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

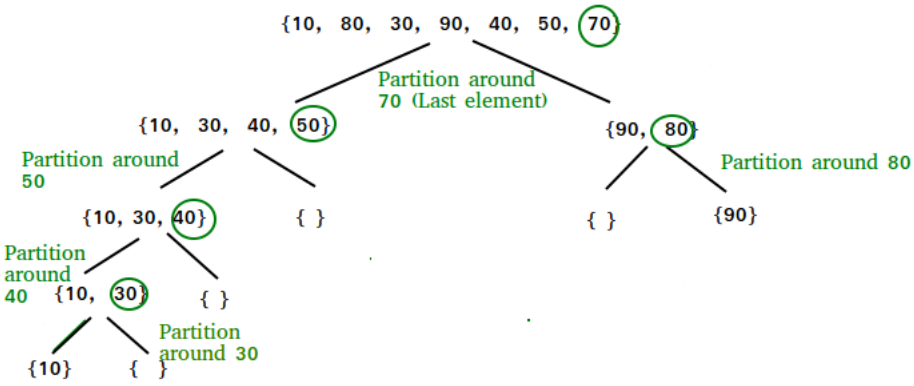
I. Introducción

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) **Quicksort**

Quicksort es un algoritmo de ordenación considerado entre los más rápidos y eficientes. El algoritmo usa la técnica divide y vencerás que básicamente se basa en dividir un problema en subproblemas y luego juntar las respuestas de estos subproblemas para obtener la solución al problema central.

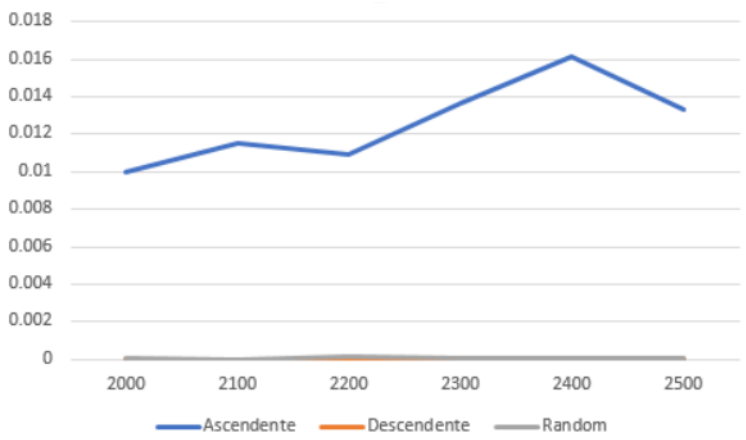
Ejemplo:



Codigo C++:

```cpp

partition( std::vector<int>& numbers, int start, int end )
{
  unsigned int pivot = numbers[start];
  unsigned int left = start + 1;
  unsigned int right = end;
  unsigned int tmp;

  while ( left != right )
  {
    if ( numbers[left] <= pivot ) left++;
    else
    {
      while ( ( left != right ) && ( pivot < numbers[right] ) ) right--;
      tmp = numbers[right];
      numbers[right] = numbers[left];
      numbers[left] = tmp;
    }
  }

  if ( numbers[left] > pivot ) left--;
  numbers[start] = numbers[left];
  numbers[left] = pivot;

  return ( left );
}


quickSort( std::vector<int>& numbers, int left, int right )
{
  if ( left < right )
  {

    int pi = partition( numbers, left, right );

    quickSort( numbers, left, pi - 1 );
    quickSort( numbers, pi + 1, right );
  }

  return numbers;
}
```

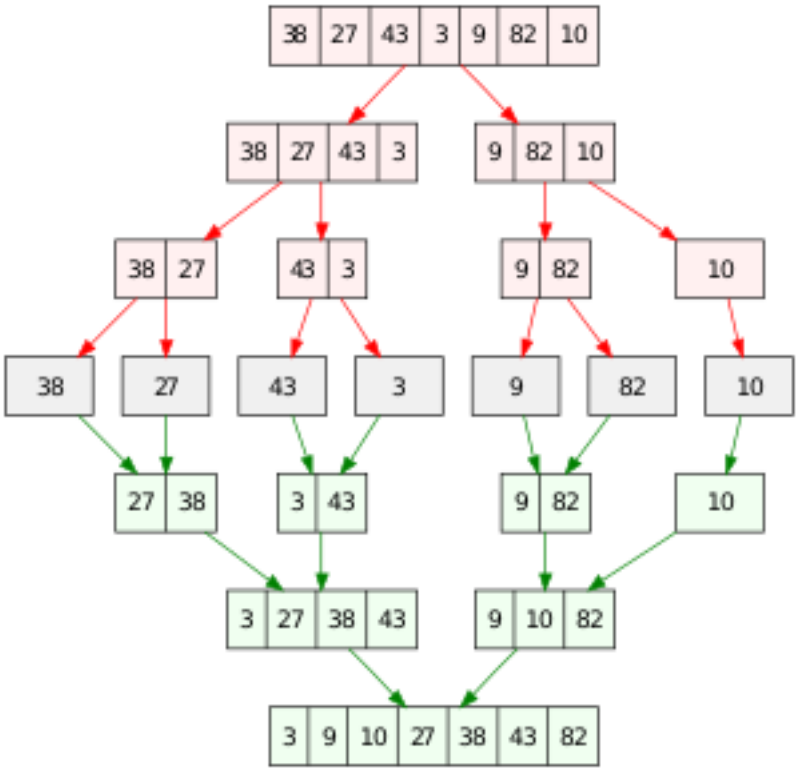
BenchMark:



2) **Mergesort**

Mergesort usa la técnica divide y vencerás. Divide el vector en dos mitades, se llama a sí mismo para las dos mitades y luego fusiona las dos mitades clasificadas.

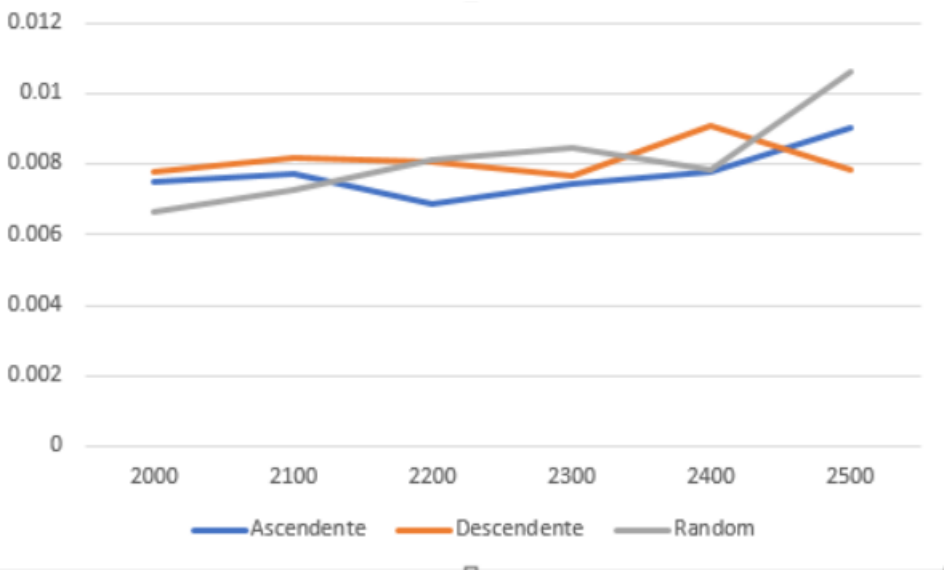
Ejemplo:



Codigo C++:

```

mergeSort( std::vector<int>& numbers, int left, int right )
{
  if ( left < right )
  {
    int mid = left + ( right - left ) / 2;

    mergeSort( numbers, left, mid );
    mergeSort( numbers, mid+ 1, right );

    merge( numbers, left, mid, right );
  }
  return numbers;
}

merge( std::vector<int>& numbers, int left, int mid, int right )
{
  int i, j, k;
  int n1 = mid - left + 1;
  int n2 = right - mid;

  std::vector<int> Left, Right;

  for ( i = 0; i < n1; i++ )
    Left.push_back( numbers[left + i] );
  for ( j = 0; j < n2; j++ )
    Right.push_back( numbers[mid + 1 + j] );

  i = 0;
  j = 0;
  k = left;

  while ( i < n1 && j < n2 )
  {
    if ( Left[i] <= Right[j] )
    {
      numbers[k] = Left[i];
      i++;
    }
    else
    {
      numbers[k] = Right[j];
      j++;
    }
    k++;
  }

  while ( i < n1 )
  {
    numbers[k] = Left[i];
    i++;
    k++;
  }

  while ( j < n2 )
  {
    numbers[k] = Right[j];
    j++;
    k++;
  }

}
```

BenchMark:


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