Marcus Anderson

Homework 2

CS 6515: Introduction to Graduate Algorithms

**My approach: *Binary Search Algorithm***

* First, we’ll find the middle index (*mid*) of the sorted input array (*A*)*,* by adding the lowest index (*low*) to the highest index (*high*) and dividing the sum by 2. *mid* = (*low* + *high*)/2, where the initial input values are: *low = 1* and *high = n – 1*
* Next, we’ll set both the number within the array at index *mid*, as well as the number directly to the left of it. These will be called *mid\_num­* and *left\_num* respectively.
* Afterwards we’ll check if *mid\_num* is equal to *left\_num*, if *mid\_num* is equal to *mid* + 1, or if neither condition is true.
* If *mid\_num* == *left\_num*, then that means *mid* equals the repeated number and we’ll return it accordingly.
* If *mid\_num* == *mid + 1*, then the repeated number is stored in the right side of the *mid* index, and we’ll recursively run the algorithm with *A*, *mid + 1*, and *high* as the inputs.
* If neither condition is *true*, then the repeated number is stored in the left side of the *mid* index, and we’ll recursively run the algorithm with *A*, *low*, and *mid - 1* as the inputs.

**Why this works:**

* Our input array, *A*, is sorted and already in ascending order. This helps our binary search determine if the repeated number is in the right side, left side, or midpoint of the input array.
* When *mid\_num* equals *left\_num*, this means we found the repeated number and return *mid*.
* When *mid\_num* equals *mid + 1*, we can assume there are more numbers on the right side of the array, meaning the repeated number is between *mid + 1* to *high*.
* When both conditions are *false*, we can assume there are more numbers on the left side of the array, meaning the repeated number is between *low* to *mid – 1*.

**Runtime: *O(log n)***

The binary search algorithm finds the repeated numbers within the sorted array by essentially diving the array into smaller subarrays at each run. The recurrence for this evaluates to T(*n*) = T(n/2) + O(1), where *a = 1, b = 2,* and *d = 0*. Using the master theorem, we get an overall runtime of O(log n), matching the typical runtime of the binary search algorithm.

**References:**

* [https://www.geeksforgeeks.org/binary-search/#](https://www.geeksforgeeks.org/binary-search/)

**Collaborators:**

* Lilley, Zachary J: zlilley3@gatech.edu
* Bertrand, James M: [jbertrand9@gatech.edu](mailto:jbertrand9@gatech.edu)
* Ramasamy, Veerajothi: vramasamy9@gatech.edu
* Acker, Joshua R: jacker7@gatech.edu
* Shah, Jeet Hemant: jshah328@gatech.edu