Análisis de complejidad computacional — Complejidad de tiempo

Miguel Ortiz

Programación competitiva - ICPC

Abril 2025

Introducción

- El análisis de complejidad mide la cantidad de recursos que utiliza un algoritmo
- La eficiencia de algoritmos es *muy* importante en competencias de programación

Complejidad de tiempo

- Para motivos prácticos, estima el tiempo que demora un algoritmo en función al tamaño de la entrada
- Regla general: 10^8 operaciones por segundo

Notación Big O

Notación Big O

Sea $T(n)$ una función; dada otra función $f(n)$, se dice que $T(n)$ es $O(f(n))$ si existen constantes $c > 0$ y $n_0 \geq 0$, tales que para todo $n \geq n_0$ se tenga $T(n) \leq c \cdot f(n)$

Notación Big O

- $O(...) \rightarrow$ "O de ..."
- ... $\rightarrow$ expresión matemática. Ej.: $n, n^2, n + m, n2^n$

Notación Big O

- $O(...) \rightarrow$ "O de ..."
- $... \rightarrow$ expresión matemática. Ej.: $n, n^2, n + m, n2^n$
- Digamos $O(n^2)$. Reemplazamos n por su valor máximo en el problema
- $n \leq 10^5 \rightarrow n^2 = \boxed{10^{10}} \leftarrow$ cantidad aproximada de operaciones

Notación Big O

- $O(...) \rightarrow$ "O de ..."
- $... \rightarrow$ expresión matemática. Ej.: $n, n^2, n + m, n2^n$
- Digamos $O(n^2)$. Reemplazamos n por su valor máximo en el problema
- $n \leq 10^5 \rightarrow n^2 = \boxed{10^{10}} \leftarrow$ cantidad aproximada de operaciones
- ¿Es eficiente?

Notación Big O

- $X \leftarrow$ Cantidad aproximada de operaciones
- $X \leq 10^7 \rightarrow$ Probablemente se suficientemente rápido
- $X \approx 10^8 \rightarrow$ Zona de peligro
- $X \geq 10^9 \rightarrow$ Probablemente sea muy lento

Notación Big O – Operaciones simples

```cpp
int n;
cin >> n;
long long respuesta = (long long)n * (n-1) / 2;
cout << respuesta << '\n';
```

- Cantidad constante de operaciones $\rightarrow O(1)$
- Si $n = 10^5$, el código de arriba sigue siendo $O(1)$

Notación Big O – Ciclos

```
for (int i = 1; i <= n; ++i) {
  // Código O(1)
}
```

- *Código* se realiza n veces $\rightarrow O(n)$

Notación Big O – Ciclos

```
for (int i = 1; i <= n; ++i) {
  for (int j = 1; j <= n; ++j) {
    // Código O(1)
  }
}
```

- *Código* se realiza $n \times n$ veces $\rightarrow O(n^2)$

Notación Big O – Ciclos

```
for (int i = 1; i <= n; ++i) {
  for (int j = 1; j <= n; ++j) {
    // Código O(1)
  }
}
```

- *Código* se realiza $n \times n$ veces $\rightarrow O(n^2)$
- Regla general: si hay k ciclos **anidados**, la complejidad es $O(n^k)$

Notación Big O – Orden de magnitud

```
for (int i = 1; i <= 3*n; ++i) {
  // Código O(1)
}
```

- Realiza $3n$ operaciones

Notación Big O – Orden de magnitud

```
for (int i = 1; i <= 3*n; ++i) {
  // Código O(1)
}
```

- Realiza $3n$ operaciones
- $O(n)$

Notación Big O – Orden de magnitud

```
for (int i = 1; i <= n+5; ++i) {
  // Código O(1)
}
```

- Realiza $n + 5$ operaciones

Notación Big O – Orden de magnitud

```
for (int i = 1; i <= n+5; ++i) {
  // Código O(1)
}
```

- Realiza $n + 5$ operaciones
- $O(n)$

Notación Big O – Orden de magnitud

```
for (int i = 1; i <= n; i += 2) {
  // Código O(1)
}
```

- Realiza $n/2$ operaciones

Notación Big O – Orden de magnitud

```
for (int i = 1; i <= n; i += 2) {
  // Código O(1)
}
```

- Realiza $n/2$ operaciones
- $O(n)$

Notación Big O – Fases

```
for (int i = 1; i <= n; ++i) {
  // Código O(1)
}
for (int i = 1; i <= n; ++i) {
  for (int j = 1; j <= n; ++j) {
    // Código O(1)
  }
}
for (int i = 1; i <= n; ++i) {
  // Código O(1)
}
```

- $O(n + n^2 + n)$

Notación Big O – Fases

```
for (int i = 1; i <= n; ++i) {
  // Código O(1)
}
for (int i = 1; i <= n; ++i) {
  for (int j = 1; j <= n; ++j) {
    // Código O(1)
  }
}
for (int i = 1; i <= n; ++i) {
  // Código O(1)
}
```

- $O(n + n^2 + n)$
- $O(n^2)$

Notación Big O – Más de una variable

```
for (int i = 1; i <= n; ++i) {
  for (int j = 1; j <= m; ++j) {
    // Código O(1)
  }
}
```

- Realiza $n \times m$ operaciones $\rightarrow O(nm)$

Notación Big O – Recomendación

Siempre calculen la complejidad de sus soluciones

Ejercicios

Ejercicios

```
int a = 1, b = 2;
int c = a + b;
```

- Complejidad:

Ejercicios

```
int a = 1, b = 2;
int c = a + b;
```

- Complejidad: $O(1)$

Ejercicios

```
int n;
cin >> n
int res = 0;
for (int i = 0; i < n; ++i) {
  res = res + i;
}
cout << res << endl;
```

- Complejidad:

Ejercicios

```
int n;
cin >> n
int res = 0;
for (int i = 0; i < n; ++i) {
  res = res + i;
}
cout << res << endl;
```

- Complejidad: $O(n)$

Ejercicios

```
int n;
cin >> n
int res = 0;
for (int i = 0; i < n; ++i) {
  for (int j = 0; j < n; ++j) {
    res = res + i*j;
  }
}
cout << res << endl;
```

- Complejidad:

Ejercicios

```
int n;
cin >> n
int res = 0;
for (int i = 0; i < n; ++i) {
  for (int j = 0; j < n; ++j) {
    res = res + i*j;
  }
}
cout << res << endl;
```

- Complejidad: $O(n^2)$

Ejercicios

```
int n;
cin >> n
int res = 0;
for (int i = 0; i < 3*n; ++i) {
  res = res + i%3;
}
cout << res << endl;
```

- Complejidad:

Ejercicios

```
int n;
cin >> n
int res = 0;
for (int i = 0; i < 3*n; ++i) {
  res = res + i%3;
}
cout << res << endl;
```

- Complejidad: $O(n)$

Ejercicios

```
int n;
cin >> n
int res = 0;
for (int i = 0; i < n; ++i) {
  for (int j = 0; j < n; ++j) {
    res = res - i*j;
  }
  for (int j = 0; j < n; ++j) {
    for (int k = 0; k < n; ++k) {
      res = res + i*(j - k);
    }
  }
}
cout << res << endl;
```

- Complejidad:

Ejercicios

```
int n;
cin >> n
int res = 0;
for (int i = 0; i < n; ++i) {
  for (int j = 0; j < n; ++j) {
    res = res - i*j;
  }
  for (int j = 0; j < n; ++j) {
    for (int k = 0; k < n; ++k) {
      res = res + i*(j - k);
    }
  }
}
cout << res << endl;
```

- Complejidad: $O(n \times (n + n^2))$

Ejercicios

```
int n;
cin >> n
int res = 0;
for (int i = 0; i < n; ++i) {
  for (int j = 0; j < n; ++j) {
    res = res - i*j;
  }
  for (int j = 0; j < n; ++j) {
    for (int k = 0; k < n; ++k) {
      res = res + i*(j - k);
    }
  }
}
cout << res << endl;
```

- Complejidad: $O(n \times (n + n^2)) \rightarrow O(n^3)$

Ejercicios

```
int n;
cin >> n
int res = 0;
for (int i = 0; i < n; ++i) {
  for (int j = i; j < n; ++j) {
    res++;
  }
}
cout << res << endl;
```

- Complejidad:

Ejercicios

```
int n;
cin >> n
int res = 0;
for (int i = 0; i < n; ++i) {
  for (int j = i; j < n; ++j) {
    res++;
  }
}
cout << res << endl;
```

- Complejidad: $O(\frac{n(n+1)}{2})$

Ejercicios

```
int n;
cin >> n
int res = 0;
for (int i = 0; i < n; ++i) {
  for (int j = i; j < n; ++j) {
    res++;
  }
}
cout << res << endl;
```

- Complejidad: $O(\frac{n(n+1)}{2}) \rightarrow O(n^2)$

Ejercicios

```
int n;
cin >> n
int res = 0;
for (int i = 0; i < n; ++i) {
  for (int j = 0; j < 8; ++j) {
    res++;
  }
}
cout << res << endl;
```

- Complejidad:

Ejercicios

```
int n;
cin >> n
int res = 0;
for (int i = 0; i < n; ++i) {
  for (int j = 0; j < 8; ++j) {
    res++;
  }
}
cout << res << endl;
```

- Complejidad: $O(n)$

Ejercicios

```
vector<int> v;
v.push_back(2023);
```

- Complejidad:

Ejercicios

```
vector<int> v;
v.push_back(2023);
```

- Complejidad: $O(1)$

Ejercicios

```
set<int> s;
s.insert(2023);
```

- Complejidad:

Ejercicios

```
set<int> s;
s.insert(2023);
```

- Complejidad: $O(\log_2 n) \rightarrow O(\log n)$

Ejercicios

```
int n;
cin >> n;
set<int> s;
for (int i = 0; i < n; ++i) {
  s.insert((2*i + i/3) % n);
}
cout << s.size() << endl;
```

- Complejidad:

Ejercicios

```
int n;
cin >> n;
set<int> s;
for (int i = 0; i < n; ++i) {
  s.insert((2*i + i/3) % n);
}
cout << s.size() << endl;
```

- Complejidad: $O(n \log n)$