Sorting algorithms

Params of classification

The sorting algorithms can be classified in terms of time complexity, space complexity, stability, internal vs external sort, and recursive vs non-recursive approach.

Time complexity

Classify the sorting algorithm according to the relation between the numbers of the input and the units of time spent to perform the sorting. This classificarions uses the BigO notation.

Space complexity

Classify the sorting algorithm according to the relation between the numbers of the input and the units of memory used to perform the sorting. This classificarions uses the BigO notation.

Stability

Classify as stable when the sorting algorithm preserves the original order between items with tie ordering. E.g, when the sorting algorithm convers `[6, 9, 3, 6*]` into `[3, 6*, 6, 9]`, it is unstable because 6 and 6* changes are tied in the sorting, but they switch its positions in the sorted list. A stable algorithm converts it to `[3, 6, 6*, 9]`.

Internal vs External sort

Classify the sorting algorith between the memory usage: RAM or rigid disk. The first one can be used when all the memory usage fits in the RAM, but to sort a large volume of data, there are sorting algorithms that use external memory.

Resursive or non-recursive

Classify the sorting algorithm between recursive or not.

Selection sort

```java
static int[] sort(int[] list) {
    for (int i = 0; i < list.length; i++) {
        int min = Integer.MAX_VALUE;
        int index = -1;
        for (int j = i + 1; j < list.length; j++) {
            if (list[j] <= min) {
                min = list[j];
                index = j;
            }
        }
        list[index] = list[i];
        list[i] = min;
    }
    return list;
}
```

Time complexity

Considering that each line takes a time to be done, we have, per line:

$$l3 = a;\ l4 = a;\ l6 = a;\ l7 = a;\ l8 = a;\ l11 = a;\ l12 = a$$

But these lines are run different times. The lines $l3,\ l4,\ l11\ and\ l12$ run n times. The lines $l6,\ l7\ and\ l8$ can run together:

$$\begin{aligned} 3a(n-1) + 3a(n-2) + 3a(n-3) + \ldots + 3a(1) \\ = \frac{3an}{2}(n-1) \\ = \frac{3an^2}{2} - \frac{3an}{2} \end{aligned}$$

The entire function has this time spent formula:

$$\frac{3an^2}{2} - \frac{3an}{2} + 4an$$

Removing the constants and highlighting the part of the formula with more influence in our time expent, we will find n^2. So, in the bigO notation, our function has:

$$BigO(n^2)$$

Bubble sort

```
static int[] boubleSort(int[] list) {
    for (int j = 0; j < list.length - 1; j++) {
        boolean sorted = true;
        for (int i = 0; i < list.length - 1; i++) {
            if (list[i] > list[i + 1]) {
                int temp = list[i];
                list[i] = list[i + 1];
                list[i + 1] = temp;
                sorted = false;
            }
        }
        if (sorted) {
            break;
        }
    }
    return list;
}
```

Time complexity

$$
\begin{gathered}
BigO = 3a(n-1) + (n-1)(B) \\
B = 5(n-1), \\
BigO = 3a(n-1) + (n-1)(5a(n-1)) \\
= 3an - 3a + (n-1)(5an - 5a) \\
= 3an - 3a + 5an^2 - 5an - 5an + 5a \\
= 5an^2 - 7an + 2a = n^2 - n \;=\; n^2 \\
BigO(n^2)
\end{gathered}
$$

Insertion Sort

```
static int[] insertionSort(int[] list) {
    for (int i = 1; i < list.length; i++) {
        int value = list[i];
        for (int j = i - 1; j >= 0; j--) {
            if (value < list[j]) {
                list[j + 1] = list[j];
                list[j] = value;
            } else {
                break;
            }
        }
    }
    return list;
}
```

Time complexity

$$
\begin{gathered}
BigO = (n-1)a + B \\
B = 3a(1) + 3a(2) + \ldots + 3a(n-2) \\
B = 3a\frac{n-2}{2}(1 + n - 2) \\
B = \frac{(3an - 6a)(n-1)}{2} \\
B = \frac{(3an^2 - 3an - 6an + 6a)}{2} \\
B = \frac{3an^2 - 9an + 6a}{2} \\
BigO = (n-1)a + \frac{3an^2 - 9an + 6a}{2} \\
BigO = na - a + \frac{3an^2 - 9an + 6a}{2}
\end{gathered}
$$

$$BigO = \frac{3a}{2}n^2 - \frac{9}{2a}n + 2a + na$$
$$BigO = n^2$$

Merge sort

Quick sort

Heap sort

Counting sort

Radiz sort