SSL 2024
Trabajo Práctico N°2

Lenguajes:

- C#
- Java

Integrantes:

- Ramiro Balbo
- Christian Zelaya
- Nicolas Pelullo
- Maximiliano Argañaraz

C#

Su historia comienza a finales de la década de 1990 cuando Microsoft buscaba crear un lenguaje moderno y versátil que le pueda competir con otras tecnologías de programación. C# se diseñó con el objetivo de combinar la potencia de lenguajes como C++ con la simplicidad de lenguajes como Java. Ademas buscaban crear un lenguaje moderno que pudiera integrarse con su nuevo .NET Framework, ofreciendo de esa forma una plataforma unificada para el desarrollo de aplicaciones. Su primer versión fue lanzada en el año 2000 en la cual se podia ver un lenguaje de programacion simple, moderno y orientado a objetos. A lo largo de los años fueron saliendo distintas evoluciones de este lenguaje junto a los .NET Framework 2.0 , 3.0 , 4.0 y 5.0

Ejemplo SorT

```
using System;
using System.Diagnostics;

0 references
class Main
{
    0 references
    static void main()
    {
        int size = 1000000;
        int[] numbers = new int[size];

        Random random = new Random();
        for (int i = 0; i < size; i++)
        {
            numbers[i] = random.Next(0, 1000000);
        }

        Stopwatch stopwatch = Stopwatch.StartNew();

        Array.Sort(numbers);

        stopwatch.Stop();
        long totalTime = stopwatch.ElapsedMilliseconds;

        Console.WriteLine("El tiempo de ordenamiento fue: " + totalTime + " milisegundos");
    }
}
```

Java

Java fue desarrollado en sus inicios por James Gosling, en el año 1991. Inicialmente Java era conocido como Oak o Green.
La primera versión del lenguaje Java fue publicada por Sun Microsystems en 1995. Y es en la versión del lenguaje JDK 1.0.2, cuando pasa a llamarse Java, corría el año 1996. Rápidamente Java ganó popularidad debido a su portabilidad y seguridad por lo cual se comenzó a utilizar en servidores, aplicaciones web, y posteriormente, en dispositivos móviles.

Este lenguaje es usado para motores de procesamiento de datos que puedan trabajar con conjuntos de datos complejos y cantidades masivas de datos en tiempo real.

Ejemplo SORT

```java
import java.util.Arrays;
import java.util.Random;

public class Main {
    Run | Debug
    public static void main(String[] args) {
        int size = 1000000;
        int[] numbers = new int[size];

        Random random = new Random();
        for (int i = 0; i < size; i++) {
            numbers[i] = random.nextInt(1000000);
        }

        long startTime = System.currentTimeMillis();

        Arrays.sort(numbers);

        long endTime = System.currentTimeMillis();

        long totalTime = endTime - startTime;

        // Imprimir el tiempo total
        System.out.println("El tiempo de ordenamiento fue: " + totalTime + " milisegundos");
    }

}
```

Comparación

Comparación Teórica

En los dos ejemplos estamos utilizando una secuencia de números enteros aleatoria, la cual después será ordenada con Array.sort. Los dos lenguajes están orientados a objetos; sin embargo Java utiliza distintos algoritmos dependiendo si son datos primitivos u objetos, cosa que en C# no pasa (utiliza el mismo algoritmo para los dos casos). Java para datos primitivos usa una variante del Quicksort Dual–Pivot, y para arrays de objetos utiliza Timsort pero no le vamos a dar mucha relevancia ya que en los dos casos comparativos de código utilizaremos datos primitivos, más precisamente enteros.

Los dos algoritmos son complejos y avanzados, ya que son variantes mejoradas de sus versiones originales. Mientras que Java utiliza un QuickSort con doble pivote (lo que mejora el rendimiento en comparación del Quicksort convencional), C# utiliza IntroSort que combina dos algoritmos, Quicksort y Heapsort.

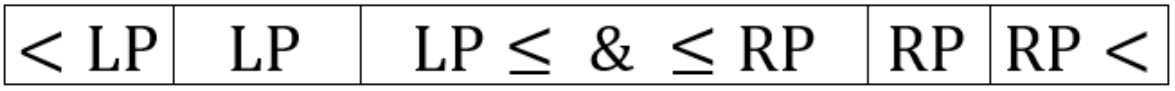
Entonces los dos algoritmos parten del QuickSort, el cual sea probablemente el algoritmo de ordenamiento convencional más eficiente, pero C# Utiliza una versión adaptativa del QuickSort. IntroSort puede ejecutar tanto QuickSort como HeapSort, cuando se detecta una profundidad de recursión elevada utiliza este último algoritmo mencionado.

Para concluir, C# evita el peor caso del QuickSort al cambiar a HeapSort asegurando un tiempo de ejecución de O(n log n) constante; mientras que Java reduce significativamente de recursión del QuickSort pero con una probabilidad menor de que se produzca una recursividad elevada $O(n^2)$.

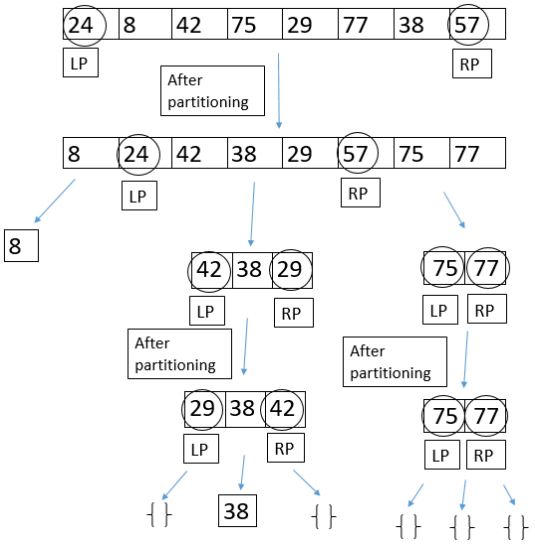
Composición Algorítmica

Como mencionamos anteriormente los dos lenguajes trabajan con variantes y adaptabilidades del QuickSort, si tuviera que dar una diferencia significativa la cual define el funcionamiento de cada algoritmo, diría que Java trabaja únicamente de manera recursiva (en arrays de datos primitivos); mientras que C# puede trabajar tanto recursivamente como no (al trabajar con QuickStart y tener una adaptabilidad de HeapSort para los casos con mayor recursividad).

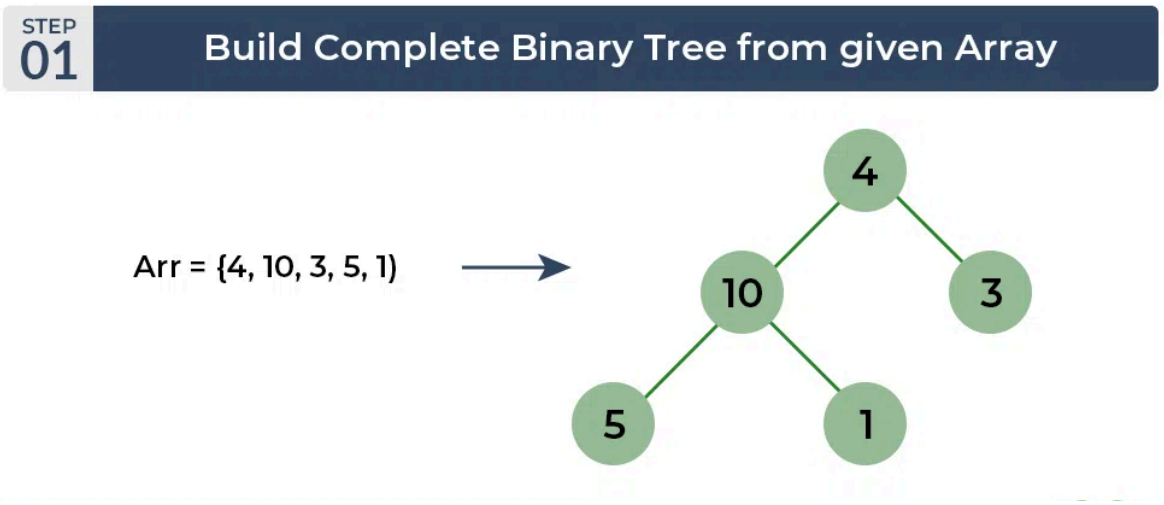
Dual-Pivot:

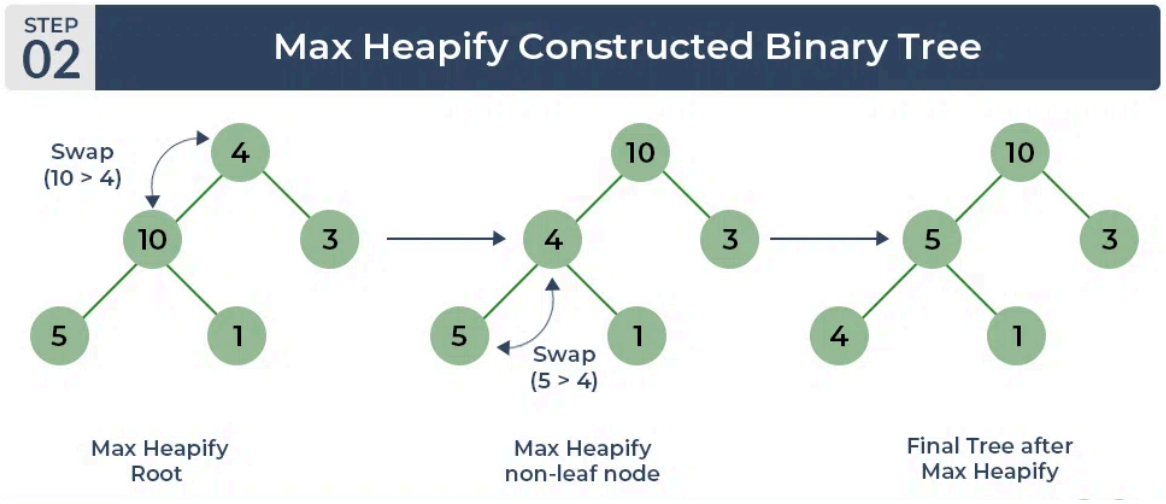


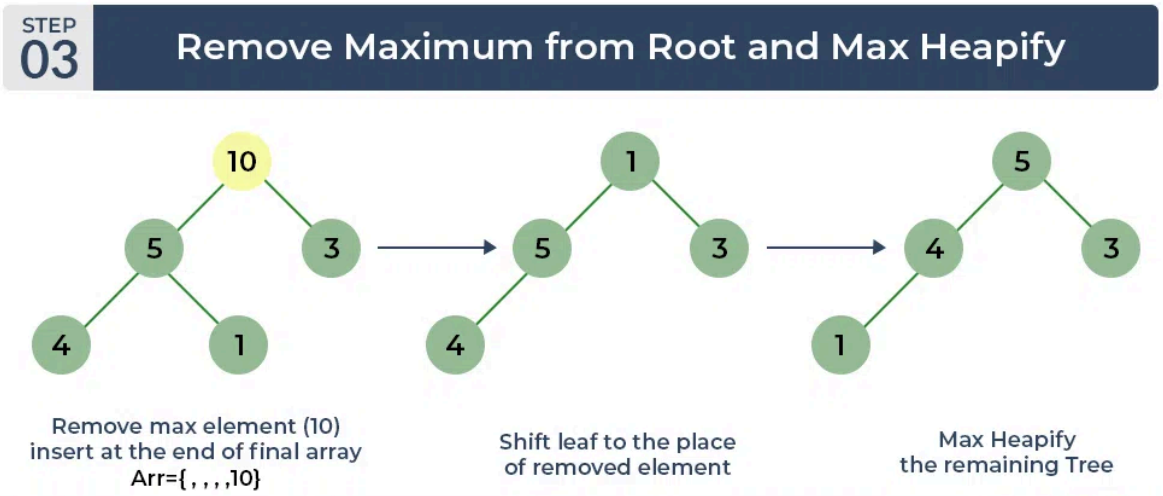
- Toma dos pivotes en los extremos del array, luego determina cual es mayor y cuál el menor; si o si el menor de los pivote tiene que estar en el extremo izquierdo del array, y el otro en el extremo derecho; en este punto contamos con tres divisiones.
- Ahora hay que leer el array original e ir ordenando los números, según las condiciones mostradas en la imagen. Se ordena usando swaps y punteros para redireccionar los valores.
- Este paso se repite recursivamente hasta que esté todo ordenado , es decir, el algoritmo va “dividiendo” los arrays en tres aplicando la misma lógica para cada subdivisión.



HeapSort:







- El HeapSort utiliza árboles binarios completos, básicamente transforma el array en un árbol y va leyendo las hojas con respecto a las raíces y compara cual es mayor. Si es mayor alguna de las hojas que la raíz hace un swap, cuando encuentra el máximo elemento lo elimina del árbol y lo pega en la lista; así sucesivamente con iteraciones y no recursividad.

Comparación Real (Testeo)

Para 1.000.000 valores enteros
Tiempos de Java: 103, 111, 143, 156, 114, 141, 142, 120, 129, 182, 108, 168
Tiempos de C#: 80, 81, 88, 82, 83, 90

Para 100.000 valores enteros
Tiempos de Java: 11, 35, 13, 14, 11, 35, 12, 12, 14
Tiempos de C#: 11, 12, 14, 15, 11, 11

Para 10.000 valores enteros
Tiempos de Java: 3, 3, 3, 2, 3, ,3, 4
Tiempos de C#: 0.

La diferencia es poco significativa pero se debe principalmente a la diferencia de que C# usa heapsort, Java utiliza doble pivote. Se puede apreciar que a los 100.000 la brecha es casi nula, en los demás hay casos hay una brecha proporcional; sin embargo C# es más constante.
Como dijimos antes ¿por qué pasa esto?, porque en C# cuando el nivel de recursividad es más alto, es decir los números están más desordenados, cambia a HeapSort. Esta brecha se puede ver a medida que crece la cantidad de datos porque el desorden de estos va a ser mayor.