Algoritmos

Carlos E. Alvarez[1].

[1]Dep. de Matemáticas aplicadas y Ciencias de la Computación, Universidad del Rosario

2019-II

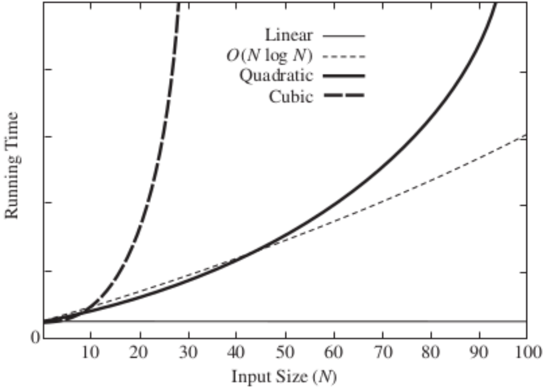
Universidad del Rosario | MACC Matemáticas Aplicadas y Ciencias de la Computación

Tiempo de ejecución

Input Size	Algorithm Time 1 $O(N^3)$	2 $O(N^2)$	3 $O(N \log N)$	4 $O(N)$
$N = 100$	0.000159	0.000006	0.000005	0.000002
$N = 1{,}000$	0.095857	0.000371	0.000060	0.000022
$N = 10{,}000$	86.67	0.033322	0.000619	0.000222
$N = 100{,}000$	NA	3.33	0.006700	0.002205
$N = 1{,}000{,}000$	NA	NA	0.074870	0.022711

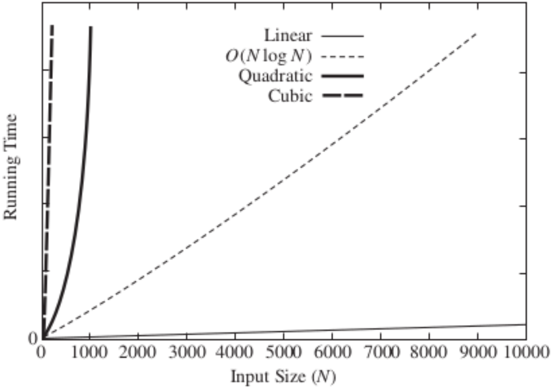
Tiempos de ejecución de 4 algoritmos para resolver el problema de la subsecuencia de suma máxima.

Tiempo de ejecución



T vs. N para varios algoritmos con diferente O.

Tiempo de ejecución



T vs. N para varios algoritmos con diferente O.

Tiempo de ejecución

Ejemplo

```c
int sum(int n){
  int partialSum;

  partialSum = 0;
  for(int i = 1; i <= n; i++)
    partialSum += i * i * i;
  return partialSum;
}
```

Tiempo de ejecución

Estimaremos el tiempo de ejecución $T(n)$ asumiendo el modelo RAM de computación:

1. La declaraciones no las contamos
2. Las líneas 4 y 7 cuentan por una unidad cada una
3. La línea 6 tiene dos productos, una suma y una asignación y se repite `n` veces. Cuenta por `4n` unidades
4. En la linea 5 se inicializa `i` una sola vez y luego se compara (`i<=n`) `n+1` veces y se incrementa (`i++`) `n` veces. Cuenta entonces por `2n+2`

El costo total es `6n+4`, pero dado que no tomamos en cuenta constantes y solo el orden mas alto del polinomio, tenemos que $T(n) = O(n)$.

Universidad del Rosario | MACC Matemáticas Aplicadas y Ciencias de la Computación

Tiempo de ejecución

Ejercicio: Analice el tiempo de ejecución de

```
for(i = 0; i < n; ++i){
  for(j = 0; j < n; ++j)
    k++;
}
```

Tiempo de ejecución

Ejercicio: Analice el tiempo de ejecución de

```
for(i = 0; i < n; ++i)
  a[i] = 0;
for(i = 0; i < n; ++i){
  for(j = 0; j < n; ++j)
    a[i] += a[j] + i + j;
}
```

Alg. para la suma de secuencia con tiempo cúbico

```
int maxSubSum1(const vector<int>& a){
  int maxSum = 0;
  for(int i = 0; i < a.size(); ++i){
    for(int j = i; j < a.size(); ++j){
      int thisSum = 0;

      for(int k = i; k <= j; ++k)
        thisSum += a[k];

      if(thisSum > maxSum)
        maxSum = thisSum;
    }
  }
  return maxSum;
}
```

Alg. para la suma de secuencia con tiempo cúbico

- Sin mirar los dos ciclos externos, la línea 5 tiene un costo de 1
- Sin mirar los dos ciclos externos, las líneas 10 y 11 un costo de 2
- Sin mirar los dos ciclos externos, las líneas 7 y 8 un costo de inicialización(1) + test$(1 * (j - i + 1))$ + incremento $(1 * (j - i))$ + suma y asignación$(2 * (j - i))$. Sin constantes, esto es simplemente del orden de

$$\sum_{k=i}^{j} 1 = j - i + 1$$

Universidad del Rosario | MACC Matemáticas Aplicadas y Ciencias de la Computación

Alg. para la suma de secuencia con tiempo cúbico

- Al sumar la contribución de 7 y 8 sobre j tenemos que

$$\sum_{j=i}^{N-1}(j-i+1)=\sum_{j=i}^{N-1} j-\sum_{j=i}^{N-1} i+\sum_{j=i}^{N-1} 1,$$

con

$$\sum_{j=i}^{N-1} 1=N-i, \quad \sum_{j=i}^{N-1} i=i(N-i)$$

y

$$\begin{aligned}\sum_{j=i}^{N-1} j &= i+(i+1)+\cdots+(i+(N-1-i)) \\ &= i(N-1-i+1)+\sum_{l=1}^{N-1-i} l \\ &= i(N-i)+\frac{(N-1-i)(N-i)}{2}\end{aligned}$$

Por lo tanto

$$\sum_{j=i}^{N-1}(j-i+1) = \left[i(N-i) + \frac{(N-1-i)(N-i)}{2}\right] - i(N-i) + (N-i)$$

$$= \frac{(N-i)(N-i+1)}{2}$$

- Finalmente hay que sumar sobre i

$$\sum_{i=0}^{N-1}\frac{(N-i)(N-i+1)}{2} = \sum_{i=1}^{N}\frac{(N-i+1)(N-i+2)}{2}$$

Alg. para la suma de secuencia con tiempo cúbico

$$
\begin{aligned}
&=\frac{1}{2}\sum_{i=1}^{N}\left[i^2-(2N+3)i+(N^2+3N+2)\right] \\
&=\frac{1}{2}\left[\frac{N(N+1)(2N+1)}{6}+(2N+3)\frac{N(N+1)}{2}\right. \\
&\qquad \left.+(N^2+3N+2)N\right] \\
&=\frac{N^3+3N^2+2N}{6}
\end{aligned}
$$

Como la función obtenida es polinomial, decimos que el tiempo de ejecución del algoritmo es $\Theta(N^3)$.

Universidad del Rosario | MACC Matemáticas Aplicadas y Ciencias de la Computación

Alg. para la suma de secuencia

Note que en algoritmo anterior i y j representan los límites de la subsecuencia y luego, dados estos límites, utilizamos un tercer ciclo (k) para hacer la suma. Sin embargo, podríamos simplemente hacer la suma a medida que avanzamos el índice j.

Alg. para la suma de secuencia con tiempo cuadrático

```
int maxSubSum2(const vector<int>& a){
  int maxSum = 0;
  for(int i = 0; i < a.size(); ++i){
    int thisSum = 0;
    for(int j = i; j < a.size(); ++j){
      thisSum += a[k];

      if(thisSum > maxSum)
        maxSum = thisSum;
    }
  }
  return maxSum;
}
```

Alg. para la suma de secuencia con tiempo cuadrático

- Sin mirar los ciclos, las líneas 6, 8 y 9 tienen un costo conjunto de 4
- El ciclo de la línea 5 tiene una inicialización, una comparación, un incremento y contiene las líneas 6 a 9, por lo que el costo es $1 + ((N-1-i)+1) + (N-1-i) + 4(N-1-i) = 6(N-1-i)+2 \sim (N-1)-i$

Alg. para la suma de secuencia con tiempo cuadrático

- El ciclo de la línea 3 contiene N+1 inicializaciones, N asignaciones, N+1 comparaciones, y el ciclo de la línea 5, esto lleva el costo a

$$\begin{aligned} \sim 3N+2+\sum_{i=0}^{N-1}(N-1-i) &= 3N+2+N(N-1)-\sum_{i=1}^{N}(i-1) \\ &= N^2+2N+2-\frac{N(N+1)}{2}-N \\ &= \frac{3}{2}N^2+\frac{3}{2}N+2 \end{aligned}$$

- La línea 2 contribuye solo un término constante de 1, lo que nos deja con $\sim N^2+N+1$

Como la función obtenida es polinomial, decimos que el tiempo de ejecución del algoritmo es $\Theta(N^2)$.