# Estructuras de Datos

Andrea Rueda

Pontificia Universidad Javeriana
Departamento de Ingeniería de Sistemas

# Ejercicio de "calentamiento"

# Fundamentos de complejidad

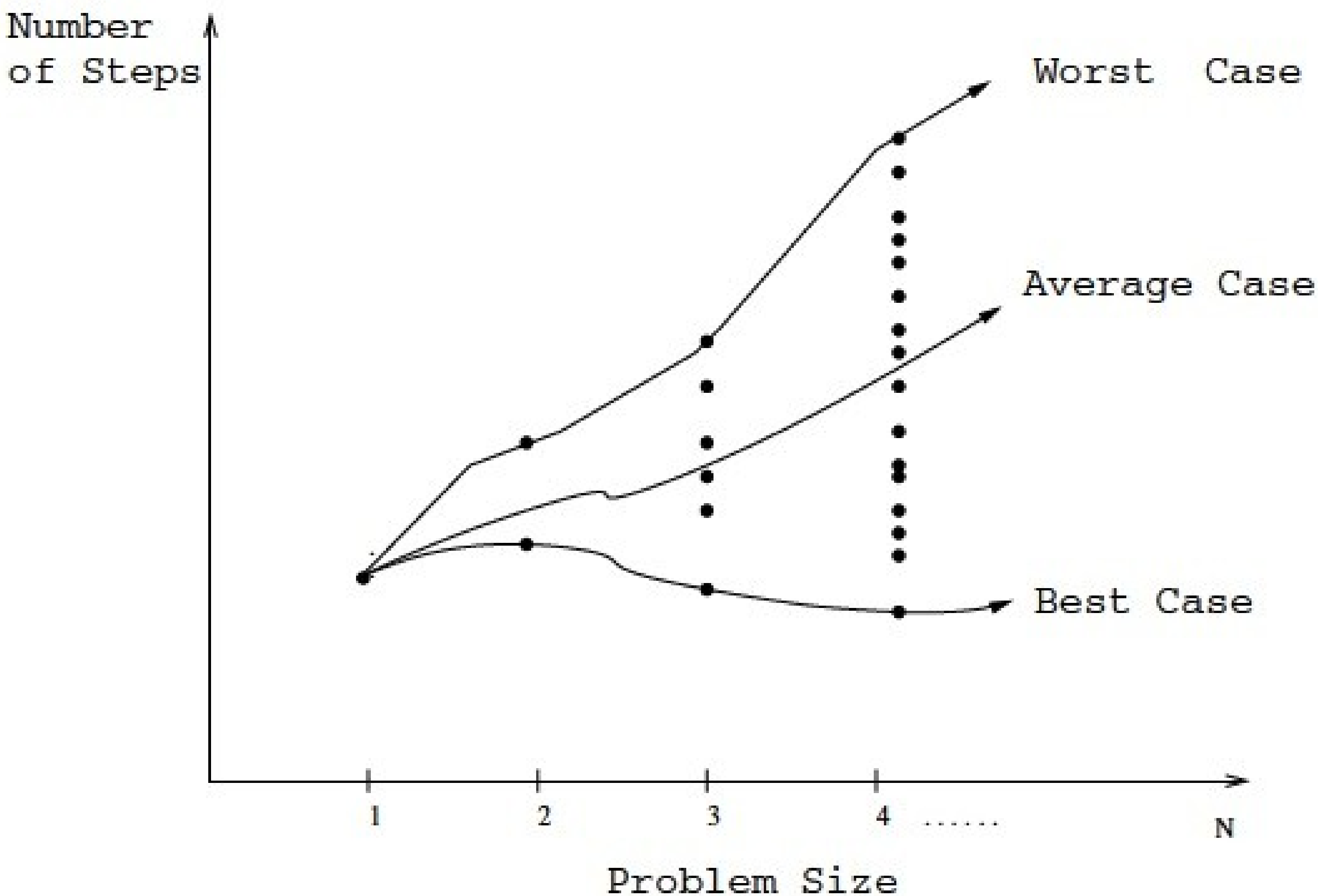
# ¿Es eficiente un algoritmo?

- Análisis de los recursos que el algoritmo requerirá.
- Complejidad en espacio:
  - Memoria.
  - Ancho de banda de comunicación.
  - Almacenamiento en disco.
- Complejidad en tiempo:
  - **Tiempo computacional.**

# ¿Cómo medir la complejidad?

- Análisis empírico.
- Análisis teórico (asintótico).

# Análisis teórico




vaxxxa.github.io/talks/introduction.to.algorithms-computational.complexity/#/17

# Análisis teórico

- Ejemplo: búsqueda lineal.

- Mejor caso: límite inferior.

  $T(n) = 1 \rightarrow \mathbf{\Omega()}$

- Peor caso: límite superior.

  $T(n) = n \rightarrow \mathbf{O()}$

- Caso promedio: límite ajustado.

  $T(n) = 1/2\ n \rightarrow \mathbf{\Theta()}$

# Ejemplo

- Análisis de complejidad de algoritmos:

```
double exp(double a, int n) {
  double res = 1;     // 1
  int i = 0;          // 1
  while(i < n) {      // 1 * (n+1)
    res = res * a;    // 2 * n
    i = i + 1;        // 2 * n
  }
  return(res);        // 1
}
```

$T_{exp}(n) = 1+1+(n+1)+n(2+2)+1 = 5n+4 \rightarrow O(n)$

# Ejemplo

- Análisis de complejidad de algoritmos:

```
double exp(double a, int n) {
  if(n == 0)                          // 1
    return(1);                        // 1
  else
    return(exp(a, n-1) * a);  // 1+1+T(n-1)
}
```

$T_{exp}(n) = 3 + T_{exp}(n\text{-}1)$

$T_{exp}(n) = 3 + 3 + T_{exp}(n\text{-}2)$

$T_{exp}(n) = 3 + 3 + \ldots + 2$

$T_{exp}(n) = 3n+2 \;\; \rightarrow O(n)$

# Ejercicio

- ¿T(n) para búsqueda lineal?

```
int busqueda_lineal
         (int elemento, int lista[], int n) {
   int res = -1;
   for (int i=0; i<n; i++)
      if (lista[i]==elemento)
         res = i;
   return(res);
}
```

# Ejercicio

- ¿T(n) para búsqueda lineal?

```
int busqueda_lineal
        (int elemento, int lista[], int n) {
   int res = -1;                     // 1
   for (int i=0; i<n; i++)           // 1+(n+1)+n
      if (lista[i]==elemento)        // n*(1+1)
         res = i;                    // 1
   return(res);                      // 1
}
```

$T(n) = 1+1+(n+1)+n+2n+1+1 = 4n+5$

# Ejercicio

- ¿T(n) para búsqueda lineal?

```
int busqueda_lineal
        (int elemento, int lista[], int n) {
   int res = -1;                         // 1
   for (int i=0; i<n; i++)               // 1+(n+1)+n
      if (lista[i]==elemento)            // n*(1+1)
         res = i;                        // 1
   return(res);                          // 1
}
```

$T(n) = 1+1+(n+1)+n+2n+1+1 = 4n+5 \rightarrow \mathbf{O(n)}$

# Ejercicio

- ¿T(n) para suma de matrices cuadradas?

```
void suma_matrices
         (int** A, int** B, int** C, int n) {
   for (int i=0; i<n; i++)
      for (int j=0; j<n; j++)
         C[i][j] = A[i][j] + B[i][j];
}
```

# Ejercicio

- ¿T(n) para suma de matrices cuadradas?

```
void suma_matrices
         (int** A, int** B, int** C, int n) {
   for (int i=0; i<n; i++)    // 1+(n+1)+n
      for (int j=0; j<n; j++)// n*(1+(n+1)+n)
         C[i][j] = A[i][j] + B[i][j];// n*n*8
}
```

$$T(n) = 1+(n+1)+n+n(1+(n+1)+n)+n*n*8$$
$$= 2n+2+2n^2+2n+8n^2 = 10n^2+4n+2$$

# Ejercicio

- ¿T(n) para suma de matrices cuadradas?

```
void suma_matrices
         (int** A, int** B, int** C, int n) {
   for (int i=0; i<n; i++)    // 1+(n+1)+n
      for (int j=0; j<n; j++)// n*(1+(n+1)+n)
         C[i][j] = A[i][j] + B[i][j];// n*n*8
}
```

$$T(n) = 1+(n+1)+n+n(1+(n+1)+n)+n*n*8$$
$$= 2n+2+2n^2+2n+8n^2 = 10n^2+4n+2 \rightarrow O(n^2)$$

# Ejercicio

- ¿T(n) para multiplicación de matrices cuadradas?

```
void mult_matrices
         (int** A, int** B, int** C, int n) {
   for (int i=0; i<n; i++)
      for (int j=0; j<n; j++) {
         C[i][j] = 0;
         for (int k=0; k<n; k++)
            C[i][j] += A[i][k] * B[k][j];
      }
}
```

# Ejercicio

- ¿T(n) para multiplicación de matrices cuadradas?

```
void mult_matrices
        (int** A, int** B, int** C, int n) {
   for (int i=0; i<n; i++)           // 1+(n+1)+n
      for (int j=0; j<n; j++) {      // n*(1+(n+1)+n)
         C[i][j] = 0;                // n*n*3
         for (int k=0; k<n; k++)     // n*n*(1+(n+1)+n)
            C[i][j] += A[i][k] * B[k][j]; // n*n*n*9
      }
}
```

$$T(n) = 2n+2+n(2n+2)+3n^2+n^2(2n+2)+9n^3$$
$$= 11n^3+7n^2+4n+2$$

# Ejercicio

- ¿T(n) para multiplicación de matrices cuadradas?

```
void mult_matrices
          (int** A, int** B, int** C, int n) {
   for (int i=0; i<n; i++)            // 1+(n+1)+n
      for (int j=0; j<n; j++) {       // n*(1+(n+1)+n)
         C[i][j] = 0;                 // n*n*3
         for (int k=0; k<n; k++)      // n*n*(1+(n+1)+n)
            C[i][j] += A[i][k] * B[k][j]; // n*n*n*9
      }
}
```

$$T(n) = 2n+2+n(2n+2)+3n^2+n^2(2n+2)+9n^3$$
$$= 11n^3+7n^2+4n+2 \rightarrow \mathbf{O(n^3)}$$

# Ejercicio

- ¿T(n) para Fibonacci recursivo?

```
unsigned int nFib( unsigned int n ) {
   if( n <= 1 )
      return( n );
   else
      return( nFib( n - 1 ) + nFib( n - 2 ) );
}
```

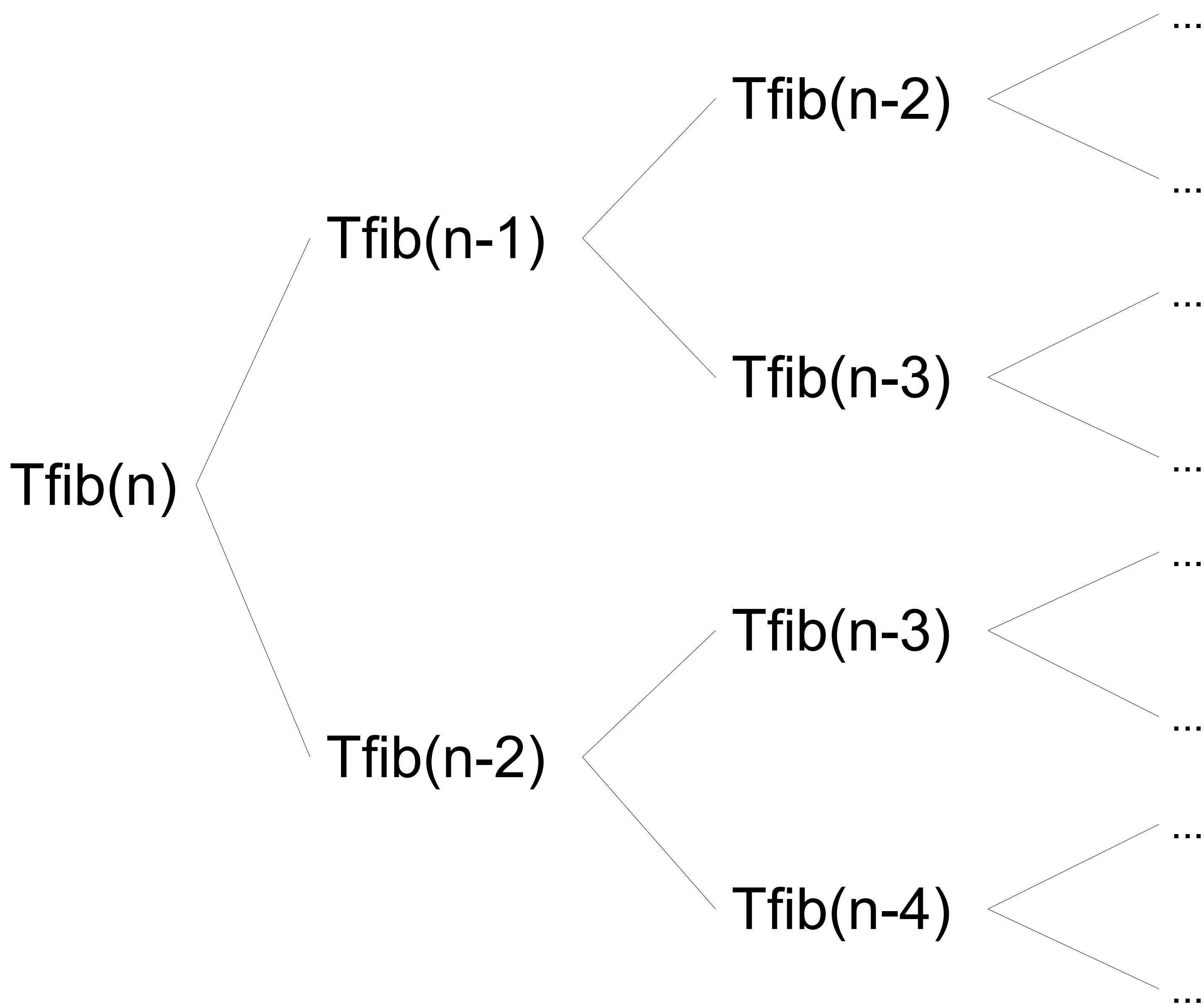
# Ejercicio

- ¿T(n) para Fibonacci recursivo?

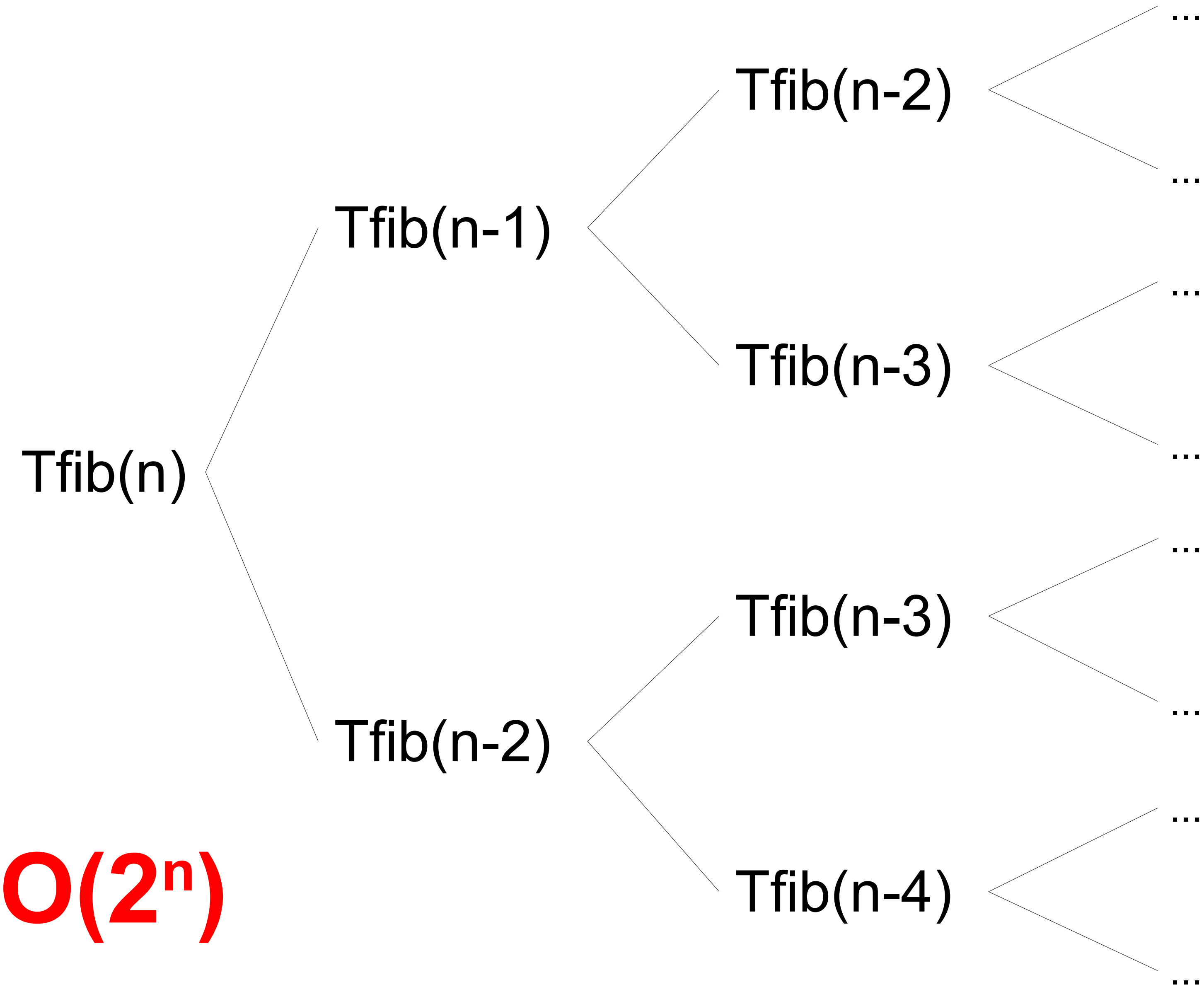
```
unsigned int nFib( unsigned int n ) {
   if( n <= 1 )
      return( n );
   else
      return( nFib( n - 1 ) + nFib( n - 2 ) );
}
```

- Tfib(n) = 1 + 2 + Tfib(n-1) + Tfib(n-2)
- Tfib(n-1) = 1 + 2 + Tfib(n-2) + Tfib(n-3)
- Tfib(n-2) = 1 + 2 + Tfib(n-3) + Tfib(n-4)

# Ejercicio



# Ejercicio




$\rightarrow O(2^n)$

# Ejercicio

- Comparación de algoritmos en términos de complejidad:

  Algoritmos de búsqueda:

  - Búsqueda lineal. → **O(n)**
  - Búsqueda binaria.

# Ejercicio

- Comparación de algoritmos: Búsqueda binaria.

```
bool busqueda_binaria
        (int elemento, int lista[], int n) {
   bool encontrado = false;
   int primero = 0;
   int ultimo = n – 1;
   while (primero <= ultimo && !encontrado) {
      int p = (primero + ultimo) >> 1;
      if (elemento < lista[p])
         ultimo = p – 1;
      else if(elemento > lista[p])
         primero = p + 1;
      else
         encontrado = true;
   }
   return(encontrado);
}
```

# Ejercicio

- Comparación de algoritmos: Búsqueda binaria.
  - $T_{bb}(n) = 6 + k$ (k: número de iteraciones necesarias)
  - $n_0 = n$; $n_1 = n/2$; $n_2 = n/4$; $n_3 = n/8$; …; $n_k = 1$.
  - $n/2^k < 2$ (tamaño del sub-arreglo después de k iteraciones)
  - $n/2^k < 2 : n < 2^{k+1}$ : fórmula de acotamiento.

# Ejercicio

- Comparación de algoritmos: Búsqueda binaria.
  - $T_{bb}(n) = 6 + k$ (k: número de iteraciones necesarias)
  - $n_0 = n$; $n_1 = n/2$; $n_2 = n/4$; $n_3 = n/8$; …; $n_k = 1$.
  - $n/2^k < 2$ (tamaño del sub-arreglo después de k iteraciones)
  - $n/2^k < 2 : n < 2^{k+1}$ : fórmula de acotamiento.
  - $T_{bb}(n)$ es **$O(\log_2(n))$**

# Ejercicio

- Comparación de algoritmos: Búsqueda binaria
  - $T_{bb}(n) = 6 + k$ (k: número de iteraciones necesarias)
  - $n_0 = n$; $n_1 = n/2$; $n_2 = n/4$; $n_3 = n/8$; …; $n_k = 1$
  - $n/2^k < 2$ (tamaño del sub-arreglo después de k iteraciones)
  - $n/2^k < 2 : n < 2^{k+1}$ : fórmula de acotamiento
  - $T_{bb}(n)$ es **$O(\log_2(n))$**

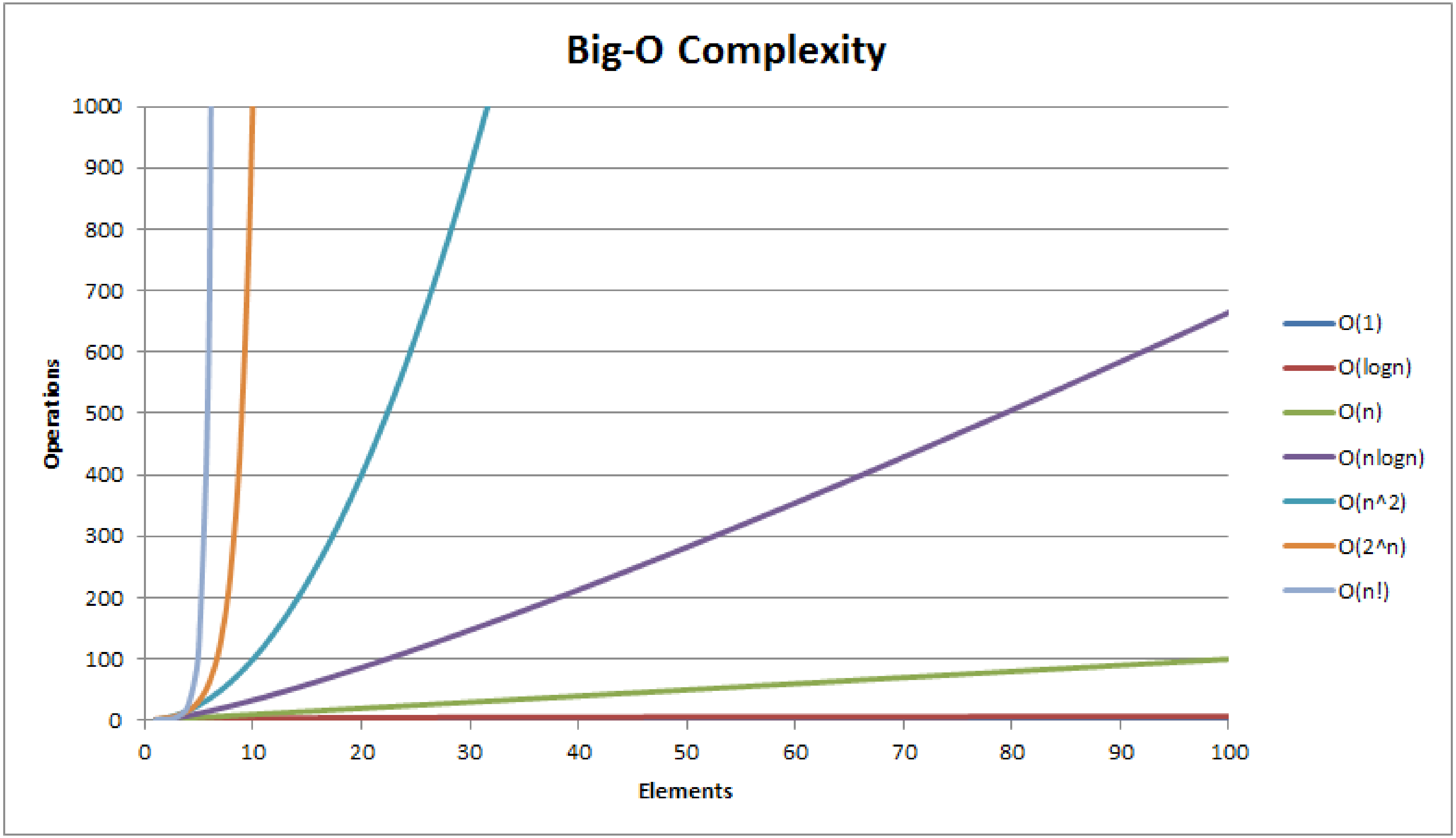
¿Cuál búsqueda es mejor?

# Análisis teórico

| Notation | Name |
| --- | --- |
| $O(1)$ | Constant |
| $O(\log(n))$ | Logarithmic |
| $O(\log(\log(n))$ | Double logarithmic (iterative logarithmic) |
| $o(n)$ | Sublinear |
| $O(n)$ | Linear |
| $O(n\log(n))$ | Loglinear, Linearithmic, Quasilinear or Supralinear |
| $O(n^2)$ | Quadratic |
| $O(n^3)$ | Cubic |
| $O(n^c)$ | Polynomial (different class for each $c > 1$) |
| $O(c^n)$ | Exponential (different class for each $c > 1$) |
| $O(n!)$ | Factorial |
| $O(n^n)$ | - (Yuck!) |

T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein.
Introduction to Algorithms, 3rd edition. MIT Press, 2009.

# Análisis teórico




vaxxxa.github.io/talks/introduction.to.algorithms-computational.complexity/index.html#/28

# Tarea

- Antes del próximo miércoles 3 de febrero
- Enunciado en Uvirtual
  → Quices, tareas y ejercicios adicionales
  → Tarea Compilación y Depuración en C++

# Ejercicios

## Función #1

```
void f1 ( long vec1[], int n,
          long mat1[][], int x, int y ) {
  int l = 0;
  for ( int i = 0; i < x; i++ )
    for ( int j = 0; j < y; j++ )
      if ( mat1[i][j] == 12345 ) {
        vec1[l] = 12345;
        l = (l + 1) % n;
      }
}
```

# Ejercicios

## Función #1

```
void f1 ( long vec1[], int n,
          long mat1[][], int x, int y ) {
  int l = 0;                              // 1
  for ( int i = 0; i < x; i++ )           // 1 + x+1 + x
    for ( int j = 0; j < y; j++ )         // x*(1 + y+1 + y)
      if ( mat1[i][j] == 12345 ) {        // x*(y*(3))
        vec1[l] = 12345;                  // x*y*2
        l = (l + 1) % n;                  // x*y*3
      }
}
```

T(n) = 1+1+x+1+x+x+xy+x+xy+3xy+2xy+3xy

# Ejercicios

## Función #1

```
void f1 ( long vec1[], int n,
          long mat1[][], int x, int y ) {
  int l = 0;                                 // 1
  for ( int i = 0; i < x; i++ )              // 1 + x+1 + x
    for ( int j = 0; j < y; j++ )            // x*(1 + y+1 + y)
      if ( mat1[i][j] == 12345 ) {           // x*(y*(3))
        vec1[l] = 12345;                     // x*y*2
        l = (l + 1) % n;                     // x*y*3
      }
}
```

$$T(x,y) = 3+4x+10xy \approx T(x) = 3+4x+10x^2 \quad \rightarrow O(n^2)$$

# Ejercicios

## Función #2

```
void f2 ( float mat1[][], int m, int n ) {
  int mi = 0;
  int mj = 0;
  for ( int i = 0; i < m; i++ )
    for ( int j = 0; j < n; j++ )
      if ( mat1[i][j] > mat1[mi][mj] ) {
        mi = i;
        mj = j;
      }
  mat1[mi][mj] = 0,0;
}
```

# Ejercicios

## Función #2

```
void f2 ( float mat1[][], int m, int n ) {
  int mi = 0;                                   // 1
  int mj = 0;                                   // 1
  for ( int i = 0; i < m; i++ )                 // 1+m+1+m
    for ( int j = 0; j < n; j++ )               // m*(1+n+1+n)
      if ( mat1[i][j] > mat1[mi][mj] ) {        // m*n*5
        mi = i;                                 // m*n*1
        mj = j;                                 // m*n*1
      }
  mat1[mi][mj] = 0,0;                           // 3
}
```

$$T(m,n) = 7+4m+9mn \approx T(m) = 7+4m+9m^2 \rightarrow O(n^2)$$

# Ejercicios

## Función #3

```
void f3 ( int mat[][], int m ) {
  int vec[m];
  for ( int i = 0; i < m; i++ )
    vec[i] = 0;
  for ( int i = 0; i < 3; i++ )
    for ( int j = 0; j < m; j++ )
      vec[j] = vec[j] + mat[j][i];
}
```

# Ejercicios

## Función #3

```
void f3 ( int mat[][], int m ) {
  int vec[m];                              // 1
  for ( int i = 0; i < m; i++ )            // 1 + m+1 + m
    vec[i] = 0;                            // m * 2
  for ( int i = 0; i < 3; i++ )            // 1 + 4 + 3
    for ( int j = 0; j < m; j++ )          // 3 * (1 + m+1 + m)
      vec[j] = vec[j] + mat[j][i];         // 3 * m * 6
}
```

$$T(m) = 1+1+m+1+m+2m+8+3+3m+3+3m+18m$$
$$= 17+28m \rightarrow O(n)$$

# Ejercicios

## Función #4

```
void f4 ( float mat[][], int m, int n ) {
  float vec[m];
  for ( int i = 0; i < m; i++ ) {
    for ( int j = 0; j < n/2; j++ )
      // no se hace nada
    vec[i] = mat[i][j];
  }
}
```

# Ejercicios

## Función #4

```
void f4 ( float mat[][], int m, int n ) {
  float vec[m];                            // 1
  for ( int i = 0; i < m; i++ ) {   // 1 + m+1 + m
    for ( int j = 0; j < n/2; j++ ) // m*(1+n/2+1+n/2)
      // no se hace nada
    vec[i] = mat[i][j];                   // m * 4
  }
}
```

$$T(m,n) = 1+1+m+1+m+m+mn/2+m+mn/2+4m$$

$$= 3+8m+mn \approx T(m) = 3+8m+m^2 \rightarrow O(n^2)$$

# Ejercicios

## Función #5

```
void f5 ( char mat[][], int m, int n ) {
  int vec[m];
  for ( int i = 0; i < m; i++ )
    vec[i] = 0;
  for ( int i = 0; i < m; i++ )
    for ( int j = 0; j < n; j++ )
      if ( mat[i][j] == '?' ) {
        vec[i]++;
      }
}
```

# Ejercicios

## Función #5

```
void f5 ( char mat[][], int m, int n ) {
  int vec[m];                           // 1
  for ( int i = 0; i < m; i++ )         // 1 + m+1 + m
    vec[i] = 0;                         // m * 2
  for ( int i = 0; i < m; i++ )         // 1 + m+1 + m
    for ( int j = 0; j < n; j++ )       // m * (1 + n+1 + n)
      if ( mat[i][j] == '?' ) {         // m*n*3
        vec[i]++;                       // m*n*3
      }
}
```

$$T(m,n) = 5+8m+8mn \approx T(m) = 5+8m+8m^2 \rightarrow O(n^2)$$

# Referencias


- T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein. Introduction to Algorithms, 3rd edition. MIT Press, 2009.
- vaxxxa.github.io/talks/introduction.to.algorithms-computational.complexity/