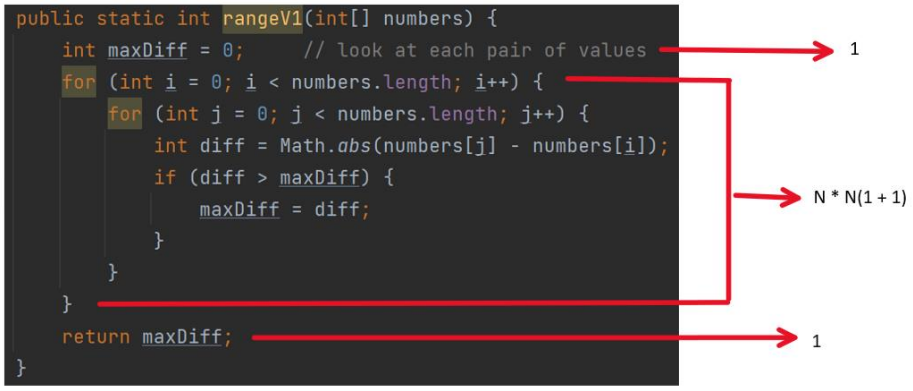
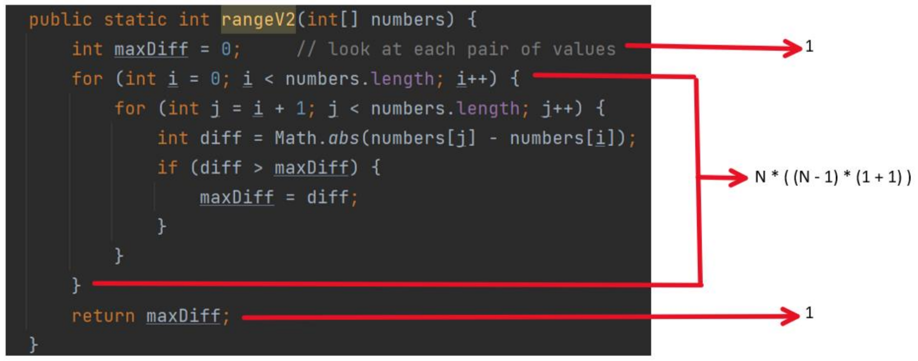
Runtime Analysis:



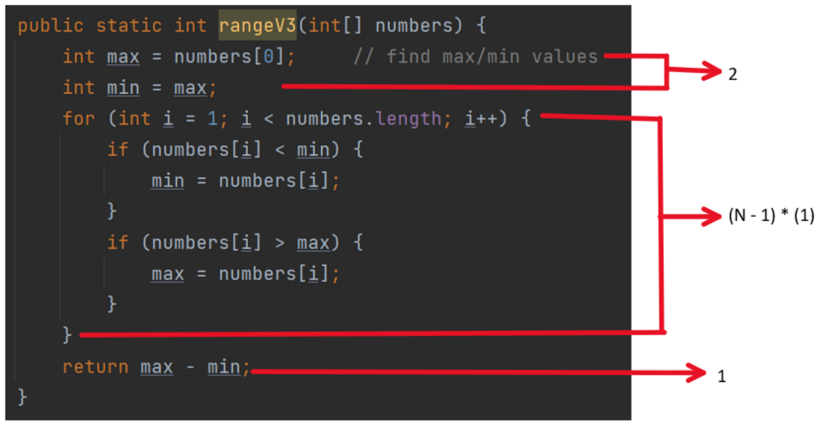
```java
public static int rangeV1(int[] numbers) {
    int maxDiff = 0;      // look at each pair of values
    for (int i = 0; i < numbers.length; i++) {
        for (int j = 0; j < numbers.length; j++) {
            int diff = Math.abs(numbers[j] - numbers[i]);
            if (diff > maxDiff) {
                maxDiff = diff;
            }
        }
    }
    return maxDiff;
}
```

The slowest of the range algorithms, rangeV1, runs $1 + N * N(1 + 1) + 1 = 2N^2 + 2$ statements. This means rangeV1 runs **O(N^2)**.



```java
public static int rangeV2(int[] numbers) {
    int maxDiff = 0;      // look at each pair of values
    for (int i = 0; i < numbers.length; i++) {
        for (int j = i + 1; j < numbers.length; j++) {
            int diff = Math.abs(numbers[j] - numbers[i]);
            if (diff > maxDiff) {
                maxDiff = diff;
            }
        }
    }
    return maxDiff;
}
```

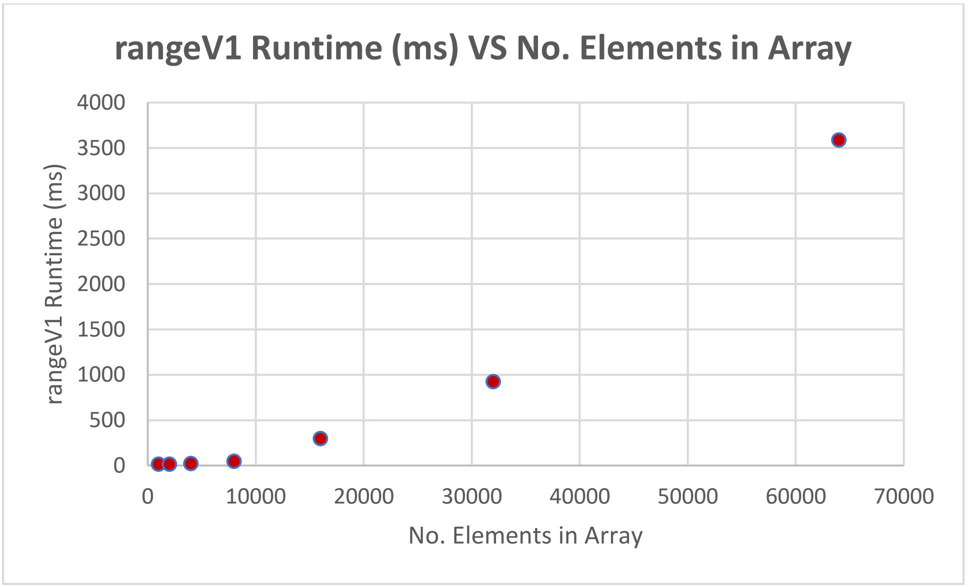
Method rangeV2 runs $1 + N * \big((N - 1) * (1 + 1)\big) + 1 = 2N^2 - 2N + 2$ statements. This means rangeV2 runs **O(N^2)**.



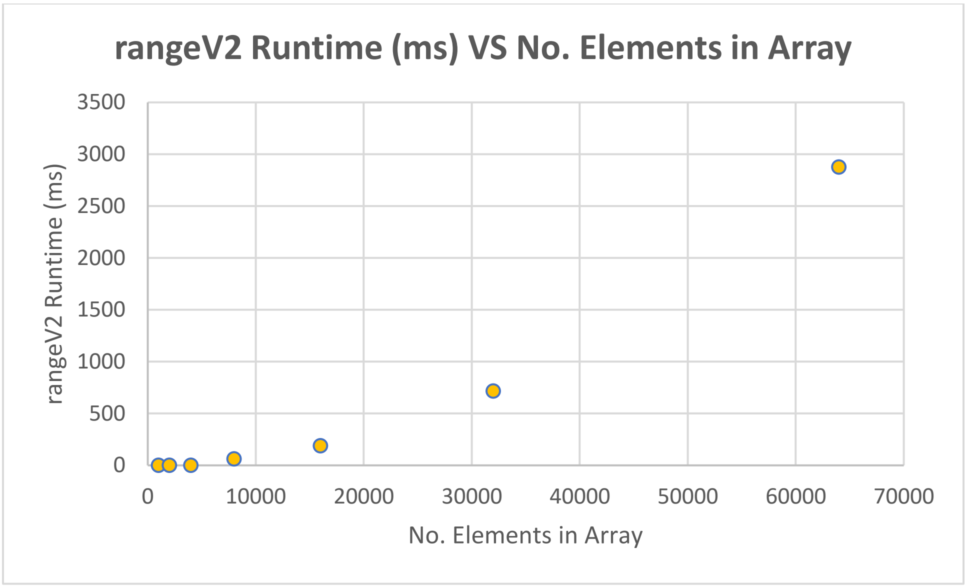
```java
public static int rangeV3(int[] numbers) {
    int max = numbers[0];     // find max/min values
    int min = max;
    for (int i = 1; i < numbers.length; i++) {
        if (numbers[i] < min) {
            min = numbers[i];
        }
        if (numbers[i] > max) {
            max = numbers[i];
        }
    }
    return max - min;
}
```

rangeV3 runs $2 + (N - 1) * (1) + 1 = 2 + N - 1 + 1 = 2 + N$ statements. This means rangeV3 runs **O(N)**, making it the most efficient of the three range algorithms.

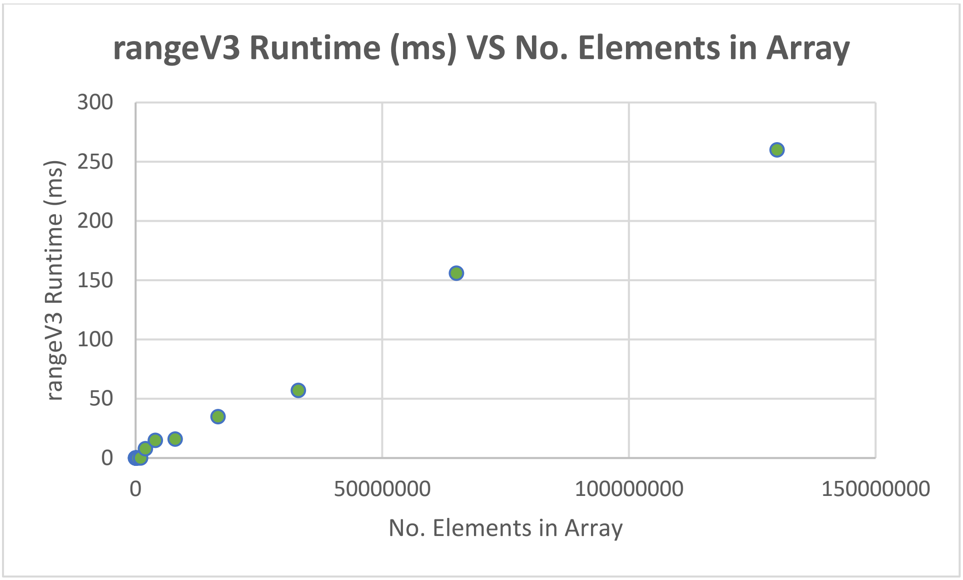
Runtime Graphs:



No. of Elements in Array	rangeV1 Runtime (ms)
1000	16
2000	16
4000	23
8000	47
16000	298
32000	924
64000	3588



No. of Elements in Array	rangeV2 Runtime (ms)
1000	0
2000	0
4000	0
8000	63
16000	189
32000	716
64000	2878



No. of Elements in Array	rangeV3 Runtime (ms)
1000	0
2000	0
4000	0
8000	0
16000	0
32000	0
64000	0
128000	0
256000	0
512000	0
1000000	0
2000000	8
4000000	15
8000000	16
16700000	35
33000000	57
65000000	156
130000000	260