



Para los siguientes ejercicios recomendamos utilizar `contruir_vector`, que recibe un entero  $n$  y un `string` *disposicion* y devuelve un vector de  $n$  elementos en la disposición especificada (opciones para disposición: “asc”, “desc”, “azar”, “iguales”)

### Ejercicio 1. Medición de tiempos

En el archivo `ejercicios.cpp` encontrarán la implementación de `hayDuplicados(vector<int> v)` que dado un vector  $v$  demora aproximadamente  $c \cdot n^2$  segundos en terminar en el peor caso (siendo  $n$  el tamaño del vector de entrada y  $c$  una constante que no depende de  $n$ ). Es decir, su tiempo de ejecución de peor caso pertenece a  $O(n^2)$ .

Calcular cuántos segundos demoraría (sin correrlo!) aproximadamente en el peor caso si  $n = 2200000000$  (# de usuarios de Facebook) y para  $c=0,1$ ;  $0,01$ ;  $1$ ;  $10$ ;

### Ejercicio 2. Graficar

Escribir los tiempos calculados en un archivo con  $n$  desde 1 hasta 10000 (paso 500). Este archivo deberá contener una columna para  $n$  (con encabezado “n”) y una columna (con cualquier encabezado declarativo) para el tiempo para dicho  $n$  en segundos.

Generar un gráfico con eje  $x$ :  $n$  y eje  $y$ : tiempo de ejecución en segundos. Para graficar, pueden utilizar cualquier programa que deseen o el script provisto por la cátedra. Para ver el modo de uso del script: `python3 graficador.py --help`. Si se desea visualizar más de una curva a la vez, pueden agregar columnas al archivo con nombres declarativos como encabezado de la columna.

### Ejercicio 3. Opcional

Modificar las guardas del programa del punto 1 para que los ciclos terminen en caso de haber encontrado un duplicado. Medir nuevamente. ¿Cambió la complejidad?

### Ejercicio 4.

Escribir programas con tiempo de ejecución de peor caso pertenecientes a  $O(1)$ ,  $O(n)$ ,  $O(n^3)$  y  $O(n \cdot \log(n))$ . Utilizar si se desea la función provista en el ejercicio 1 y la función `busqueda_binaria(vector<int> v, int e)` incluida en los archivos de la clase.

1. Medir el tiempo en segundos y completar la siguiente tabla

	$O(1)$	$O(n)$	$O(n^3)$	$O(n \cdot \log(n))$
<b>n=100</b>				
<b>n=1000</b>				
<b>n=10000</b>				
<b>n=100000</b>				

### Ejercicio 5. STL

Realizar simulaciones para estimar la complejidad temporal de las siguientes operaciones sobre vectores en C++. Chequear los resultados con lo que indica la especificación de cada operación en [www.cplusplus.com](http://www.cplusplus.com)

1. `v.size()`
2. `v.push_back(e)`  
Tip: Analizar que pasa cuando se supera el tamaño del vector en memoria <http://www.cplusplus.com/reference/vector/vector/capacity/> <http://www.cplusplus.com/reference/vector/vector/reserve/>
3. `v.pop_back()`
4. `v[i]` (¿cambia la complejidad estimada según el valor de  $i$ ?)
5. `v[i] = e;` (¿cambia la complejidad estimada según  $i$ ?)
6. `v.flip()` (sólo para  $v$  de tipo `vector<bool>`)
7. `v.clear()`

¿Por qué el `push_back` tiene saltos?

### Ejercicio 6.

Realizar simulaciones y gráficos para determinar el tiempo de ejecución de **peor caso** del programa `algunSubconjSuma` (se encuentra en `ejercicios.cpp`).