Universidad ICESI

Sean Quintero - Bryan Guapacha

Paola Osorio - John Kennedy

Análisis de complejidad espacial de algoritmos de ordenamiento.

Insertion sort:

```
public static void insertionSortImperative(int[] input) {
        for (int i = 1; i < input.length; i++) {
        int key = input[i];
        int j = i - 1;
        while (j >= 0 && input[j] > key) {
                input[j + 1] = input[j];
                j = j - 1;
        }
        input[j + 1] = key;
    }
        }
```

Tipo	**Variable**	**Cantidad de valores atómicos**
Entrada	input	n
Auxiliar	key	1
	i	1

	j	1
Salida		

Sea n = input

Complejidad Espacial Total = Entrada + Auxiliar + Salida = n + 1 + 1 + 1 = n + 3 = **θ(n)**

Complejidad Espacial Auxiliar = 1 + 1 + 1 = **θ(1)**

Complejidad Espacial Auxiliar + Salida = 1 + 1 + 1 = **θ(1)**

Radix sort:

```
private static int findMaximumNumberIn(int[] arr) {
        return Arrays.stream(arr).max().getAsInt();
```

Tipo	**Variable**	**Cantidad de valores atómicos**
Entrada	arr	n
Auxiliar		
Salida	return	1

Complejidad Espacial Total = Entrada + Auxiliar + Salida = n + 1 = **θ(n)**

Complejidad Espacial Auxiliar = 0 = **θ(0)**

Complejidad Espacial Auxiliar + Salida = 0 + 1 = **θ(1)**

```
private static int calculateNumberOfDigitsIn(int number) {
        return (int) Math.log10(number) + 1; // valid only if number > 0
```

Tipo	**Variable**	**Cantidad de valores atómicos**
Entrada	number	1
Auxiliar		
Salida	return	1

Complejidad Espacial Total = Entrada + Auxiliar + Salida = 1 +1= **θ(1)**

Complejidad Espacial Auxiliar = 0 = **θ(0)**

Complejidad Espacial Auxiliar + Salida = 0 + 1 = **θ(1)**

```
private static void applyCountingSortOn(int[] numbers, int placeValue) {
        int range = 10; // radix or the base

        int length = numbers.length;
```

```
int[] frequency = new int[range];
int[] sortedValues = new int[length];

for (int i = 0; i < length; i++) {
int digit = (numbers[i] / placeValue) % range;
frequency[digit]++;
}

for (int i = 1; i < range; i++) {
frequency[i] += frequency[i - 1];
}

for (int i = length - 1; i >= 0; i--) {
int digit = (numbers[i] / placeValue) % range;
sortedValues[frequency[digit] - 1] = numbers[i];
frequency[digit]--;
}

System.arraycopy(sortedValues, 0, numbers, 0, length);
}
```

Tipo	**Variable**	**Cantidad de valores atómicos**
Entrada	numbers	m

	placeValue	1
Auxiliar	range	1
	length	1
	frecuency	m
	sortedValues	m
	i	1
	digit	1
Salida		

Complejidad Espacial Total = Entrada + Auxiliar + Salida =m+ 1+ 1 + 1 +1+1+m+m= 3m + 5 = **θ(m)**

Complejidad Espacial Auxiliar = 1 + 1 + 1 + +1+m+m= **θ(m)**

Complejidad Espacial Auxiliar + Salida =1 + 1 + 1 + +1+m+m= **θ(m)**

```
public static void radixSort(int numbers[]) {
    int maximumNumber = findMaximumNumberIn(numbers);

    int numberOfDigits = calculateNumberOfDigitsIn(maximumNumber);
```

```
    int placeValue = 1;

    while (numberOfDigits-- > 0) {
      applyCountingSortOn(numbers, placeValue);
      placeValue *= 10;
    }
  }
```

Tipo	**Variable**	**Cantidad de valores atómicos**
Entrada	numbers	n
Auxiliar	maximunNumber	1
	numberOfDigits	1
	placeValue	1
Salida		

Complejidad Espacial Total = Entrada + Auxiliar + Salida = n + 1 + 1 + 1 = **θ(n)**

Complejidad Espacial Auxiliar = 1 + 1 + 1 = **θ(1)**

Complejidad Espacial Auxiliar + Salida = 1 + 1 + 1 + 0 = **θ(1)**