

UNIVERSIDAD NACIONAL DE SAN CRISTÓBAL DE HUAMANGA
FACULTAD DE INGENIERÍA DE MINAS, GEOLOGÍA Y CIVIL
ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMAS



ESTRUCUTURA DE DATOS (IS – 241)

TAREA N° 02 -Eficiencia de un Algoritmo (*optimización*)

Encontrando Tiempos de Ejecución de un
Algoritmo

Docente : Ing. Elinar Carrillo Riveros

Estudiante : Isaías Ramos López

Cod. Estudiante : 27202506

Fecha : 29 de Septiembre de 2022

AYACUCHO – PERÚ

2022

1 Ejercicio N° 1

1.1 Importacion




```
1 import time;
```

Fig. 1: Importacion de paquete `time`.

1.2 Definiendo Funciones

1.2.1 Funcion Lineal

Nuestra función lineal, calculara la suma de n priemros números consecutivos mediante la estructura condicional for, iterando $n + 1$ veces, realizando una suma acumulativa.



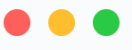
```
1 def sum_lineal(n):
2     sum = 0;
3     for i in range (1, n+1):
4         sum +=i;
5     return sum;
```

Fig. 2: Definición de la función `sum_lineal`.

1.2.2 Funcion Constante

La función constante realizara la suma de los primeros número enteros consecutivos mediante la fórmula siguiente:

$$1 + 2 + 3 + \dots + n = \frac{n(n + 1)}{2}$$



```
1 def sum_constant(n):
2     return (n*(n+1))/2;
```

Fig. 3: Definición de la función `sum_constante`.

1.3 Inicializando y Asignando Valores

Usaremos 5 eventos para poder experimentar el tiempo de ejecución y así poder determinar comprobar cual de los algoritmos es más eficiente, óptimo. Los eventos serán 1000000, 10000000, 100000000, 1000000000 y 10000000000, almacenaremos estos valores en una arreglo de 5 elementos (ver **Fig 4**).



```
1 array_amount = [1000000, 10000000, 100000000, 1000000000, 10000000000];
```

Fig. 4: Para el evento 1

1.4 Programa Principal

Esta parte de nuestro código ejecutará las funciones definidas anteriormente definidas para los cinco eventos. Vamos a recorrer nuestro arreglo elemento por elemento de la manera mas sencilla podriamos usar un método para recorrer todo nuestro arreglo y simplificar aún más el código.



```
1 for i in range(0, len(array_amount)):
2     t0 = time.time();
3     sum1 = sum_lineal(array_amount[i]);
4     t1 = time.time();
5     sum2 = sum_constant(array_amount[i]);
6     t2 = time.time();
7     print('sum1: %d, sum2: %d, time1: %f, time2: %f' % (sum1, sum2, t1-t0, t2-t1));
```

Fig. 5: Programa principal.

1.5 Ejecución

Ejecutando nuestro código, hay que tener paciencia para obtener los resultados.

```
sum1: 500000500000, sum2: 500000500000, time1: 0.055003, time2: 0.000000
sum1: 50000005000000, sum2: 50000005000000, time1: 0.611047, time2: 0.000000
sum1: 5000000050000000, sum2: 5000000050000000, time1: 5.868471, time2: 0.000999
sum1: 500000000500000000, sum2: 500000000500000000, time1: 63.863117, time2: 0.000000
sum1: 50000000005000000000, sum2: 50000000005000003584, time1: 1929.456874, time2: 0.000000
```

1.6 Tabla de Resultados

Evento	Tiempo Función Lineal	Tiempo Función Constante
1	0.055003	0.000000
2	0.611047	0.000000
3	5.868471	0.000999
4	63.863117	0.000000
5	1929.456874	0.000000

1.7 Conclusiones

- ① La **función constante** (Fig.3) es más eficiente, óptimo que la **función lineal** (Fig.2).
- ② En el primer evento la diferencia no se llega a apreciar, a medida que el número aumenta la diferencia se llega a notar, teniendo diferencias muy grandes.
- ③ Es recomendable optimizar nuestros algoritmos para en un futuro no tengamos fallos, colapsos, etc. Sería un gran problema.

1.8 Comentario

Al momento de tomar la foto del ejemplo no logré capturar el código, pero intenté hacer uno similar, tomando referencia lo aprendido en el curso de Algoritmos, que llevé con el Ing. Manuel Lagos, en la parte final del curso realizamos un tema similar.