1. QuickSort

El ordenamiento rápido(quicksort en ingles) es un algoritmo basado en comparaciones que utiliza la técnica divide y vencerás que permite ordenar N con costo $O(nlog(n))$. Es una de las técnicas más rápidas para ordenar elementos (basado en comparaciones). Fue desarollada por el científico brítanico en computacion Charles Antony Richard en 1960.

1.1. Descripción

El algortimo trabaja de la siguiente forma:

1. Elegir un elemento del arreglo a ordenar, que llamaremos pivote.
2. Resituar los demas elementos del arreglo a acada lado del pivote, de manera que al lado izquiero solo queden los menores a el pivote y al lado derecho solo queden lo mayores a el pivote.
3. El arreglo queda "dividido.[en] dos los menores al pivote y los mayores a el.
4. Repetir de forma recursiva para cada división hasta que el tamaño de las diviciones sea de 1 elemento.
5. luego de realixar todas las diviciones el arreglo queda ordenado.

1.2. Implementación

1.2.1. Pivote

El pivote sera el primer elemento del arreglo a ordenar

1.2.2. Sort

El método *Sort*() sera el encargado de dividir el arreglo de forma recursiva hasta que solo contengan un elemento.

```
private void Sort(int[] ordenar, int bot, int top) {
    int j = 0;
    if (bot < top) {
        j = particion(ordenar, bot, top);
        Sort(ordenar, bot, j - 1);
        Sort(ordenar, j + 1, top);
    }
}
```

Figura 1: Metodo Sort()

1.2.3. Particion

EL método *paricion*() es el encargado de mover los elementos del arreglo a los lados del pivote y retorna un entero que sera la pocición final del pivote donde se dividira el arreglo.

```
private int particion(int[] ordenar, int bot, int top) {
    int piv = ordenar[bot];
    int i = bot;
    int j = top + 1;
    while (true) {
        // encuentra mayor
        while (ordenar[++i] <= piv) {
            if (i == top) {
                break;
            }
        }
        // encuantra menor
        while (ordenar[--j] >= piv) {
            if (j == bot) {
                break;
            }
        }
        if (i >= j) {
            break;
        }
        Swap(ordenar, i, j);
    }
    Swap(ordenar, bot, j);

    return j;
}
```

Figura 2: Metodo particion()

2. MergeSort

El ordenamiento por mescla (mergesort en ingles) es un algortimo basado en comparaciones que al igual que en el Quicksort utiliza la tecnica de divir y vencerás, este algortimo permite ordenar N elementos de un arreglo con costo $O(nLog(n))$ ademas de ser estable.

2.1. Descripción

El algortimo trabaja de la siguiente forma:

1. Si el arreglo es de longitud 0 o 1 entonces ya se encuentra oredenado paso 3. En caulquier otro caso paso 2.
2. Divivir el arreglo en dos mitades de aproximadamente la mitad del tamaño, con cada arreglo repetir paso 1.
3. Reordenar cada sub-array de forma recursiva aplicandoe el ordenamiento por mescla.
4. Mesclar los dos sub-array resultantes para generar un rreglo ordenado.

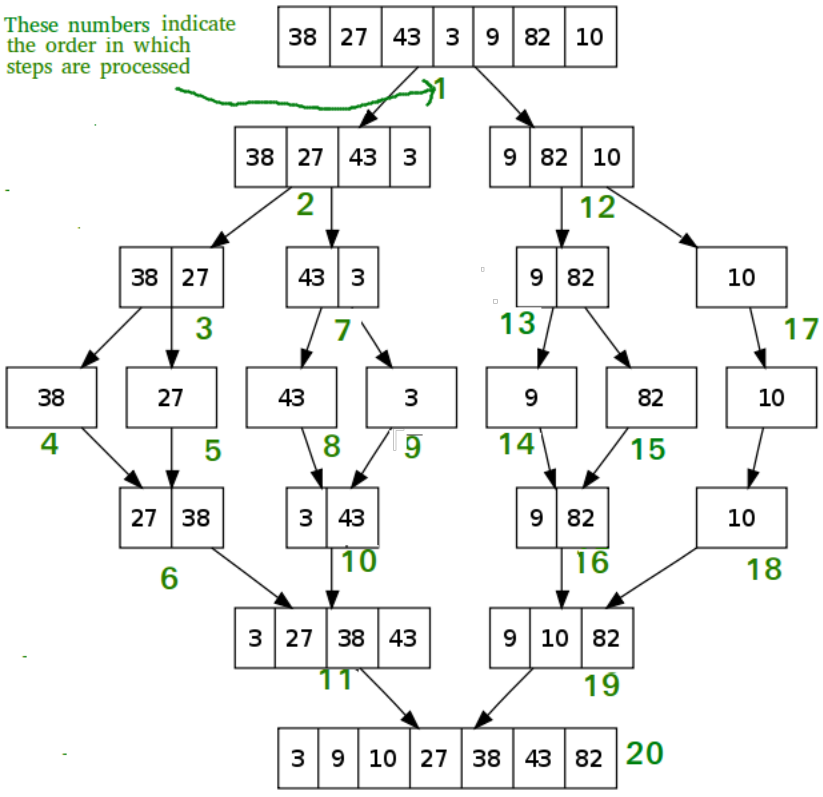

Figura 3: MergeSort

2.2. Implementación

2.2.1. Sort

Primero debemos crear un arreglo auxiliar que nos ayudara a ordenar el arreglo original.

```java
public void Sort(int[] ordenar) {
    int[] aux = new int[ordenar.length];
    Sort(ordenar, aux, 0, ordenar.length);
}
```

Figura 4: Sort

El segundo método Sort realizara las diviciones de forma recursiva hasta tener los arreglos de tamaño 1.

```java
private void Sort(int[] ordenar, int[] aux, int bot, int top) {
    // caso base
    if (top - bot > 1) {
        int mid = bot + (top - bot) / 2;
        Sort(ordenar, aux, bot, mid);
        Sort(ordenar, aux, mid, top);
        Merge(ordenar, aux, bot, mid, top);
    }
}
```

Figura 5: Sort

2.2.2. Merge

Método que ordenara los elementos de los sub arreglos mediante comparaciones en el arreglo auxiliar, luego de tener el arreglo auxiliar ordenado, se pasan los elementos a el arreglo original.

```java
private void Merge(int[] ordenar, int[] aux, int bot, int mid, int top) {
    int i = bot;
    int j = mid;

    for (int k = bot; k < top; k++) {
        if (i == mid) {
            aux[k] = ordenar[j++];
        } else if (j == top) {
            aux[k] = ordenar[i++];
        } else if (ordenar[j] < ordenar[i]) {
            aux[k] = ordenar[j++];
        } else {
            aux[k] = ordenar[i++];
        }
    }

    for (int k = bot; k < top; k++) {
        ordenar[k] = aux[k];
    }
}
```

Figura 6: Merge