Cuestión de tiempo

Alberto Almagro   Juan Carlos Llamas   Jaime Martínez
Santiago Mourenza   Pedro Palacios   Enrique Rey

Resumen

En este estudio analizamos tanto teóricamente como experimentalmente la eficiencia de dos algoritmos que resuelven el problema `316 - Racha afortunada`, uno de ellos con solución `TLE` (Time Limit Exceeded) y el otro `AC` (Accepted).

1. Análisis teórico

1.1. Solución ineficiente

Algoritmo 1: Solución ineficiente (TLE)

```
void resolverCaso() {
    int lista[100000]; int cont;
    cin >> cont;
    for (int i = 0; i < cont; i++)
        cin >> lista[i];
    int max = 0, p, q;
    for (int i = 0; i < cont; i++) {
        for (int j = i; j < cont; j++) {
            int suma = 0;
            for (int k = i; k <= j; k++) {
                suma += lista[k];
            }
            if (  (suma > max) && (lista[i] != 0) &&
                  (lista[j] != 0)  ){
                max = suma; p = i; q = j;
            }
        }
    }
    cout << p + 1 << " " << q + 1 << endl;
}
```

Veamos a que orden pertenece $T(n)$, el orden del algoritmo, cuyo tamaño se mide en $n := \textit{cont}$:

$$T(n) \in \Theta\left(\sum_{i=0}^{n-1} 1 + \sum_{i=0}^{n-1}\sum_{j=i}^{n-1}\sum_{k=i}^{j} 1\right) = \Theta\left(n + \frac{1}{6}n\left(n^2 + 3n + 2\right)\right) = \Theta\left(n^3\right)$$

Por tanto, el algoritmo ineficiente tiene coste cúbico, muy superior al lineal esperado por el juez, por lo que obtenemos el resultado `TLE`.

1.2. Solución eficiente

Algoritmo 2: Solución eficiente (AC)

```
void resolverCaso() {
    int cont, num, ini = 1, fin = cont;
    int i, f, hasta = -10001, hasta2 = -10001;
    cin >> cont;
    for (int j = 0; j < cont; j++) {
        cin >> num;
        if (hasta <= 0) {
            hasta = num; fin = ini = j + 1;
        } else {
            hasta += num; fin++;
        } if (hasta2 < hasta) {
            hasta2 = hasta; i = ini; f = fin;
        } else if(  (fin - f < ini - i) &&
                    (hasta == hasta2)   ) {
            i = ini; f = fin;
        }
    }
    cout << i << " " << f << endl;
}
```

Veamos a que orden pertenece $T(n)$, el orden del algoritmo, cuyo tamaño se mide en $n := \textit{cont}$:

$$T(n) \in \Theta\left(\sum_{i=0}^{n-1} 1\right) = \Theta\left(n\right)$$

Por tanto, el algoritmo eficiente tiene coste lineal, lo esperado por el juez, por lo que obtenemos el resultado `AC`.

2. Análisis experimental

Hemos generado vectores aleatorios de tamaño $n \in \{100i : 1 \leq i \leq 20\}$ y hemos ejecutado ambos algoritmos con 5 repeticiones para eliminar posible ruido estadístico. Los datos obtenidos son los siguientes:

Tamaño del vector	**Tiempo eficiente (ms)**	**Tiempo ineficiente (ms)**
100	0.021842	0.293704
200	0.015466	1.94641
300	0.020868	6.27136
400	0.026655	14.7729
500	0.032554	35.7921
600	0.040884	62.5808
700	0.044481	89.6352
800	0.050529	123.741
900	0.058400	165.654
1000	0.063282	220.49
1100	0.070667	294.425
1200	0.076716	380.805
1300	0.080629	482.429
1400	0.100300	600.234
1500	0.094642	738.071
1600	0.100410	901.872
1700	0.119105	1071.45
1800	0.112296	1274.85
1900	0.147598	1523.1
2000	0.123992	1738.03

Figura 1: Tiempo de ejecución algoritmo eficiente

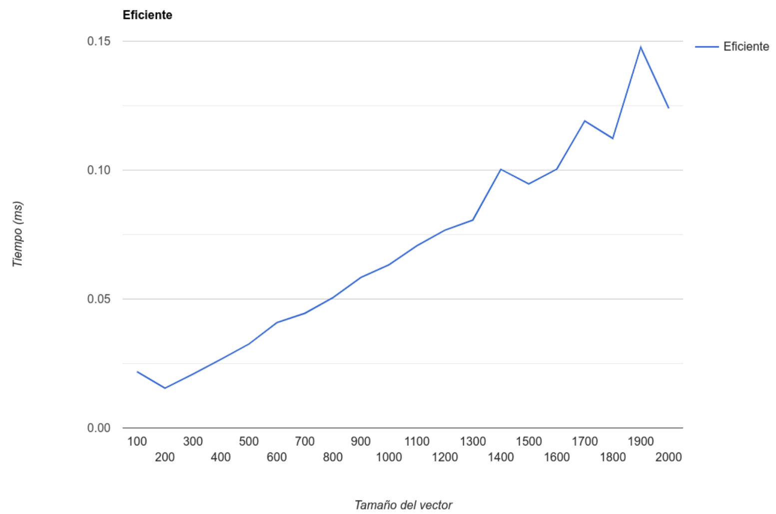


Figura 2: Tiempo de ejecución algoritmo ineficiente

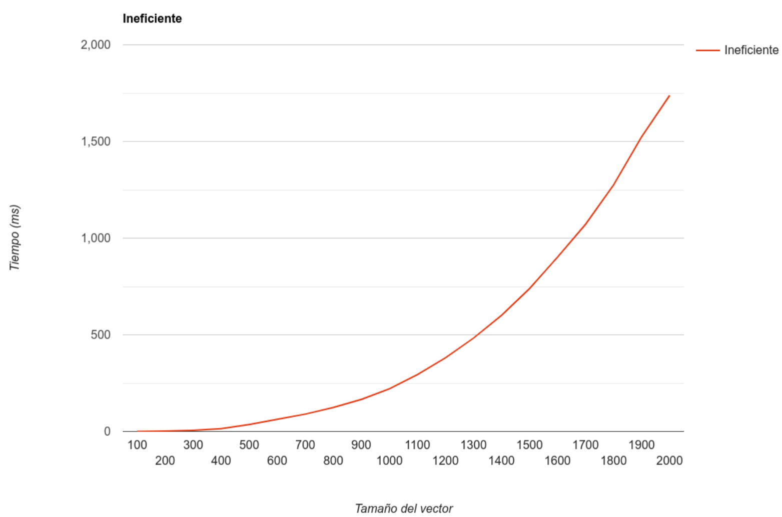
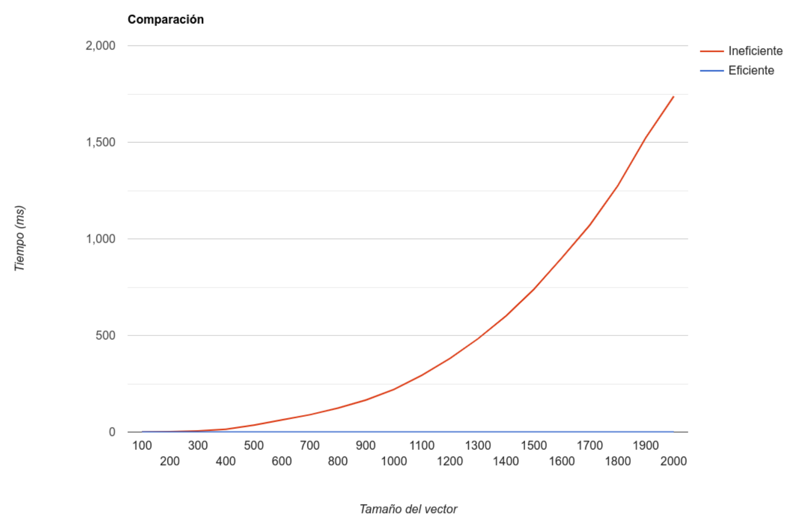


Figura 3: Comparación entre los tiempos de ejecución



3. Conclusión

Como muestran los datos, el tiempo de ejecución del algoritmo cúbico crece radicalmente más rápido que el tiempo de ejecución del algoritmo lineal a medida que crece el tamaño del vector de entrada. Por tanto, queda patente la importancia de la obtención de algoritmos eficientes.