public static void bubbleSort(float[] arr) {  
 float tmp; //L1  
 for (int i = 0; i < arr.length - 1; i++) { //L2  
 boolean swapped = false; //L3  
 for (int j = 0; j < arr.length - i - 1; j++) { //L4  
 if (arr[j] > arr[j + 1]) { //L5  
 tmp = arr[j]; //L6  
 arr[j] = arr[j + 1]; //L7  
 arr[j + 1] = tmp; //L8  
 swapped = true; //L9  
 }  
 }  
 if(swapped == false) { //L10  
 return;  
 }  
 }  
}

\* Explaining x

With nesting loop as above, given an array length N = 5 -> 0 <= i < 4

|  |  |  |
| --- | --- | --- |
| i | Possible j < 4 - i | Run time |
| 0 | 0, 1, 2, 3 | 4 (N - 1) |
| 1 | 0, 1, 2 | 3 (N - 2) |
| 2 | 0, 1 | 2 (…) |
| 3 | 0 | 1 (1) |

* x = 1 + 2 + …. + (N – 2) + (N – 1) = (N – 1) (N – 1 + 1) = - N

Bubble sort

**Worst and average case:**

L1 = 0

L2:

i = 0 -> first assignment 🡪 run time = 1

0 <= i < arr.length - 1 🡪 run time = (N – 1) + 1 (count last comparison before terminating the loop)

i++ 🡪 run time = N – 1

L2 = 1 + N – 1 + 1 + N – 1 = 2N

L3 = N – 1

L4:

j = 0 🡪 run time = N - 1

j < arr.length – i – 1 🡪 run time = x + 1 (\*)

j++ 🡪 run time = x

L4 = N – 1 + x + 1 + x = 2x + N

L5 = L6 = L7 = L8 = L9 = x

L10 = N – 1

T(N) = 0 + 2N + N - 1 + 2x + N + 5x + N – 1 = 5N + 7x – 2 = + N – 2

* Time complexity with big-O-notation is O()

**Best case:**

L1 = 0

L2:

i = 0 🡪 run time = 1

i < arr.length – 1 🡪 run time = 1 (loop will break with i = 0)

i++ 🡪 run time = 0 (loop will break with i = 0)

L2 = 2

L3 = 1

L4 :

j = 0 🡪 run time = 1

j < arr.length – i – 1 🡪 run time = N – 1 + 1 (count last comparison before terminating the loop)

j++ 🡪 run time = N – 1

L4 = 2N

L5 = N – 1

L6 = L7 = L8 = L9 = 0 (array is sorted)

L10 = 1

T(N) = 2 + 1 + 2N + N – 1 + 1 = 3N + 3

* Time complexity with big-O-notation is O(N)

public static void insertionSort(float[] arr) {  
 for (int i = 1; i < arr.length; i++) { //L1  
 int j = i - 1; //L2  
 float key = arr[i]; //L3  
 while (j >= 0 && arr[j] > key) { //L4  
 arr[j+1] = arr[j]; //L5  
 j--; //L6  
 }   
 arr[j+1] = key; //L7  
 }  
}

\*\*Explaining y:

With nesting loop as above, given an array with length N = 5 -> 1 <= i < 5

|  |  |  |
| --- | --- | --- |
| i | Possible j (0 <= j <= 3) | Run time |
| 1 | 0 | 1 1 |
| 2 | 1, 0 | 2 (…) |
| 3 | 2, 1, 0 | 3 (N – 2) |
| 4 | 3, 2, 1, 0 | 4 (N – 1) |

* y = 1 + 2 + …. + (N – 2) + (N – 1) = (N – 1) (N – 1 + 1) = - N

Insertion sort

**Worst and average case:**

L1 = 1 + N – 1 + 1 + N – 1 = 2N

L2 = N – 1

L3 = N – 1

L4 = y + 1 + y = 2y + 1 (when j = -1, compare one more time before terminating while loop) (\*\*)

L5 = L6 = y

L7 = N – 1

T(N) = 2N + N – 1 + N – 1 + 2y + 1 + 2y + N – 1 = 5N + 4y – 2 = 2 + 3N - 2

* Time complexity with big-O-notation is O()

**Best case:**

L1 = 1 + N – 1 + 1 + N – 1 = 2N

L2 = N – 1

L3 = N – 1

L4 = 2(N – 1) = 2N – 2 (first comparison is always true, so second comparison always be executed)

L5 = L6 = 0 (array is sorted, so while loop will not be executed)

L7 = N – 1

T(N) = 2N + N – 1 + N – 1 + 2N – 2 + N – 1 = 7N – 5

* Time complexity with big-O-notation is O(N)

public static void selectionSort(float[] arr) {  
 float tmp; //L1  
 for (int i = 0; i < arr.length - 1; i++) { //L2  
 int min = i; //L3  
 for (int j = i + 1; j < arr.length; j++) { //L4  
 if (arr[j] < arr[min]) { //L5  
 min = j; //L6  
 }  
 }  
 if (min != i) { //L7  
 tmp = arr[min]; //L8  
 arr[min] = arr[i]; //L9  
 arr[i] = tmp; //L10  
 }  
 }  
}

(\*\*\*) Explaining z:

With nesting loop as above, given an array with length N = 5 -> 0 <= i < 4

|  |  |  |
| --- | --- | --- |
| i | Possible j (0 <= j <= 3) | Run time |
| 0 | 1, 2, 3, 4 | 4 (N – 1) |
| 1 | 2, 3, 4 | 3 (N – 2) |
| 2 | 3, 4 | 2 (…) |
| 3 | 4 | 1 1 |

* z = 1 + 2 + …. + (N – 2) + (N – 1) = (N – 1) (N – 1 + 1) = - N

Selection sort

**Worst and average case:**

L1 = 0

L2 = 1 + N – 1 + 1 + N – 1 = 2N

L3 = N – 1

L4 = N - 1 + z + 1 + z = 2z + N (\*\*\*)

L5 = L6 = z

L7 = L8 = L9 = L10 = N – 1

T(N) = 2N + N – 1 + 2z + 2 + 2z + 4N – 4 = 8N + 4z – 5 = 2 + 6N – 5

* Time complexity with big-O-notation is O()

**Best case:**

L1 = 0

L2 = 1 + N – 1 + 1 + N – 1 = 2N

L3 = N – 1

L4 = N - 1 + z + 1 + z = N + 2z

L5 = z

L6 = 0

L7 = N - 1

L8 = L9 = L10 = 0

T(N) = 2N + N – 1 + N + 2z + z + N – 1 = 5N + 3z – 2

* Time complexity with big-O-notation is O()