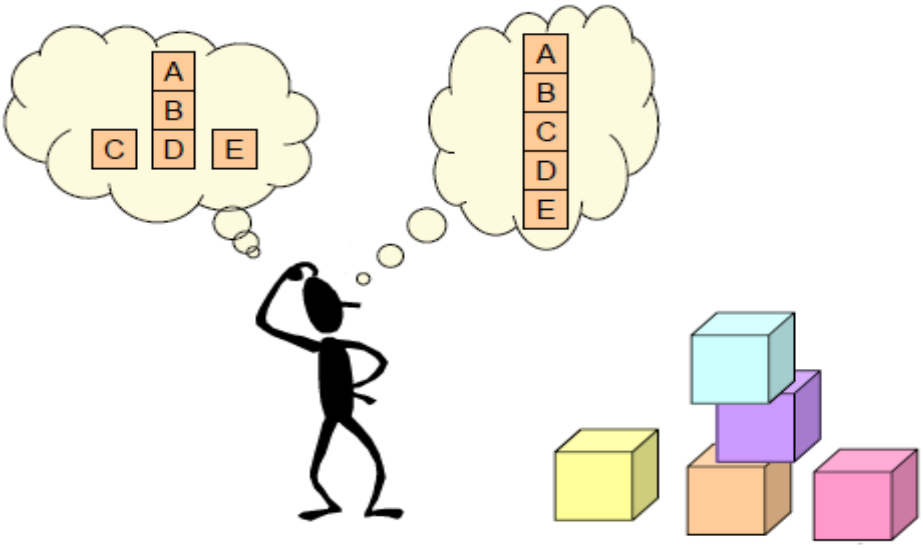
ESTUDIO EMPÍRICO DE RENDIMIENTO DE LOS ALGORITMOS DE ORDENACIÓN:

Algoritmo y estructuras de datos avanzado



19 DE ABRIL DE 2018

KATHRINA ARROCHA UMPIÉRREZ
alu0100913293@ull.edu.es

En este estudio empírico queremos observar los cambios producidos al incrementar el tamaño del array a ordenar. Para realizar el estudio, se ha requerido desarrollar un programa en lenguaje C++ que cuente el número de operaciones de comparación de clave que se realizan durante la ordenación de un array. Para ello, el programa utilizará un contador de comparaciones, que se inicializa a cero antes de la ejecución de cada método de ordenación y se incrementa con cada ejecución de una operación de comparación de claves. Al finalizar la ordenación el valor contenido en el contador se utiliza para actualizar una estadística que registra los valores mínimos, máximos y medias del número de comparaciones obtenidos. Para que estos valores estadísticos sean significativos el experimento de ordenación debe repetirse un número suficiente de veces. A continuación, mostraremos un modelo de visualización obtenidos en la ejecución del programa:

	Número de Comparaciones		
	Mínimo	Medio	Máximo
Método 1	xxxx	xxxx	xxxx
Método 2	xxxx	xxxx	xxxx
...			

***

En este estudio, será importante destacar la importancia de la complejidad de cada uno de los algoritmos a visualizar, dichos algoritmos son: inserción, método de la burbuja, heapsort, Quicksort y shellshort. A este último algoritmo, le insertaremos distintos valores de alfa para comprobar los cambios que acontecen.

- Algoritmo de inserción:

  La idea de este algoritmo de ordenación consiste en ir insertando un elemento de la lista o un arreglo en la parte ordenada de la misma, asumiendo que el primer elemento es la parte ordenada, el algoritmo ira comparando un elemento de la parte desordenada de la lista con los elementos de la parte ordenada, insertando el elemento en la posición correcta dentro de la parte ordenada, y así sucesivamente hasta obtener la lista ordenada.

  Rendimiento:

  - En el caso óptimo, con los datos ya ordenados, el algoritmo sólo efectuará n comparaciones. Por lo tanto, la complejidad en el caso óptimo es en Θ(n).
  - En el caso desfavorable, con los datos ordenados a la inversa, se necesita realizar (n-1) + (n-2) + (n-3)... + 1 comparaciones e intercambios, o (n2-n) / 2. Por lo tanto, la complejidad es en Θ(n2).
  - En el caso medio, la complejidad de este algoritmo es también en Θ(n2)

- Algoritmo Bubblesort:

  La Ordenación de burbuja funciona revisando cada elemento de la lista que va a ser ordenada con el siguiente, intercambiándolos de posición si están en el

orden equivocado. Es necesario revisar varias veces toda la lista hasta que no se necesiten más intercambios, lo cual significa que la lista está ordenada.

Rendimiento:

- En el caso óptimo, con los datos ya ordenados, el algoritmo sólo efectuará n comparaciones. Por lo tanto, la complejidad en el caso óptimo es en Θ(n).
- En el caso desfavorable, con los datos ordenados a la inversa, la complejidad es en Θ(n2).
- En el caso medio, la complejidad de este algoritmo es también en Θ(n2).

- Algoritmo Heapsort:

El Heapsort ordena un vector de n elementos construyendo un heap con los n elementos y extrayéndolos, uno a uno del heap a continuación. El propio vector que almacena a los n elementos se emplea para construir el heap, de modo que heapsort actúa in-situ y sólo requiere un espacio auxiliar de memoria constante.

Rendimiento:

- El coste de este algoritmo es (n log n) (incluso en caso mejor) si todos los elementos son diferentes.

- Algoritmo Quicksort:

El ordenamiento rápido (quicksort en inglés) es un algoritmo basado en la técnica de divide y vencerás.

Rendimiento:

- El tiempo de ejecución promedio es O(n log (n)).
- En el peor de los casos, el tiempo de ejecución es de O(n log (n)).

- Algoritmo Shellsort:

El algoritmo Shell es una mejora de la ordenación por inserción, donde se van comparando elementos distantes, al tiempo que se los intercambian si corresponde. A medida que se aumentan los pasos, el tamaño de los saltos disminuye; por esto mismo, es útil tanto como si los datos desordenados se encuentran cercanos, o lejanos.

Rendimiento:

- Dependiendo de la elección de la secuencia de espacios, Shell sort tiene un tiempo de ejecución en el peor caso de O(n2) (usando los incrementos de Shell que comienzan con 1/2 del tamaño del vector y se dividen por 2 cada vez), O(n3 / 2) (usando los incrementos de Hibbard

de 2k − 1), O(n4 / 3) (usando los incrementos de Sedgewick de 9(4i) − 9(2i) + 1, o 4i + 1 + 3(2i) + 1), o O(nlog2n), y posiblemente mejores tiempos de ejecución no comprobados. La existencia de una implementación O(nlogn) en el peor caso del Shell sort permanece como una pregunta por resolver.

A continuación, podremos observar algunos ejemplos variando los datos para poder así ver los cambios pertinentes y las diferencias entre los distintos algoritmos.

```
Ingrese el tamaño del vector de prueba:
100

Ingrese el numero de pruebas a realizar:
10

Ingrese la constante alpha para el algoritmo shellsort:
0.4

**********************************************************************
MODO ESTADISTICO

                      Minimo          Maximo          Media
Insercion             2395            2765            2580

Burbuja               4950            4950            4950

Heapsort              1002            1039            1020

Quicksort             697             790             743

Shellsort             3835            4570            4202
```

```
Ingrese el tamaño del vector de prueba:
100

Ingrese el numero de pruebas a realizar:
50

Ingrese la constante alpha para el algoritmo shellsort:
0.8

**********************************************************************
MODO ESTADISTICO

                      Minimo          Maximo          Media
Insercion             2238            2866            2552

Burbuja               4950            4950            4950

Heapsort              1004            1043            1023

Quicksort             636             872             754

Shellsort             3399            4256            3827
```

```
Ingrese el tamaño del vector de prueba:
1000

Ingrese el numero de pruebas a realizar:
10

Ingrese la constante alpha para el algoritmo shellsort:
0.4

**********************************************************************
MODO ESTADISTICO

                      Minimo          Maximo          Media
Insercion             30070145        30687976        30379060

Burbuja               60494500        60494500        60494500

Heapsort              261837          262249          262043

Quicksort             173633          198083          185858

Shellsort             50821162        51950464        51385813
```

```
Ingrese el tamaño del vector de prueba:
1000

Ingrese el numero de pruebas a realizar:
10

Ingrese la constante alpha para el algoritmo shellsort:
0.8

**********************************************************************
MODO ESTADISTICO

                      Minimo          Maximo          Media
Insercion             242194          256204          249199

Burbuja               499500          499500          499500

Heapsort              16830           16903           16866

Quicksort             11311           13941           12626

Shellsort             295655          312533          304094
```

Tras realizar algunos ejemplos, hemos comprobado que en la mayoría de casos obtenemos peores resultados con el método de burbuja y mejores al usar el Quicksort.