COSC 3020
Algorithms

Name: Jacob Tuttle

Lab 03

September 25, 2019

1. Divide and Conquer Sum: Solutions to the divide an conquer sum problem are provided within the lab file `lab_03_tuttle.html`.

2. Runtime Analysis: Before we are able to present an accurate runtime estimation, or the recurrence relation for $T(n)$, we must first evaluate the `divideAndConquerSum()` method. Below I have issolated each key equation from the code in order to evaluate its running time.

```
if (!Array.isArray(a)) {
    a = [a];
}
```

(1)

```
if (a.length <= 2) {
    if (a.length == 2) {
        return a[0] + a[1];
    } else {
        return a[0]
    }
```

(2)

```
} else {
    var fThird = Math.floor(a.length / 3);
    var sThird = Math.floor(a.length / 3) * 2;

    return divideAndConquerSum(a.slice(0, fThird)) +
        divideAndConquerSum(a.slice(fThird, sThird)) +
        divideAndConquerSum(a.slice(sThird, a.length));
}
```
(3)

Moving through each of the code equations presented above, the time complexities of each can be analyzed in order to determine what the overal time complexity of the algorithm is.

1. A single boolean evaluation and an assignment statement will execute in constant time 1.
2. As the base case, this equation will evaluate immediately and just add a constant time of 1 to the recurence relation.
3. As the key portion of the recurence relation, this branch of code is what will mostly contribute to the time complexity; it's the section where the dividing and conquering part of the algorithm happens. On each execution, the two calculations for the bounds of the thirds will happen in constant time, adding 2 operations to our overall time complexity. Then the recurence part will happen when `divideAndConquerSum` is called with each third of the array. Ergo, the overall complexity for only the recursive part of this equation will be $3T(\frac{n}{3})$ and the overall branch time complexity for it will be $3T(\frac{n}{3}) + 2$

From here, we are able to write this recurence relation as a piecewise function $T(n)$ as:

$$T(n) = \begin{cases} 1 & n \leq 2 \\ 3T(\frac{n}{3}) + 2 & n > 2 \end{cases}$$

Solving by substitution, the system above can be generalized as:

$$\begin{aligned}
T(n) &= 3T(\frac{n}{3}) + 2 \\
&= 3(3T(\frac{\frac{n}{3}}{3}) + 2) + 2 \\
&= 9T(\frac{n}{9}) + 4 \\
&= 27T(\frac{n}{27}) + 6 \\
&= \vdots \\
&= 3^i T(\frac{n}{3^i}) + 2i
\end{aligned}$$

On an array of size n this recurence will need to execute at most $\log_3 n$ times before every branch would terminate at the base case. Thus, we are able to set $i = \log_3 n$. (Note: the $nT(1)$ below comes from the addition actions that take place in order to sum each element. This takes place in equation (2) above.)

$$T(n) = nT(1) + n\log_3 n + 2\log_3 n = n + (n+2)\log_3 n \in \Theta(n\log_3 n)$$

Following this solution by substitution, it has been shown that the function `divideAndConquerSum` is a member of $\Theta(n\log_3 n)$.