Algoritmos de Ordenamiento II

Victor Hugo Flores Márquez
Universidad de Artes Digitales

Guadalajara, Jalisco

Email: idv16c.vflores@uartesdigitales.edu.mx

Profesor: Efraín Padilla

Mayo 18, 2019

I. Introducción

En computación y matemáticas un algoritmo de ordenamiento es un algoritmo que pone elementos de una lista o un vector en una secuencia dada por una relación de orden, es decir, el resultado de salida ha de ser una re-ordenamiento de la entrada que satisfaga la relación de orden dada.

1) **Countingsort**

Es un algoritmo de ordenamiento en el que se cuenta el número de elementos de cada clase para luego ordenarlos. Sólo puede ser utilizado por tanto para ordenar elementos que sean contables (como los números enteros en un determinado intervalo, pero no los números reales, por ejemplo).

Ejemplo:

```
int input[] = { 2, 1, 4, 5, 7, 1, 7, 11, 8, 9, 10 };
```

Index	0	1	2	3	4	5	6	7	8	9	10	11
Count[]	0	2	1	0	1	1	0	2	1	1	1	1

Index	0	1	2	3	4	5	6	7	8	9	10	11
Modified Count[]	0	2	3	3	4	5	5	7	8	9	10	11

Count[i]=Count[i] + Count[i-1]

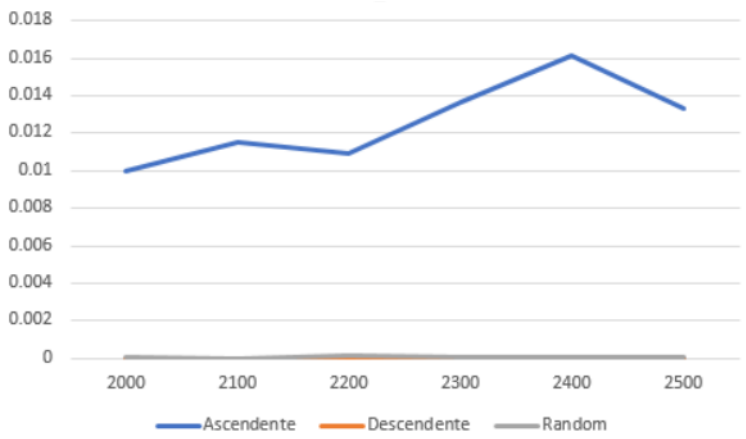
Result[]	0	1	1	2	4	5	7	7	8	9	10	11

Count[input[i]] will tell you the index position of input[i] in Result[]

Codigo C++:

```cpp
countingSort( std::vector<int>& numbers )
{
  int max = *max_element( numbers.begin(), numbers.end() );
  int min = *min_element( numbers.begin(), numbers.end() );
  int range = max - min + 1;

  std::vector<int> count( range ), output( numbers.size() );

  for ( int i = 0; i < numbers.size(); i++ )
  {
    count[numbers[i] - min]++;
  }

  for ( int i = 1; i < count.size(); i++ )
  {
    count[i] += count[i - 1];
  }

  for ( int i = numbers.size() - 1; i >= 0; i-- )
  {
    output[count[numbers[i] - min] - 1] = numbers[i];
    count[numbers[i] - min]--;
  }

  for ( int i = 0; i < numbers.size(); i++ )
  {
    numbers[i] = output[i];
  }

  return numbers;
}
```

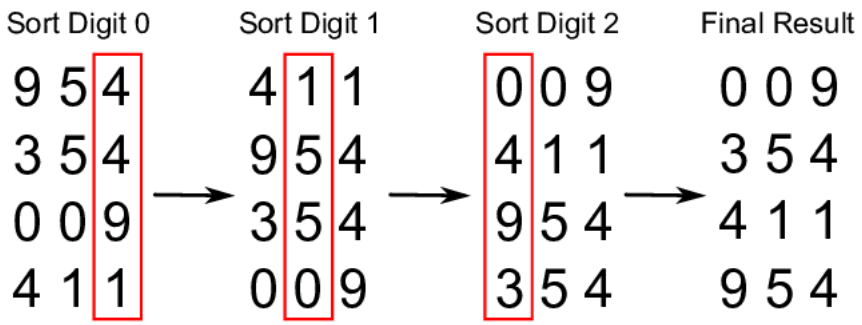
BenchMark:



2) **Radixsort**

El ordenamiento Radix es un algoritmo de ordenamiento que ordena enteros procesando sus dígitos de forma individual. Como los enteros pueden representar cadenas de caracteres (por ejemplo, nombres o fechas) y, especialmente, números en punto flotante especialmente formateados, radix sort no está limitado sólo a los enteros.

Ejemplo:

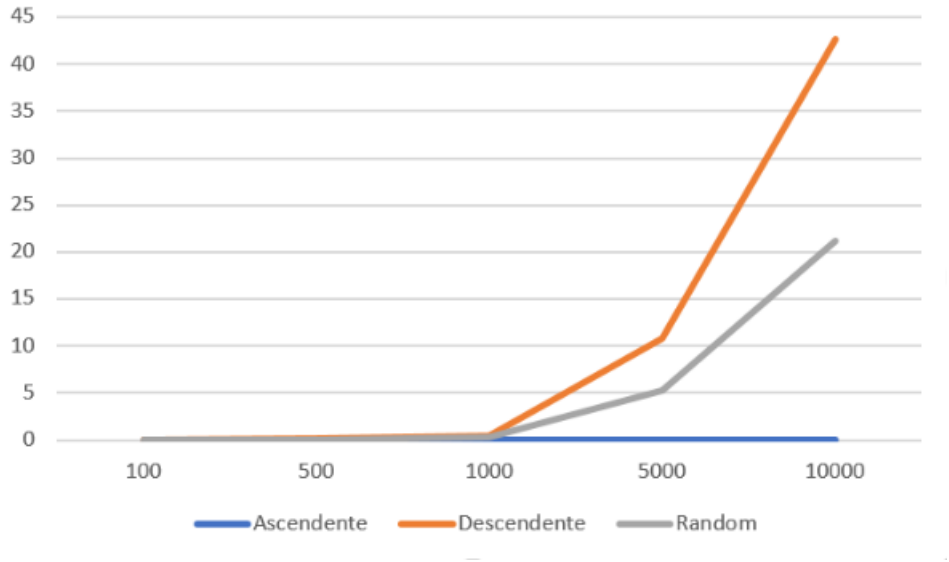


Codigo C++:

```
radixSort( std::vector<int>& numbers )
{
  int m = *max_element( numbers.begin(), numbers.end() );

  for ( int exp = 1; m / exp > 0; exp *= 10 )
  {
    countSort( numbers, exp );
  }

  return numbers;
}

countSort( std::vector<int>& numbers, int exp )
{
  int n = numbers.size();
  std::vector<int> output;
  output.resize( n );

  int i, count[10] = { 0 };

  for ( i = 0; i < n; i++ )
  {
    count[( numbers[i] / exp ) % 10]++;
  }

  for ( i = 1; i < 10; i++ )
  {
    count[i] += count[i - 1];
  }

  for ( i = n - 1; i >= 0; i-- )
  {
```

```
    output[count[( numbers[i] / exp ) % 10] - 1] = numbers[i];
    count[( numbers[i] / exp ) % 10]--;
  }

  for ( i = 0; i < n; i++ )
  {
    numbers[i] = output[i];
  }

  return numbers;
}
```

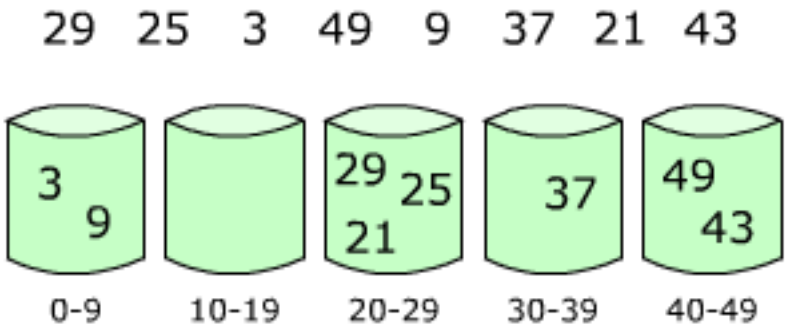
BenchMark:



3) **BucketSort**

es un algoritmo de ordenamiento que distribuye todos los elementos a ordenar entre un número finito de casilleros. Cada casillero sólo puede contener los elementos que cumplan unas determinadas condiciones.

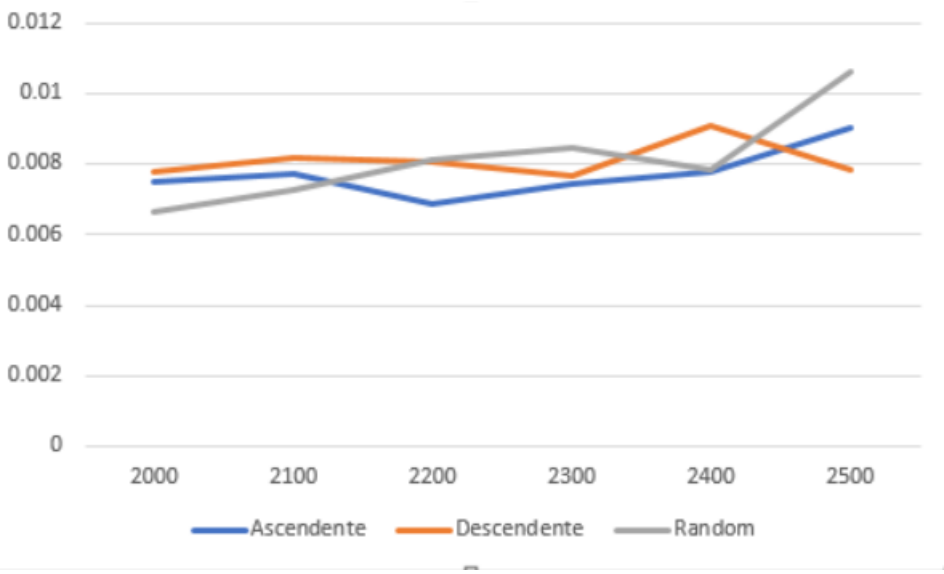
Ejemplo:



Codigo C++:

```
bucketSort( std::vector<int>& numbers )
{

  int n = numbers.size();
  std::vector<std::vector<float>> buckets;
  buckets.resize( n );

  for ( int i = 0; i < n; i++ )
  {
    int bi = n * numbers[i];
    buckets[bi].push_back( numbers[i] );
  }

  for ( int i = 0; i < n; i++ )
  {
    std::sort( buckets[i].begin(), buckets[i].end() );
  }

  int index = 0;
  for ( int i = 0; i < n; i++ )
  {
    for ( int j = 0; j < buckets[i].size(); j++ )
    {
      numbers[index++] = buckets[i][j];
    }
  }

  return numbers;
}
```

BenchMark:


0.012
0.01
0.008
0.006
0.004
0.002
0
2000
2100
2200
2300
2400
2500
Ascendente
Descendente
Random