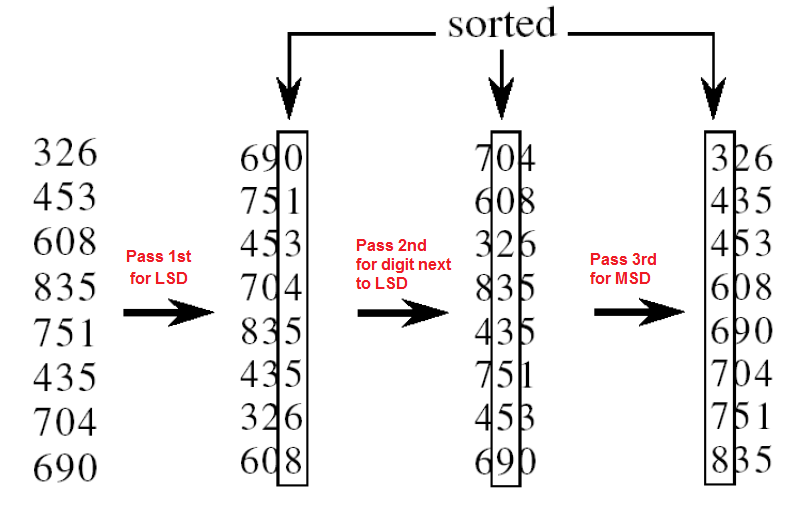
**RadixSort**

**Como Funciona:**



Inicia tomando un array de n cantidad de elementos, luego se recorre ese array con un ciclo y va obteniendo un numero n. Seguidamente se selecciona el digito menos significativo del número n y este es almacenado en un array, este los ordena con respecto al digito menos significativo (0-9), más adelante realiza un desplazamiento hacia la izquierda (ordenando) y repite ese proceso hasta llegar el digito más significativo.

**Conseguí el algoritmo de:**

http://www.w3resource.com/csharp-exercises/searching-and-sorting-algorithm/searching-and-sorting-algorithm-exercise-10.php

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

**Clase Radixsort:**

impresionInversoRadixSort(); = 17

impresionAleatorioRadixSort(); = 17

impresionAscendenteRadixSort(); = 17

RadixSor(); = 14

impresionTotalRadixSort(); = 24

**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: RadixSort-Inverso

| Operaciones | Tamaños del arreglo | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 10 | 100 | 1000 | 5000 | 10000 | 50000 |
| Asignaciones | 1650012 | 15150102 | 150151002 | 750155002 | 1500160002 | -1089734590 |
| Comparaciones | 550000 | 5050000 | 50050000 | 250050000 | 500050000 | -1794917296 |
| Cantidad de líneas ejecutadas | 2200012 | 20200102 | 200201002 | 1000205002 | 2000210002 | 1410315410 |
| Tiempo de ejecución | 0,01 s | 0,08 s | 0,88 s | 3,65 s | 8,74 s | 31,3 s |
| Cantidad de líneas del código | RadixSor();=14,impresionInversoRadixSort();=17, Inverso();=3,VerificarOrden();=8= 42 | | | | | | |

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

| Operaciones | Tamaños del arreglo | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 10 | 100 | 1000 | 5000 | 10000 | 50000 |
| Asignaciones | 1650012 | 15150102 | 150151002 | 750155002 | 1500160002 | -1089734590 |
| Comparaciones | 550000 | 5050000 | 50050000 | 250050000 | 500050000 | -1794917296 |
| Cantidad de líneas ejecutadas | 2200012 | 20200102 | 200201002 | 1000205002 | 2000210002 | 1410315410 |
| Tiempo de ejecución | 0,01 s | 0,07 s | 0,58 s | 3,00 s | 8,75 s | 43,6 s |
| Cantidad de líneas del código | RadixSor();=14,impresionAleatorioRadixSort();=17, Aletorio();=6,VerificarOrden();=8= 45 | | | | | | |

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

| Operaciones | Tamaños del arreglo | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 10 | 100 | 1000 | 5000 | 10000 | 50000 |
| Asignaciones | 1650012 | 15150102 | 150151002 | 750155002 | 1500160002 | -1089734590 |
| Comparaciones | 550000 | 5050000 | 50050000 | 250050000 | 500050000 | -1794917296 |
| Cantidad de líneas ejecutadas | 2200012 | 20200102 | 200201002 | 1000205002 | 2000210002 | 1410315410 |
| Tiempo de ejecución | 0,01 s | 0,07 s | 0,57 s | 3,03 s | 8,01 s | 31,1 s |
| Cantidad de líneas del código | RadixSor();=14,impresionAscendenteRadixSort();=17, Ascendente();=3,VerificarOrden();=8= 42 | | | | | | |

Determinar el factor de crecimiento

3.1Nombre del algoritmo de ordenamiento #1: RadixSort

| Talla | | Factor talla | Factor Asig | Factor Comp | Factor Cantidad de líneas ejecutadas | Factor Tiempo de ejecución |
| --- | --- | --- | --- | --- | --- | --- |
| De 10 a 100 | aleatorio | 10 | 9.18 | 9.18 | 9.18 | 7 |
| orden | 9.18 | 9.18 | 9.18 | 7 |
| inverso | 9.18 | 9.18 | 9.18 | 8 |
| De 100 a 1000 | aleatorio | 10 | 9.91 | 9.91 | 9.18 | 8.28 |
| orden | 9.91 | 9.91 | 9.18 | 8.14 |
| inverso | 9.91 | 9.91 | 9.18 | 11 |
| De 1000 a 5000 | aleatorio | 5 | 4.99 | 4.99 | 4.99 | 2.91 |
| orden | 4.99 | 4.99 | 4.99 | 5.31 |
| inverso | 4.99 | 4.99 | 4.99 | 2.39 |
| De 5000 a 10000 | aleatorio | 2 | 1.99 | 1.99 | 1.99 | 2.91 |
| orden | 0.50 | 1.99 | 1.99 | 2.64 |
| inverso | 1.99 | 1.99 | 1.99 | 4.14 |
| De 10000 a 50000 | aleatorio | 5 | -0.72 | 3.58 | 0.70 | 4.98 |
| Orden | 0.72 | -3.58 | 0.70 | 3.88 |
| Inverso | 0.72 | -3.58 | 0.70 | 3.58 |
| De 1000 a 50000 | aleatorio | 50 | -7.25 | -35.86 | 0.70 | 75.17 |
| Orden | 7.25 | -35.86 | 0.70 | 54.56 |
| Inverso | 7.25 | -35.86 | 0.70 | 35.56 |

|  |  |  |
| --- | --- | --- |
| 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 | O pequeña | O pequeña | O pequeña |

Medición analítica

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

|  |  |
| --- | --- |
| 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 Sort(int[] v, int tam)  {  int i, j;  int[] tmp = new int[v.Length];  for (int shift = tam; shift > -1; --shift)  {  j = 0;  for (i = 0; i < v.Length; ++i)  {  bool move = (v[i] << shift) >= 0;  if (shift == 0 ? !move : move)  v[i - j] = v[i];  else  tmp[j++] = v[i];  }  Array.Copy(tmp, 0, v, v.Length - j, j);  }  } | 1  1  2n+2  1  (2n+2)n  n\*n  1  1  1 |
|  |  |
|  |  |
|  |  |
| Total (la suma de todos los pasos) | 1+1+(2n+2)+1+(2n+2)\*n+(n\*n)+1+1+1=3n2+4n+8 |
| Clasificación en notación O Grande |  |