1 Sorting

Algorithm	Best	Average	Worst	Stable	Adaptive	In/out-of-place
Bubble	$O(n)$	$O(n^2)$	$O(n^2)$	Yes	Yes	In
Cocktail shaker	$O(n)$	$O(n^2)$	$O(n^2)$	Yes	Yes	In
Insertion	$O(n)$	$O(n^2)$	$O(n^2)$	Yes	Yes	In
Selection	$O(n^2)$	$O(n^2)$	$O(n^2)$	No	No	Out
Merge	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	Yes	No	In

Comparable: a.compareTo(b)

$a > b \rightarrow > 0$

$a < b \rightarrow < 0$

$a == b \rightarrow = 0$

Comparator: comparator.compare(a,b)

Strategies:

Iterative: bubble, cocktail shaker, selection, insertion (sort)

Divide and conquer: merge, LSD radix, in-place quicksort

Qualities:

- Stability: duplicates retain relative order
- Adaptive: algorithm can end early
- In-place: use $O(1)$ extra space or recursion. Out-of-place: use more extra space.

Algorithms

- Bubble sort:

```
// We can apply a ”no-swap” optimization so that it’s not necessary to keep track of the last swap index
outer loop: end to 1 (n)
    loop from 0 to $n - 1$ (i)
    compare arr[i] and arr[i + 1]
    if not in order: swap
```

- Cocktail shaker:

```
// If not applying ”no-swap”, take into account if indices are increasing or decrasing to mark the last swap index.
//That is, if they are decreasing, the last swap marker will be in the element with the smaller index, and vice-versa.
outer while loop
    bubble sort forward
    bubble sort backwards
```

- Insertion sort:

```
for (i = 1 to end)
    curr = arr[i]
    for ($j = i - 1$ to 0)
        if curr < arr[j]
            arr[j + 1] = arr[j]
        else
            arr[j + 1] = curr
```

- Selection:

```
for i = 0 to $n - 1$
    minIndex = i
    for $j = i + 1$ to n
        if arr[j] < arr[minIndex]
            minIndex = j
        swap arr[i], arr[minIndex]
```

- Selection:

```
// Divide array into left, right sub-arrays
mergesort(left)
mergesort(right)
merge left, right back recursively (with smaller element to the "left")
```