**Shell Sort**

**Como Funciona:**



Consiste en comparar los elementos con saltos de mayor tamaño, pero con incrementos decrecientes; así, los elementos quedaran ordenados en el arreglo más rápidamente.

1. Se compara un elemento x con otro elemento y que está en la posición del doble de este elemento x seleccionado.
2. Compara los elementos y los intercambia si es necesario.
3. Luego disminuye la cantidad del incremento y repite el proceso hasta quedar ordenado.
4. Repite este proceso hasta terminar el arreglo.

**Conseguí el algoritmo de:**

<http://www.w3resource.com/csharp-exercises/searching-and-sorting-algorithm/searching-and-sorting-algorithm-exercise-1.php>

**Cantidad de líneas de código por clases:**

shellSort(); = 20

impresionInversoShellSort(); = 17

impresionAleatorioShellSort(); = 17

impresionAscendenteShellSort(); = 17

impresionTotalShellSort(); = 22

**Clase Consultas:**

Arrays de uso general; = 4

Inverso(); = 3

Aleatorio(); = 6

scendente(); = 3

VerificarOrden(); = 8

**Clase Program:**

Instanciaciones; 2

Medición empírica

* 1. Nombre del algoritmo de ordenamiento #1.0: ShellSort-Inverso

| Operaciones | Tamaños del arreglo | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 10 | 100 | 1000 | 5000 | 10000 | 50000 |
| Asignaciones | 299944 | 299404 | 294004 | 270004 | 240004 | 4 |
| Comparaciones | 4 | 4 | 4 | 4 | 4 | 4 |
| Cantidad de líneas ejecutadas | 299948 | 299408 | 294008 | 270008 | 240008 | 8 |
| Tiempo de ejecución | 0,00 s | 0,00 s | 0,00 s | 0,03 s | 0,07 s | 0,05 |
| Cantidad de líneas del código | shellSort(); = 20 | | | | | | |

* 1. Nombre del algoritmo de ordenamiento #1.1: ShellSort-Aleatorio

| Operaciones | Tamaños del arreglo | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 10 | 100 | 1000 | 5000 | 10000 | 50000 |
| Asignaciones | 299944 | 299404 | 294004 | 270004 | 240004 | 4 |
| Comparaciones | 4 | 4 | 4 | 4 | 4 | 4 |
| Cantidad de líneas ejecutadas | 299948 | 299408 | 294008 | 270008 | 240008 | 8 |
| Tiempo de ejecución | 0,01 s | 0,02 s | 0,03 s | 0,05 s | 0,06 s | 0,08 s |
| Cantidad de líneas del código | shellSort(); = 20 | | | | | | |

* 1. Nombre del algoritmo de ordenamiento #1.2: ShellSort-Ascendente

| Operaciones | Tamaños del arreglo | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 10 | 100 | 1000 | 5000 | 10000 | 50000 |
| Asignaciones | 299944 | 299404 | 294004 | 270004 | 240004 | 4 |
| Comparaciones | 4 | 4 | 4 | 4 | 4 | 4 |
| Cantidad de líneas ejecutadas | 299948 | 299408 | 294008 | 270008 | 240008 | 8 |
| Tiempo de ejecución | 0,00 s | 0,02 s | 0,04 s | 0,05 s | 0,06 s | 0,07 s |
| Cantidad de líneas del código | shellSort(); = 20 | | | | | | |

Determinar el factor de crecimiento

3.1Nombre del algoritmo de ordenamiento #1: ShellSort

| Talla | | Factor talla | Factor Asig | Factor Comp | Factor Cantidad de líneas ejecutadas | Factor Tiempo de ejecución |
| --- | --- | --- | --- | --- | --- | --- |
| De 10 a 100 | aleatorio | 10 | 0.99 | 1 | 0.99 | 2 |
| orden | 0.99 | 1 | 0.99 | 2 |
| inverso | 0.99 | 1 | 0.99 | 0 |
| De 100 a 1000 | aleatorio | 10 | 0.98 | 1 | 0.98 | 1.5 |
| orden | 0.98 | 1 | 0.98 | 2 |
| inverso | 0.98 | 1 | 0.98 | 0 |
| De 1000 a 5000 | aleatorio | 5 | 0.91 | 1 | 0.91 | 1.66 |
| orden | 0.91 | 1 | 0.91 | 1.25 |
| inverso | 0.91 | 1 | 0.91 | 0.03 |
| De 5000 a 10000 | aleatorio | 2 | 0.88 | 1 | 0.88 | 1.2 |
| orden | 0.88 | 1 | 0.88 | 1.2 |
| inverso | 0.88 | 1 | 0.88 | 2.33 |
| De 10000 a 50000 | aleatorio | 5 | 1.66 | 1 | 3.33 | 1.33 |
| Orden | 1.66 | 1 | 3.33 | 1.16 |
| Inverso | 1.66 | 1 | 3.33 | 0.71 |
| De 1000 a 50000 | aleatorio | 50 | 1.36 | 1 | 2.72 | 0.05 |
| Orden | 1.36 | 1 | 2.72 | 1.75 |
| Inverso | 1.36 | 1 | 2.72 | 0.05 |

|  |  |  |
| --- | --- | --- |
| Clasificación del comportamiento de las **asignaciones** | Lineal |  |
| Clasificación del comportamiento de las **comparaciones** | Lineal |

|  |  |  |  |
| --- | --- | --- | --- |
| Clasificación según su orden de entrada de los datos use la notación  Theta, O Grande, y Omega según corresponda | | | |
| **Orden** | **Ordenados** | **Inverso** | **Aleatorio** |
| Clasificación |  |  |  |

Medición analítica

* 1. Nombre del algoritmo de ordenamiento #1: **ShelSort**

|  |  |
| --- | --- |
| Código fuente  Solo se analiza el código del método de ordenar. | Medición de líneas ejecutadas en el peor de los casos  (línea por línea) |
|  |  |
| static void shellSort(int[] arr, int array\_size)  {  int i, j, inc, temp;  inc = 3;  while (inc > 0)  {  for (i = 0; i < array\_size; i++)  {  j = i;  temp = arr[i];  while ((j >= inc) && (arr[j-inc] >temp))  {  arr[j] = arr[j-inc];  j = j - inc;  }  arr[j] = temp;  }  if (inc / 2 != 0)  {  inc = inc / 2;  }  else if (inc == 1)  {  inc = 0;  }  Else  {  inc = 1;  }  }  } | 1  1  2n+1  1  1  1  1  1  1  1  1 |
|  |  |
|  |  |
|  |  |
| Total (la suma de todos los pasos) |  |
| Clasificación en notación O Grande |  |