EX 1.1

1. $10n^2 + 100n + 1000 \Rightarrow O(n^2)$ *~lowest order*
2. $n^2 \log n \Rightarrow O(n^2 \log n)$
3. $10n^3 - 7 \Rightarrow O(n^3)$
4. $2^n + 100n^3 \Rightarrow O(2^n)$ *~highest order*

EX 1.3

```
private static int findLargest(int[] arr) {
    int largest = arr[0];
    for (int i = 1; i < arr.length-1; i++) {.
        if (arr[i] > largest)
            largest = arr[i];
    }
    return largest;
}
```

Time complexity: $O(n^2)$

EX 18.1

```
int[] arr = {3, 8, 12, 34, 54, 84, 91, 110};
int firstTarget = 54;
int secTarget = 45;
```

Linear Search:
A linear search algorithm will increment through the array and compare each index of the array's value against it's target until either a match is return or the end of the array is reached.

- *Searching for firstTarget:*

1. 3 != firstTarget, i++
2. 8 != firstTarget, i++
3. 12 != firstTarget, i++
4. 34 != firstTarget, i++
5. 54 == firstTarget, return arr[4]

- *Searching for secTarget:*

1. 3 != secTarget, i++
2. 8 != secTarget, i++
3. 12 != secTarget, i++
4. 34 != secTarget, i++
5. 54 != secTarget, i++
6. 84 != secTarget, i++
7. 91 != secTarget, i++
8. 110 != secTarget, i++
9. exit loop, return -1

   Binary Search:
   A binary search algorithm takes an ordered list and continuously divides it in half depending on if the value found is larger or smaller than the target until only the target (or the lack of one) remains.

- *Searching for firstTarget:*

1. 34 < firstTarget
2. 84 > firstTarget
3. 54 == firstTarget, return arr[4]

- *Searching for secTarget:*

1.
    1. 34 < firstTarget
2. 84 > firstTarget
3. 54 == firstTarget and is only remaining value, return -1

EX 18.4

```
list: 90, 8, 7, 56, 123, 235, 9, 1, 653
```

a) Selection Sort:

1. 1, 8, 7, 56, 123, 235, 9, 90, 653
2. 1, 7, 8, 56, 123, 235, 9, 90, 653
3. 1, 7, 8, 9, 123, 235, 56, 90, 653
4. 1, 7, 8, 9, 56, 235, 123, 90, 653
5. 1, 7, 8, 9, 56, 90, 123, 235, 653

b) Insertion Sort:

1. 8, 90, 7, 56, 123, 235, 9, 1, 653
2. 7, 8, 90, 56, 123, 235, 9, 1, 653
3. 7, 8, 56, 90, 123, 235, 9, 1, 653
4. 7, 8, 9, 56, 90, 123, 235, 1, 653
5. 1, 7, 8, 9, 56, 90, 123, 235, 653

c) Bubble Sort

1. 8, 90, 7, 56, 123, 235, 9, 1, 653
2. 8, 7, 90, 56, 123, 235, 9, 1, 653
3. 8, 7, 56, 90, 123, 235, 9, 1, 653
4. 8, 7, 56, 90, 123, 9, 235, 1, 653
5. 8, 7, 56, 90, 123, 9, 1, 235, 653
6. 7, 8, 56, 90, 123, 9, 1, 235, 653
7. 7, 8, 56, 90, 9, 123, 1, 235, 653
8. 7, 8, 56, 90, 9, 1, 123, 235, 653
9. 7, 8, 56, 9, 90, 1, 123, 235, 653
10. 7, 8, 56, 9, 1, 90, 123, 235, 653
11. 7, 8, 9, 56, 1, 90, 123, 235, 653
12. 7, 8, 9, 1, 56, 90, 123, 235, 653
13. 7, 8, 1, 9, 56, 90, 123, 235, 653
14. 7, 1, 8, 9, 56, 90, 123, 235, 653
15. 1, 7, 8, 9, 56, 90, 123, 235, 653