Introduction to Algorithm Analysis

`reorder()` is a method that sorts two array elements.

```
void reorder(int[] array, int i, int j) {
    if (array[i] > array[j]) {
        int temp = array[i];
        array[i] = array[j];
        array[j] = temp;
    }
}
```

Figure 1

Question 1. Suppose an array `a` contains the values {`6, 11, 9, 13`}. List the contents of `a` after the method call `reorder(a, 1, 2)`.

6,9,11,13

Question 2. Suppose we define an **operation** as an *assignment statement*, *arithmetic operation*, or *comparison*. How many operations does the method execute when `reorder(a, 1, 2)` is called?

4 operations.

Question 3. How many operations does the method execute when `reorder(a, 0, 1)` is called?

1 operation.

Question 4. Suppose an array `b` contains the values {`2, 6, 13, 8, 3`}. How many operations does the method execute when `reorder(b, 3, 4)` is called?

4 operations.

Question 5. How many operations does the method execute when `reorder(b, 1, 2)` is called?

1 operation.

Question 6. Is there an upper bound (i.e. maximum amount) on the number of operations that `reorder()` can execute? Why or why not?

4 are executed if the branch is take, 1 if not. Therefore, 4 is the maximum.

Question 7. Does the number of operations the `reorder()` method executes depend on the size of its input (*i.e.*, the number of elements in the input)? Why or why not?

No. Regardless of the size of the array input, only two elements from the array are ever accessed.

Question 8. We say that the `reorder()` method executes in *constant* time. Another way to say this is that the method is $\mathcal{O}(1)$. Complete the following sentence:

A method is $\mathcal{O}(1)$ (or executes in constant time) if...

If the method has a maximum number of operations that it may execute, regardless of the size of its input.

Below is a Java method `normalize()` that maps values that are in the range $[min..max]$ to the range $[0..1]$:

```java
void normalize(double[] array, double min, double max) {
    for (int i = 0; i < array.length; i++) {
        array[i] = (array[i] - min) / (max - min);
    }
}
```

Figure 2

Question 9. Suppose an array `a` contains the values {`5, 15, 10`} and the method is called with the following method call:

```
normalize(a, 5, 15);
```

What are the contents of the array after this method call?

0, 1, 0.5

Question 10. How many operations does the method execute when `normalize(a, 5, 15)` is called?
Note: the initialization of the variable `i` executes before the first iteration of the loop. The iteration and comparison statements occur after each iteration of the loop.

19 : 1 initialization and 6 (comparison, iteration, loop body) times 3 iterations.

Question 11. Suppose the `normalize()` method is called with an array of length 20 as an argument. How many operations are executed by the method?

121 - 1 initialization and 6 operations times 20 iterations.

Question 12. Suppose the `normalize()` method is called with an array of length n as an argument. How many operations are executed by the method?

$6n + 1$

Question 13. We say that the `normalize()` method runs in *linear* time. Another way to say this is that the method is $\mathcal{O}(n)$. Complete the following sentence:

A method is $\mathcal{O}(n)$ (or executes in linear time) if...

If the number of operations the method executes is a linear function of the size of its input.

Question 14. We say that *quadratic* time methods are $\mathcal{O}(n^2)$. Complete the following sentence:

A method is $\mathcal{O}(n^2)$ (or executes in quadratic time) if...

If the number of operations the method executes is a quadratic function of the size of its input.

Label each of the following methods either $\mathcal{O}(1)$, $\mathcal{O}(n)$, or $\mathcal{O}(n^2)$.

```
int max(int a, int b) {
    if (a > b) {
        return a;
    } else {
        return b;
    }
}
```

Question 15. The `max()` method is $\mathcal{O}(\quad)$. Justify your answer.

The `max()` method is $\mathcal{O}(1)$ as it always executes 2 operations.

```
int maxElement(int[] array) {
    int max = array[0];

    for (int i = 0; i < array.length; i++) {
        if (array[i] > max) {
            max = array[i];
        }//end if
    } //end for

    return max;
}
```

Question 16. The `maxElement()` method is $\mathcal{O}(\quad)$. Justify your answer.

The `maxElement()` method is $\mathcal{O}(n)$ as the number of operations is a linear function of its input size.

```
int maxSubseqSum(int[ ] array) {
    int max = array[0];

    for (int i = 0; i < array.length; i++) {
        int sum = 0;
        for (int j = i; j < array.length; j++) {
            sum += j;

            if (sum > max) {
                max = sum;
            } //end if
        } //end for
    } //end for

    return max;
}
```

Question 17. The `maxSubseqSum()` method is $\mathcal{O}(\quad)$. Justify your answer.

The `maxSubseqSum()` method is $\mathcal{O}(n^2)$ as the number of operations is a quadratic function of its input.

Question 18. We are using the number of operations a method executes as a measure of its run time. In a few complete sentences, explain why we are using this measure of time rather than a wall-clock measure of time (*i.e.*, minutes, seconds, *etc.*).

We are using an abstract notion of time and instead are coming up with a consistent technique that analyzes an algorithm independent of any computer.

Question 19. Why is knowing that a method is $\mathcal{O}(n)$ more valuable than knowing that it takes fifteen seconds to execute on a 2.7GHz i7? In the space below, list the pros and cons for each statement.

- "The method is $\mathcal{O}(n)$."
  Pro: Assessment of run time doesn't rely on the computer.
  Con: Doesn't provide a wall-clock time prediction of run time.
- "The method took 15s on my i7."
  Pro: If it is run on another 2.7GHz i7, we can be reasonably certain it will take 15 seconds.
  Con: Not predictive of how long it will take to run on another computer.

Question 20. Is it possible that there are inputs for which a $\mathcal{O}(1)$ method executes more operations than a $\mathcal{O}(n)$ method that has the same specification (does the same thing)? Why or why not?

Yes. Suppose that the size of the input is small; it may be the case that the number of operations that the constant time method exe- cutes is more than the number of operations that the linear time method executes. As n grows, though, the linear time method will eventually exceed the constant time method (as the linear time methods operation count must grow linearly, while the constant time method has a maximum number of operations).Yes. Suppose that the size of the input is small; it may be the case that the number of operations that the constant time method exe- cutes is more than the number of operations that the linear time method executes. As n grows, though, the linear time method will eventually exceed the constant time method (as the linear time methods operation count must grow linearly, while the constant time method has a maximum number of operations).