Universidad de Granada

Algoritmos de Ordenación con estructuras jerárquicas

Algorítmica

Lukas Häring García 2ºD

Tabla de contenidos

Árbol binario de búsqueda

0.1 Código

```
1  using namespace std;
2
3  int main(int argc, char * argv[])
4  {
5    int n = atoi(argv[1]);
6    int* A = new int[n];
7    ABB<int> ab_bus;
8    srand(time(0));
9    // Introducimos
10   for (int i = 0; i < n; i++){
11     ab_bus.Insertar(rand());
12   }
13
14   ABB<int>::nodo k;
15   int m = 0;
16   clock_t t_antes = clock();
17
18   /*
19     Generar codigo O(1) que el compilador no va a eliminar o
20     desenrrollar.
21   */
22   for (k = ab_bus.begin(); k != ab_bus.end(); ++k){
23
24     A[m++] = *k;
25   }
26   clock_t t_despues = clock();
27   cout << n << "  " << ((double)(t_despues - t_antes)) /
       CLOCKS_PER_SEC << endl;
28
29   return 0;
30 };
```

0.2 Eficiencia

0.2.1 Teórica

Para analizar el código, debemos suponer cuál es la forma de dicha estructura tras introducir datos, a simple vista, este parece tener $O(n)$, pero en realidad, hay que observar el operador incremento y el método end(Tras observar el código, vemos que es O(1)):

1. **El peor caso**. La altura del árbol es la misma que el número de elementos: Por tanto, este deberá bajar hasta la última posición, coste $O(n)$ y luego recorrerlo hacia atrás, por lo que también sería $O(n)$.

2. **El mejor caso**. Esto ocurre cuando hay números en diferentes alturas, este tendrá que bajar como máximo a una altura $log_2(d)$ donde d es la altura, pero al tener que recorrer todos los elementos, $O(log_2(n)+n) \in O(n)$

Concluimos que su eficiencia es $\Theta(n)$ pues ambos (peor y mejor caso) coinciden.

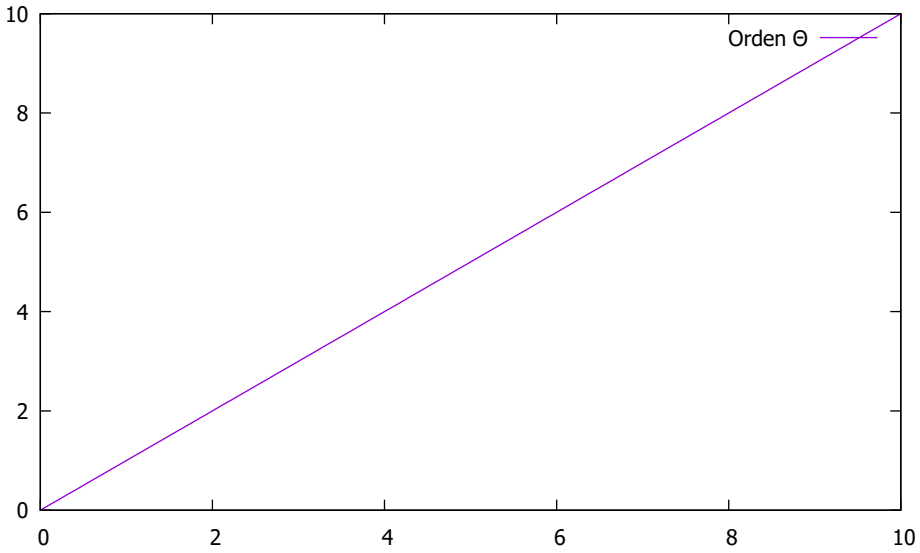
0.2.2 Empírica

Realizando el algoritmo de ordenación por inserción desde 0 dando pasos de 100 hasta 10000 elementos y realizando un ajuste con gnuplot, calculando las variables ocultas, obtenemos una eficiencias de:

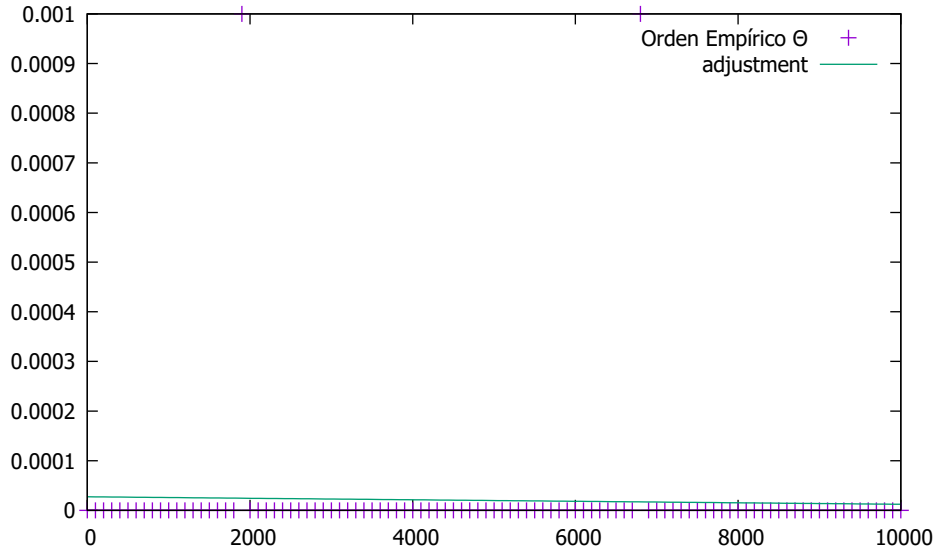
$$f(n) = -1.51427 \cdot 10^{-9} \cdot n + 2.73733 \cdot 10^{-5} \in \Theta(n)$$

0.3 Gráficas

0.3.1 Teórica



0.3.2 Empírica



Árbol parcialmente ordenado

1.1 Código

```
1  int main(int argc, char * argv[])
2  {
3    int n = atoi(argv[1]);
4    int* A = new int[n];
5    APO<int> apo_tree;
6    srand(time(0));
7    // Introducimos
8    for (int i = 0; i < n; i++){
9      apo_tree.insertar(rand());
10   }
11
12   int m = 0;
13   clock_t t_antes = clock();
14   while(!apo_tree.vacio()){
15     apo_tree.borrar_minimo();
16   }
17   clock_t t_despues = clock();
18   cout << n << "  " << ((double)(t_despues - t_antes)) /
       CLOCKS_PER_SEC << endl;
19
20   return 0;
21 };
```

1.2 Eficiencia

1.2.1 Teórica

De la misma forma, el recorrer el arbol es $O(n)$, pero quién tiene importancia son los métodos insertar, que son más complejos.

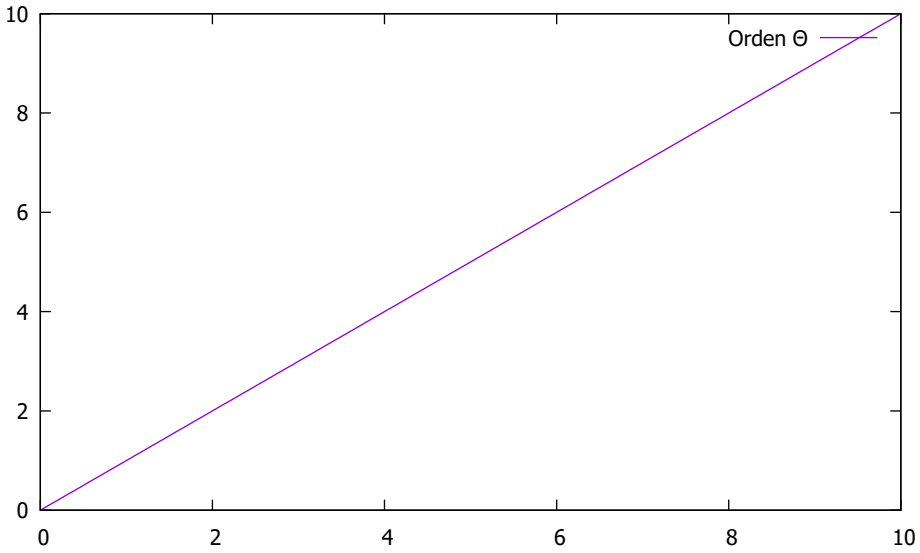
1.2.2 Empírica

Realizando el algoritmo de ordenación por selección desde 0 dando pasos de 100 hasta 10000 elementos y realizando un ajuste con gnuplot, calculando las variables ocultas, obtenemos una eficiencias de:

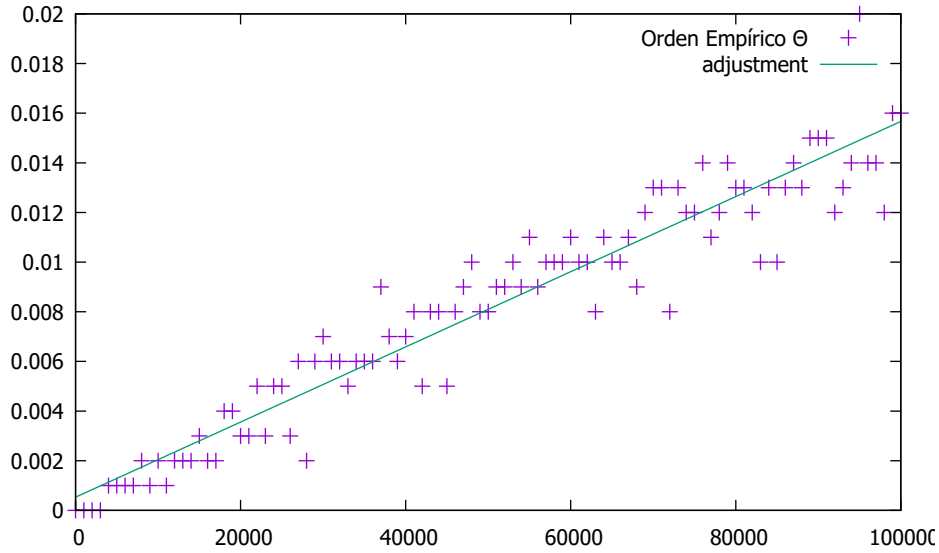
$$f(n) = 1.5138 \cdot 10^{-7} \cdot n + 0.000529994 \in \Theta(n)$$

1.3 Gráficas

1.3.1 Teórica



1.3.2 Empírica



Especificaciones

1. Windows 10.0.14393
2. Procesador Intel(R) Core(TM) i7-7800X CPU @ 3.50GHz, 3504 Mhz
3. 6 procesadores principales.
4. 12 procesadores lógicos.
5. Memoria física instalada (RAM) 8,00 GB x 2
6. Compilador MinGW.