

Análisis de eficiencia de algoritmos

Algorítmica. Práctica 1

Jose Alberto Hoces Castro Javier Gómez López Moya Martín Castaño

Contenidos

- 1. Introducción
- 2. Análisis de los algoritmos propuestos
- 3. Casos especiales
- 4. Conclusiones

Introducción

Análisis de eficiencia de algoritmos

- Análisis de la eficiencia teórica: estudio de la complejidad teórica de algoritmos.
- Análisis de la eficiencia empírica: ejecución y medición de tiempos de ejecución de los algoritmos estudiados.
- Análisis de la eficiencia híbrida: obtención de las constantes ocultas.

Consiste en analizar sobre el papel el peor tiempo de ejecución posible en un algoritmo para decidir en qué clase de funciones en notación O se encuentra

Cálculo de la eficiencia empírica

Ejecución de los algortimos en distintos agentes tecnológicos, calculando su tiempo de ejecución con la librería <chrono>.

Cálculo de la eficiencia híbrida

Obtención de las constantes ocultas a través de gnuplot.

propuestos _____

Análisis de los algoritmos

Algoritmos trabajados

Se ha realizado un análisis de los siguientes algoritmos:

- 1. Algoritmo de Inserción
- 2. Algoritmo de Selección
- 3. Algoritmo de Quicksort
- 4. Algoritmo de Heapsort
- 5. Algoritmo de Floyd
- 6. Algoritmos de las torres de Hanoi

Inserción

El código del algoritmo de Inserción es el siguiente:

```
static void insercion_lims(int T[], int inicial, int final)
2 {
    int i, j;
3
    int aux;
    for (i = inicial + 1; i < final; i++) { // 0(n)
      i = i; // 0(1)
6
      while ((T[j] < T[j-1]) \&\& (j > 0)) \{ // 0(n) \}
        aux = T[j]; // 0(1)
8
        T[j] = T[j-1]; // O(1)
9
        T[i-1] = aux; // 0(1)
10
        j--; // 0(1)
11
     };
12
    };
13
14 }
```

En los comentarios del código observamos el análisis de la función.

Son dos bucles, uno for y otro while, los cuales están anidados y por ser cada uno O(n):

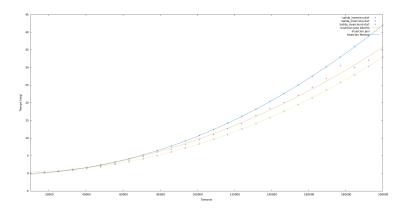
$$T(n) \in O(n^2)$$

Inserción. Eficiencia empírica

| Intel Core i7-6700 3.40 GHz | | i5-1095G1 1.00 GHz | | Ordenador Moya | |
|-----------------------------|------------|--------------------|------------|----------------|------------|
| Elementos (n) | Tiempo (s) | Elementos (n) | Tiempo (s) | Elementos (n) | Tiempo (s) |
| 17600 | 0.251124 | 17600 | 0.35799 | 17600 | 0.321303 |
| 25200 | 0.560574 | 25200 | 0.609488 | 25200 | 0.661228 |
| 32800 | 0.905768 | 32800 | 0.998112 | 32800 | 1.12508 |
| 40400 | 1.33038 | 40400 | 1.52076 | 40400 | 1.705 |
| 48000 | 1.87672 | 48000 | 2.12746 | 48000 | 2.41886 |
| 55600 | 2.51991 | 55600 | 2.89747 | 55600 | 3.23931 |
| 63200 | 3.29735 | 63200 | 3.74891 | 63200 | 4.18747 |
| 70800 | 4.08019 | 70800 | 4.70754 | 70800 | 5.2435 |
| 78400 | 4.98866 | 78400 | 6.08267 | 78400 | 6.4519 |
| 86000 | 6.08448 | 86000 | 6.88299 | 86000 | 7.74454 |
| 93600 | 7.17045 | 93600 | 8.15529 | 93600 | 9.18276 |
| 101200 | 8.36352 | 101200 | 9.6372 | 101200 | 10.7333 |
| 108800 | 9.71516 | 108800 | 10.9647 | 108800 | 12.4379 |
| 116400 | 11.0357 | 116400 | 12.6405 | 116400 | 14.2603 |
| 124000 | 12.5611 | 124000 | 14.1936 | 124000 | 16.1453 |
| 131600 | 14.1345 | 131600 | 16.3756 | 131600 | 18.1743 |
| 139200 | 15.7984 | 139200 | 18.3599 | 139200 | 20.4184 |
| 146800 | 17.6155 | 146800 | 20.0244 | 146800 | 22.6048 |
| 154400 | 19.5025 | 154400 | 22.1302 | 154400 | 25.0412 |
| 162000 | 21.4432 | 162000 | 24.3748 | 162000 | 27.5086 |
| 169600 | 23.5908 | 169600 | 26.8462 | 169600 | 30.1526 |
| 177200 | 25.7055 | 177200 | 30.5882 | 177200 | 32.9759 |
| 184800 | 27.9704 | 184800 | 30.0598 | 184800 | 35.8989 |
| 192400 | 30.2777 | 192400 | 32.0387 | 192400 | 38.8935 |
| 200000 | 32.8911 | 200000 | 34.7391 | 200000 | 41.9351 |

Tabla 1: Experiencia empírica de algoritmo de Inserción sin optimizar

Inserción. Eficiencia híbrida



Inserción. Eficiencia híbrida

- i7-6700 3.40Ghz $\rightarrow T_1(n) = 8.49924 \cdot 10^{-10}x^2 - 8.57879 \cdot 10^{-6}x + 0.546581$
- Ordenador José Alberto $\rightarrow T_2(n) = 7.96341 \cdot 10^{-10} x^2 + 2.23563 \cdot 10^{-5} x - 0.592279$
- Ordenador Manuel $\rightarrow T_3(n) = 1.04394 \cdot 10^{-9} x^2 + 1.58593 \cdot 10^{-6} x - 0.0969414$

Varianza residual:

- $T_1(n) \longrightarrow Var.res = 0.00162352$
- $T_2(n) \longrightarrow Var.res = 0.0050675$
- $T_3(n) \longrightarrow Var.res = 0.00161535$

Selección

El código del algoritmo de Selección es el siguiente:

```
static void seleccion_lims(int T[], int inicial, int final)
2 {
    int i, j, indice_menor;
3
    int menor, aux;
    for (i = inicial; i < final - 1; i++) { // 0(n)
      indice_menor = i; // 0(1)
6
      menor = T[i]: // O(1)
      for (j = i; j < final; j++) // O(n)
8
        if (T[j] < menor) {
9
      indice_menor = j; // 0(1)
10
      menor = T[j]; // O(1)
11
      aux = T[i]; // 0(1)
      T[i] = T[indice\_menor]; // 0(1)
14
      T[indice\_menor] = aux; // 0(1)
15
    };
16
17
```

En los comentarios del código observamos el análisis de la función. Son dos bucles for, los cuales están anidados y por ser cada uno

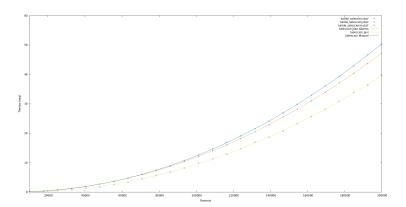
O(n):

$$T(n) \in O(n^2)$$

Selección. Eficiencia empírica

| Intel Core i7-6700 3.40 GHz i5-1095G1 1.00 GHz | | | Ordenador Moya | | | |
|--|------------|---------------|----------------|---|---------------|------------|
| Elementos (n) | Tiempo (s) | Elementos (n) | Tiempo (s) | | Elementos (n) | Tiempo (s) |
| 17600 | 0.260828 | 17600 | 0.391322 | | 17600 | 0.357489 |
| 25200 | 0.504322 | 25200 | 0.76325 | | 25200 | 0.730079 |
| 32800 | 0.835328 | 32800 | 1.27203 | | 32800 | 1.25708 |
| 40400 | 1.25944 | 40400 | 1.92909 | | 40400 | 1.90694 |
| 48000 | 1.7931 | 48000 | 2.71989 | | 48000 | 2.69298 |
| 55600 | 2.54346 | 55600 | 3.65109 | Ш | 55600 | 3.61969 |
| 63200 | 3.42159 | 63200 | 4.71913 | Ш | 63200 | 4.72056 |
| 70800 | 4.46734 | 70800 | 5.91709 | | 70800 | 6.01156 |
| 78400 | 5.61827 | 78400 | 7.25541 | | 78400 | 7.41941 |
| 86000 | 6.80333 | 86000 | 8777 | | 86000 | 8.98684 |
| 93600 | 8.16667 | 93600 | 10.3443 | | 93600 | 10.7273 |
| 101200 | 9.86304 | 101200 | 12.0904 | | 101200 | 12.5778 |
| 108800 | 11.3351 | 108800 | 14.0135 | Ш | 108800 | 14.6332 |
| 116400 | 12.9138 | 116400 | 16.0454 | Ш | 116400 | 16.8798 |
| 124000 | 14.7895 | 124000 | 18.19 | Ш | 124000 | 19.0523 |
| 131600 | 16.9792 | 131600 | 20.5113 | Ш | 131600 | 21.5316 |
| 139200 | 18.6877 | 139200 | 22.8553 | Ш | 139200 | 24.0439 |
| 146800 | 20629 | 146800 | 25.4735 | Ш | 146800 | 26.9219 |
| 154400 | 23.2312 | 154400 | 28141 | Ш | 154400 | 29.7736 |
| 162000 | 25691 | 162000 | 31.0438 | Ш | 162000 | 32.9393 |
| 169600 | 28.1704 | 169600 | 33.9582 | Ш | 169600 | 36.1122 |
| 177200 | 30.78 | 177200 | 37.0641 | | 177200 | 39.2833 |
| 184800 | 33.7999 | 184800 | 40.3583 | | 184800 | 42.7955 |
| 192400 | 36.3688 | 192400 | 43.7206 | | 192400 | 46.6683 |
| 200000 | 39.5352 | 200000 | 47.2209 | Ш | 200000 | 50.4019 |

Tabla 2: Experiencia empírica de algoritmo de Selección sin optimizar



Selección. Eficiencia híbrida

- i7-6700 3.40Ghz $\rightarrow T_1(n) = 1.0371 \cdot 10^{-9} x^2 + -9.86278 \cdot 10^{-6} x + 0.0216418.$
- Ordenador José Alberto $\rightarrow T_2(n) = 1.17905 \cdot 10^{-9} x^2 + 3.97249 \cdot 10^{-7} x - 0.00421685.$
- Ordenador Manuel $\rightarrow T_3(n) = 1.29484 \cdot 10^{-9} x^2 - 7.43377 \cdot 10^{-6} x + 0.0733569.$

Varianza residual:

- $T_1(n) \longrightarrow Var.res = 0.0164518$
- $T_2(n) \longrightarrow Var.res = 0.000537586$
- $T_3(n) \longrightarrow Var.res = 0.00387134$

Floyd

El código del algoritmo de Floyd es el siguiente:

```
void Floyd(int **M, int dim)

for (int k = 0; k < dim; k++) //O(n)

for (int i = 0; i < dim;i++) //O(n)

for (int j = 0; j < dim;j++) //O(n)

{
   int sum = M[i][k] + M[k][j];
   M[i][j] = (M[i][j] > sum) ? sum : M[i][j]; //O(1)

//Total O(n^3)
```

Floyd. Eficiencia teórica

En los comentarios del código observamos el análisis de la función. Son tres bucles for anidados, cada uno O(n) y por tanto,

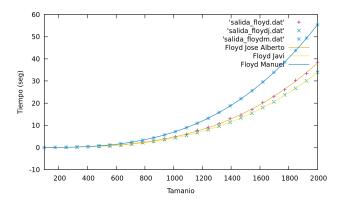
$$T(n) \in O(n^3)$$

Floyd. Eficiencia empírica

| Intel Core i7-67 | Intel Core i7-6700 3.40 GHz i5-1095G1 1.00 GHz | | Ordenado | r Moya | |
|------------------|--|---------------|------------|---------------|------------|
| Elementos (n) | Tiempo (s) | Elementos (n) | Tiempo (s) | Elementos (n) | Tiempo (s) |
| 176 | 0.0244106 | 176 | 0.0274773 | 176 | 0.038495 |
| 252 | 0.0721776 | 252 | 0.0995705 | 252 | 0.111472 |
| 328 | 0.155828 | 328 | 0.20657 | 328 | 0.244523 |
| 404 | 0.288165 | 404 | 0.307902 | 404 | 0.45528 |
| 480 | 0.465947 | 480 | 0.51806 | 480 | 0.761621 |
| 556 | 0.724968 | 556 | 0.799187 | 556 | 1.17395 |
| 632 | 1.09236 | 632 | 1.16729 | 632 | 1.73408 |
| 708 | 1.54374 | 708 | 1.65895 | 708 | 2.4355 |
| 784 | 2.13392 | 784 | 2.42549 | 784 | 3.29426 |
| 860 | 2.67022 | 860 | 3.00331 | 860 | 4.35444 |
| 936 | 3.52897 | 936 | 3.84788 | 936 | 5.64407 |
| 1012 | 4.4074 | 1012 | 4.84029 | 1012 | 7.16827 |
| 1088 | 5.42559 | 1088 | 5.97643 | 1088 | 8.91362 |
| 1164 | 6.6698 | 1164 | 7.78043 | 1164 | 10.9311 |
| 1240 | 8.06967 | 1240 | 9.08228 | 1240 | 13.2386 |
| 1316 | 9.55022 | 1316 | 10.7251 | 1316 | 15.8513 |
| 1392 | 11.4197 | 1392 | 12.9933 | 1392 | 18.7744 |
| 1468 | 13.3942 | 1468 | 14.6689 | 1468 | 21.9844 |
| 1544 | 15.5 | 1544 | 17.2185 | 1544 | 25.5768 |
| 1620 | 18.0399 | 1620 | 20.2626 | 1620 | 29.5543 |
| 1696 | 20.5893 | 1696 | 22.9733 | 1696 | 33.8275 |
| 1772 | 23.6714 | 1772 | 26.0557 | 1772 | 38.5849 |
| 1848 | 26.7337 | 1848 | 30.2843 | 1848 | 43.8038 |
| 1924 | 30.1601 | 1924 | 33.4252 | 1924 | 49.4368 |
| 2000 | 33.9673 | 2000 | 38.5217 | 2000 | 55.3965 |

Tabla 3: Experiencia empírica de algoritmo de Floyd sin optimizar

Floyd. Eficiencia híbrida



Floyd. Eficiencia híbrida

- i7-6700 3.4GHz \rightarrow $T_1(n)$ = 4.38237 · 10⁻⁹ x^3 4.33753 · 10⁻⁷ x^2 + 0.000337001x 0.0504332
- i5-1095G1 1.00 GHz $\rightarrow T_2(n) = 5.12922 \cdot 10^{-9} x^3 - 1.11315 \cdot 10^{-6} x^2 + 0.00083571x - 0.134397$
- Ordenador Moya $\rightarrow T_3(n) =$ 6.77297 · 10⁻⁹ x^3 + 5.13099 · 10⁻⁷ x^2 0.000427834x + 0.0714028

Varianza residual:

- $T_1(n) \longrightarrow Var.res = 0.00204522$
- $T_2(n) \longrightarrow Var.res = 0.044778$
- $T_3(n) \longrightarrow Var.res = 0.000855184$

Hanoi

El código del algoritmo de las torres de Hanoi es el siguiente:

```
void hanoi (int M, int i, int j)

if (M > 0)

hanoi(M-1, i, 6-i-j);
hanoi (M-1, 6-i-j, j);

}
```

Hanoi. Eficiencia teórica

Estamos ante un algoritmo recursivo, cuya ecuación de recurrencia es:

$$T(n) = 2T(n-1) + 1$$

$$(x-2)(x-1)=0$$

$$T(n) = c_1 \cdot 2^n + c_2$$

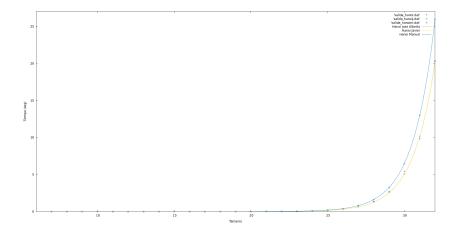
Por tanto:

$$T(n) \in O(2^n)$$

| Elementos (n) Tiempo (s) Elementos (n) Tiempo (s) 8 0.00000136207 8 0.0000037376 9 0.00000267907 9 0.0000037376 10 0.0000028653 10 0.000001256 11 0.000012702 11 0.0000283526 12 0.0000284959 12 0.000046082 14 0.0000904406 14 0.0001022887 15 0.000198225 15 0.0002333459 16 0.000439214 16 0.0002333459 17 0.0008158 17 0.000273949 18 0.0014513 18 0.00142487 19 0.00253865 19 0.0027949 20 0.00499491 20 0.0053405 21 0.010156 21 0.01673 22 0.0209075 22 0.0238254 24 0.0878626 24 0.112827 25 0.171153 25 0.207041 26 0.339115 | Intel Core i7-6700 3.40 GHz | | i5-1095G | 1 1.00 GHz |
|--|-----------------------------|---------------|---------------|---------------|
| 9 0.0000267907 10 0.0000528653 10 0.0000737613 11 0.000012702 11 0.000012702 11 0.0000285556 12 0.0000283556 12 0.000028556 13 0.000028557 13 0.0000457819 13 0.0000722887 14 0.0000904406 15 0.000198225 15 0.000198225 16 0.000189225 17 0.00018158 17 0.000717674 18 0.00145113 18 0.00145113 18 0.0014513 19 0.00253865 19 0.00278949 20 0.00499491 20 0.00253865 20 0.001956 21 0.0101673 22 0.0009925 23 0.0002523 24 0.0002523 25 0.171153 26 0.339115 27 0.0339115 28 1.28649 28 1.48561 29 2.66592 29 2.66592 29 2.66879 30 0.00028869 30 0.0000293 31 10.1126 31 0.0000738636 | Elementos (n) | Tiempo (s) | Elementos (n) | Tiempo (s) |
| 10 0.0000528653 10 0.0000145867 11 0.0000145867 11 0.000012702 11 0.000028526 12 0.0000243599 12 0.0000469821 13 0.0000457819 13 0.0000722887 14 0.0000904,006 14 0.000106264 15 0.000189225 15 0.000189225 16 0.000189225 15 0.000189225 16 0.00018525 17 0.00018518 17 0.000717674 18 0.000181513 18 0.00142487 19 0.00253865 19 0.00278949 20 0.00499491 20 0.00253469 22 0.000954407 21 0.0101673 22 0.0009575 22 0.00285254 23 0.00055865 21 0.0101673 22 0.00095865 21 0.0101673 22 0.00095865 21 0.0101673 22 0.00095865 21 0.0101673 22 0.00095865 21 0.0101673 22 0.00095865 21 0.0101673 22 0.00095865 21 0.0101673 22 0.00095865 21 0.0101673 22 0.00095865 21 0.0101673 22 0.00095865 21 0.00101675 22 0.00095865 24 0.0118287 25 0.071153 25 0.00005865 24 0.0118287 25 0.371153 25 0.207041 26 0.339115 26 0.344851 27 0.053015 27 0.05630 | 8 | 0.00000136207 | 8 | 0.0000037376 |
| 11 0.000012702 11 0.0000283526 12 0.0000460821 13 0.000045819 13 0.000045819 13 0.000045819 14 0.0000106264 15 0.00019825 15 0.000213395 16 0.000333459 17 0.00033345 16 0.00033345 17 0.00033345 17 0.00033345 18 0.000213395 19 0.00233465 19 0.00233465 19 0.00233465 19 0.00278949 20 0.00499491 20 0.00534607 21 0.0101673 22 0.000956 21 0.0101673 22 0.002056 21 0.0101673 22 0.00256 22 0 | 9 | 0.00000267907 | 9 | 0.00000737613 |
| 12 0.0000234959 13 0.0000457819 13 0.0000457819 14 0.0000904,406 15 0.000198225 16 0.000198225 17 0.00018158 17 0.00018158 17 0.00018158 17 0.00018169 18 0.00145713 18 0.00145713 19 0.00253865 19 0.00253865 20 0.004,994,91 20 0.0026,994,91 21 0.010156 22 0.0209075 22 0.023825,40 23 0.0402523 24 0.0878626 25 0.171153 26 0.339115 26 0.344851 27 0.633015 28 1.28649 29 2.66592 29 2.66592 29 2.66879 30 5.0000026865 | 10 | 0.00000528653 | 10 | 0.0000145867 |
| 13 0.0000457819 14 0.0000904,006 14 0.000106264, 15 0.000198225 15 0.000198225 16 0.000333459 17 0.0008158 17 0.000333459 19 0.00253865 19 0.000278949 20 0.004994,91 20 0.00278949 20 0.004994,91 20 0.00534407 21 0.0101055 22 0.000975 22 0.0209075 23 0.0402523 24 0.0555082 25 0.171153 26 0.344851 27 0.633015 26 0.344851 27 0.633015 27 0.76131 28 1.28649 29 2.66592 29 2.66879 30 5.05092 30 5.05092 31 0.000025887 | 11 | 0.0000112702 | 11 | 0.0000283526 |
| 14 0.00019825 15 0.000106264 15 0.0001106264 15 0.00018282 15 0.000213395 16 0.000213395 17 0.000213395 17 0.00033459 17 0.00033459 17 0.00017674 18 0.00145713 18 0.0014287 19 0.0023865 19 0.00278949 20 0.00499491 20 0.00238467 21 0.0101673 22 0.0209075 22 0.0238254 23 0.0402523 23 0.0402523 23 0.0402523 24 0.085626 24 0.112827 25 0.27017153 25 0.207041 26 0.339115 26 0.344851 27 0.633015 27 0.761311 28 1.28649 28 1.28649 28 1.47561 29 2.60592 30 5.41493 31 10.1126 31 9.82069 | 12 | 0.0000234959 | 12 | 0.0000460821 |
| 15 0.000198225 15 0.000213395 16 0.000213395 16 0.000439214 16 0.000353459 17 0.0008158 17 0.000313459 18 0.00142487 18 0.0014513 18 0.00142487 19 0.00253865 19 0.00278949 20 0.00534407 21 0.000156 21 0.001673 22 0.0209075 22 0.0238254 23 0.0402523 23 0.0555082 24 0.0878626 24 0.11827 25 0.171153 25 0.207041 26 0.339115 26 0.344851 27 0.633015 27 0.761311 28 1.28649 28 1.41561 29 2.66592 29 2.66879 30 5.41493 31 10.1126 31 9.82069 | 13 | 0.0000457819 | 13 | 0.0000722887 |
| 16 0.000439214 17 0.00088158 18 0.00145113 18 0.00145113 19 0.00253865 19 0.00278949 20 0.00499491 20 0.00253865 21 0.010165 21 0.010165 21 0.010165 22 0.0209075 22 0.0238254 23 0.0402523 23 0.0402523 23 0.0402523 24 0.0878626 24 0.112827 25 0.171153 25 0.207041 26 0.339115 27 0.633015 27 0.633015 28 1.28649 28 1.41561 29 2.60592 29 2.66879 30 5.05092 30 5.41493 31 10.1126 31 9.82069 | 14 | 0.0000904406 | 14 | 0.000106264 |
| 17 0.00088158 17 0.000717674 18 0.00145113 18 0.00142487 19 0.00253865 19 0.00278949 20 0.00499491 20 0.00534407 21 0.0100156 21 0.0101673 22 0.0209075 22 0.0238254 23 0.0402523 23 0.0555082 24 0.0878626 24 0.112827 25 0.0711153 25 0.207041 26 0.339115 26 0.344851 27 0.633015 27 0.761311 28 1.28649 28 1.47561 29 2.60592 29 2.68719 30 5.05092 30 5.41493 31 10.1126 31 9.82069 | 15 | 0.000198225 | 15 | 0.000213395 |
| 18 0.0014513 18 0.00142487 19 0.00253865 19 0.00278949 20 0.00499491 20 0.00534407 21 0.000156 21 0.010167 22 0.0209075 22 0.0238254 23 0.0402523 23 0.0555082 24 0.0878626 24 0.112827 25 0.171153 25 0.207041 26 0.339115 26 0.344851 27 0.633015 27 0.761311 28 1.28649 28 1.41561 29 2.60592 29 2.68719 30 5.05092 30 5.41493 31 10.1126 31 9.82069 | 16 | 0.000439214 | 16 | 0.000353459 |
| 19 0.00253865 19 0.00278949 20 0.00499491 20 0.00534407 21 0.0100156 21 0.010167 22 0.029975 22 0.0238254 23 0.0402523 23 0.0555082 24 0.0878626 24 0.112827 25 0.171153 25 0.207041 26 0.339115 26 0.344851 27 0.633015 27 0.76131 28 1.28649 28 1.41561 29 2.60592 29 2.68719 30 5.05092 30 5.41493 31 10.1126 31 9.82069 | 17 | 0.00088158 | 17 | 0.000717674 |
| 20 0.00499491 20 0.00534407 21 0.000365 22 0.0209075 22 0.0238254 23 0.0402523 23 0.0555082 24 0.0878626 24 0.112827 25 0.717153 25 0.207041 26 0.339115 26 0.344851 27 0.633015 27 0.761311 28 1.28649 28 1.47561 29 2.60592 29 2.68719 30 5.05092 30 5.41493 31 10.1126 31 9.82069 | 18 | 0.00145113 | 18 | 0.00142487 |
| 21 0.0100156 21 0.0101673 22 0.0299075 22 0.0238254 23 0.0402523 23 0.0555082 24 0.0878626 24 0.112827 25 0.717153 25 0.207041 26 0.339115 26 0.344851 27 0.633015 27 0.761311 28 1.28649 28 1.41561 29 2.60592 29 2.68719 30 5.05092 30 5.41493 31 10.1126 31 9.82069 | 19 | 0.00253865 | 19 | 0.00278949 |
| 22 0.0209075 22 0.0238254 23 0.0402523 23 0.0555082 24 0.0878626 24 0.112827 25 0.171153 25 0.207041 26 0.339115 26 0.344851 27 0.633015 27 0.761311 28 1.28649 28 1.41561 29 2.60592 29 2.68719 30 5.05092 30 5.41493 31 10.1126 31 9.82069 | 20 | 0.00499491 | 20 | 0.00534407 |
| 23 0.0402523 23 0.0555082 24 0.0878626 24 0.112827 25 0.171153 25 0.207041 26 0.339115 26 0.346851 27 0.633015 27 0.761311 28 1.28649 28 1.41561 29 2.60592 29 2.68719 30 5.05092 30 5.41493 31 10.1126 31 9.82069 | 21 | 0.0100156 | 21 | 0.0101673 |
| 24 0.0878626 24 0.112827 25 0.171153 25 0.2070041 26 0.339115 26 0.344851 27 0.633015 27 0.761311 28 1.28669 28 1.41561 29 2.60592 29 2.68719 30 5.05092 30 5.41493 31 10.1126 31 9.82069 | 22 | 0.0209075 | 22 | 0.0238254 |
| 25 0.171153 25 0.207041 26 0.339115 26 0.344851 27 0.633015 27 0.761311 28 1.28649 28 1.41561 29 2.60592 29 2.68719 30 5.05092 30 5.41493 31 10.1126 31 9.82069 | 23 | 0.0402523 | 23 | 0.0555082 |
| 26 0.339115 26 0.344851 27 0.633015 27 0.761311 28 1.28649 28 1.41561 29 2.60592 29 2.68719 30 5.05092 30 5.41493 31 10.1126 31 9.82069 | 24 | 0.0878626 | 24 | 0.112827 |
| 27 0.633015 27 0.761311 28 1.28649 28 1.41561 29 2.60592 29 2.68719 30 5.05092 30 5.44693 31 10.1126 31 9.82069 | 25 | 0.171153 | 25 | 0.207041 |
| 28 1.28649 28 1.41561 29 2.60592 29 2.68719 30 5.05092 30 5.41493 31 10.1126 31 9.82069 | 26 | 0.339115 | 26 | 0.344851 |
| 29 2.60592 29 2.68719 30 5.05092 30 5.41493 31 10.1126 31 9.82069 | 27 | 0.633015 | 27 | 0.761311 |
| 30 5.05092 30 5.41493 31 10.1126 31 9.82069 | 28 | 1.28649 | 28 | 1.41561 |
| 31 10.1126 31 9.82069 | 29 | 2.60592 | 29 | 2.68719 |
| | 30 | 5.05092 | 30 | 5.41493 |
| 32 20.301 32 20.2358 | 31 | 10.1126 | 31 | 9.82069 |
| | 32 | 20.301 | 32 | 20.2358 |

| Ordenador Moya | | |
|----------------|---------------|--|
| Elementos (n) | Tiempo (s) | |
| 8 | 0.0000017978 | |
| 9 | 0.00000348253 | |
| 10 | 0.00000737093 | |
| 11 | 0.0000137999 | |
| 12 | 0.0000274451 | |
| 13 | 0.0000548052 | |
| 14 | 0.000110116 | |
| 15 | 0.000198426 | |
| 16 | 0.000427075 | |
| 17 | 0.000796963 | |
| 18 | 0.00159355 | |
| 19 | 0.00321857 | |
| 20 | 0.00633508 | |
| 21 | 0.012697 | |
| 22 | 0.0253476 | |
| 23 | 0.0506946 | |
| 24 | 0.101314 | |
| 25 | 0.202542 | |
| 26 | 0.405264 | |
| 27 | 0.809707 | |
| 28 | 1.6195 | |
| 29 | 3.23942 | |
| 30 | 6.47798 | |
| 31 | 12.9623 | |

Tabla 4: Experiencia empírica de algoritmo de Hanoi sin optimizar



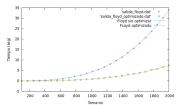
Hanoi, Eficiencia híbrida

- i7-6700 3.40GHz $\rightarrow T_1(n) = 4.72408 \cdot 10^{-9} \cdot 2^X$.
- i5-1095G1 1.00 GHz $\rightarrow T_2(n) = 4.70707 \cdot 10^{-9} \cdot 2^{x}$.
- Ordenador Moya $\to T_3(n) = 6.03512 \cdot 10^{-9} \cdot 2^X$.

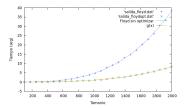
Varianza residual:

- $T_1(n) \longrightarrow Var.res = 0.000302074$
- $T_2(n) \longrightarrow Var.res = 0.0113386$
- $T_3(n) \longrightarrow Var.res = 3.87795 \cdot 10^{-7}$

Casos especiales

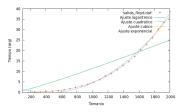


Intel i7-6700 3.40 GHz

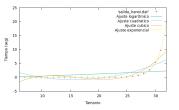


i5-1095G1 1.00 GHz

Casos especiales. Otros posibles ajustes funcionales



Intel i7-6700 3.40 GHz



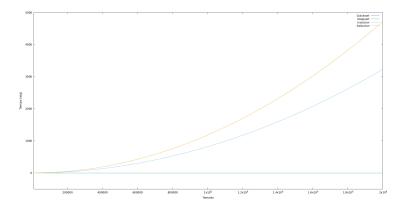
Casos especiales 000000000000

i5-1095G1 1.00 GHz

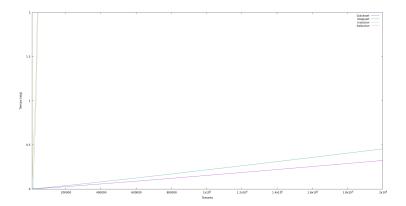
Comparativa de los algoritmos de ordenación

Una vez estudiados inserción, selección, quicksort y heapsort, procedemos a compararlos gráficamente y sacar conclusiones

Casos especiales



Comparativa de los algoritmos de ordenación



Comparativa de los algoritmos de ordenación

Las conclusiones que sacamos son:

- Se verifica que los algoritmos $\in O(n \cdot log(n))$ son más eficientes que los algoritmos $\in O(n^2)$)
- Es tal la diferencia que las gráficas de quicksort y heapsort parecen paralelas al eje X
- Quicksort y Heapsort no pasan del segundo por ejecución, mientras que Inserción y Selección para 2000000 requieren más de 5000 segundos por ejecución

Para el peor caso, tomamos un vector ordenado a la inversa:

```
for (int i = 0; i < n; i++)

{
     T[i] = n - i;
};</pre>
```

Para el mejor caso, tomamos un vector ordenado correctamente:

```
for (int i = 0; i < n; i++)

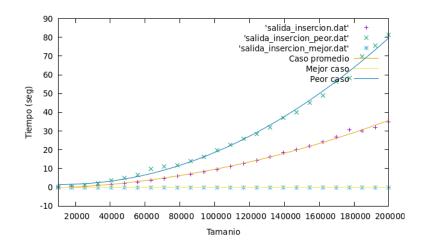
T[i] = i;
};</pre>
```

Los datasets de los dos casos extremos para inserción fueron:

| Peor caso de inserción | | |
|------------------------|------------|--|
| Elementos (n) | Tiempo (s) | |
| 17600 | 0.604135 | |
| 25200 | 1.23844 | |
| 32800 | 2193 | |
| 40400 | 3.83771 | |
| 48000 | 4.92643 | |
| 55600 | 6.65173 | |
| 63200 | 9.69109 | |
| 70800 | 10.9557 | |
| 78400 | 11.5942 | |
| 86000 | 14.0269 | |
| 93600 | 16.1625 | |
| 101200 | 19.6924 | |
| 108800 | 22.6823 | |
| 116400 | 25.7428 | |
| 124000 | 28.4764 | |
| 131600 | 32.0522 | |
| 139200 | 37.1527 | |
| 146800 | 40.1051 | |
| 154400 | 45.1864 | |
| 162000 | 49014 | |
| 169600 | 56.3554 | |
| 177200 | 58.3219 | |
| 184800 | 69.8676 | |
| 192400 | 75.6931 | |
| 200000 | 81.4641 | |

| Mejor caso | de inserción |
|---------------|--------------|
| Elementos (n) | Tiempo (s) |
| 17600 | 0.000119171 |
| 25200 | 0.000154839 |
| 32800 | 0.000254579 |
| 40400 | 0.000268849 |
| 48000 | 0.000315367 |
| 55600 | 0.000221064 |
| 63200 | 0.000253476 |
| 70800 | 0.000218935 |
| 78400 | 0.000203775 |
| 86000 | 0.000202846 |
| 93600 | 0.000349709 |
| 101200 | 0.000310166 |
| 108800 | 0.000319687 |
| 116400 | 0.000467919 |
| 124000 | 0.000452129 |
| 131600 | 0.000604762 |
| 139200 | 0.000651269 |
| 146800 | 0.00058346 |
| 154400 | 0.000549033 |
| 162000 | 0.000648826 |
| 169600 | 0.000551999 |
| 177200 | 0.000769622 |
| 184800 | 0.00049894 |
| 192400 | 0.000406281 |
| 200000 | 0.000481777 |

Tabla 5: Datasets de la ejecución del peor y mejor caso para Inserción

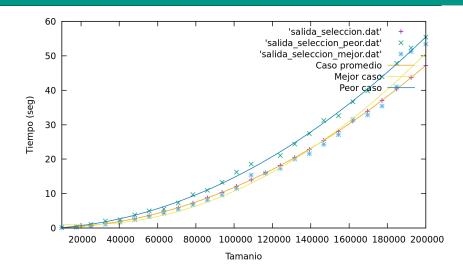


Los datasets de los dos casos extremos para selección fueron:

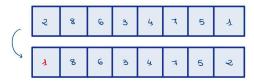
| Peor caso de selección | | |
|------------------------|------------|--|
| Elementos (n) | Tiempo (s) | |
| 17600 | 0.43298 | |
| 25200 | 1.05541 | |
| 32800 | 2.06101 | |
| 40400 | 2.43286 | |
| 48000 | 3.83567 | |
| 55600 | 4.94909 | |
| 63200 | 5.53895 | |
| 70800 | 7.26986 | |
| 78400 | 9.73586 | |
| 86000 | 10.9612 | |
| 93600 | 13.2779 | |
| 101200 | 16.2492 | |
| 108800 | 18.5379 | |
| 124000 | 21.05 | |
| 131600 | 24.3881 | |
| 139200 | 27383 | |
| 146800 | 31.2269 | |
| 154400 | 32.5782 | |
| 162000 | 36.6571 | |
| 169600 | 39.8171 | |
| 177200 | 43.9252 | |
| 184800 | 47.8018 | |
| 192400 | 52.3513 | |
| 200000 | 55.4702 | |

| Mejor caso de selección | | |
|-------------------------|------------|--|
| Elementos (n) | Tiempo (s) | |
| 17600 | 0.280803 | |
| 25200 | 0.583418 | |
| 32800 | 1.09678 | |
| 40400 | 1.68938 | |
| 48000 | 2.4139 | |
| 55600 | 3.26631 | |
| 63200 | 4.24695 | |
| 70800 | 5477 | |
| 78400 | 6.74688 | |
| 86000 | 8.11066 | |
| 93600 | 9.54485 | |
| 101200 | 11.4866 | |
| 108800 | 15.3544 | |
| 116400 | 15.8972 | |
| 124000 | 17.35 | |
| 131600 | 20.0171 | |
| 139200 | 21.5481 | |
| 146800 | 24.2838 | |
| 154400 | 27.0934 | |
| 162000 | 31.5408 | |
| 169600 | 32.8615 | |
| 177200 | 35.3771 | |
| 184800 | 40.99 | |
| 192400 | 51.2152 | |
| 200000 | 53.4133 | |

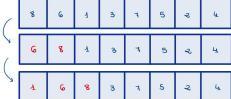
Tabla 6: Datasets de la ejecución del peor y mejor caso para Selección



¿A qué se debe este comportamiento?



Inserción



Conclusiones

Conclusiones

El análisis híbrido nos confirma nuestro análisis teórico observando el coeficiente de regresión.

Lo que más influye en el tiempo es el orden de eficiencia del algoritmo.

Diversidad de agentes tecnológicos: diferentes computadores y arquitecturas da lugar a resultados distintos.