problema 2: Determinar el palo

Nos dicen que estos 5 naipes tienen el mismo palo (♡, ♣, ♢ o ♠).
¿Cuántos pasos demandará averiguar de qué palo se trata?

¿Y lo mismo, pero con 10 naipes?

¿Y con 50 naipes?

Problema 2: Determinar el palo

En general, si nos dicen que los N naipes tienen el mismo palo, ¿cuántos pasos demandará averiguar de qué palo se trata?

Siempre alcanza con un solo paso: mirar el palo del primer naipe (o de cualquier otro). Esto es independiente de la cantidad de naipes.

Por eso, decimos que la cantidad de pasos es constante respecto del tamaño de la entrada del problema: $O(1)$.

Repaso de la clase de hoy

- Complejidad algorítmica.
- Algoritmos con complejidad constante: O(1)
- Algoritmos con complejidad lineal: O(N)

Con lo visto, pueden ir *pensando* hasta el Ejercicio 5 (inclusive) de la Guía de Ejercicios 7. Después de la próxima clase tendrán todos los elementos para resolverlos.

Bibliografía: capítulo 2 de "The Algorithm Design Manual", S. Skiena, 2da edición, Springer (disponible en el campus).