CMPT 435L ALGORITHMS
Assignment 2

November 1st 2024

Jonathan "Grant" Bever

Link to github repo

PART 1: Linear and Binary Search

The Linear Search algorithm is about as simple as a search algorithm gets, it works by looping through an array looking for an item, and comparing one by one until the search item is found.

As the name implies the asymptotic time is O(n) (linear). We can see this very clearly since the algorithm only consists of a single loop, furthermore, we can also observe that its execution time increases linearly with the size of the input array. My implementation is as follows.

```
export function linearSearch(arr: any[], searchItems: any[]): void{
    let totalComparisons = 0;

    for (const item of searchItems) {
        let comparisons = 0;

        for (let i = 0; i < arr.length; i++) {
            comparisons++;
            if (arr[i] === item) {
                break;
            }
        }
        totalComparisons += comparisons;
    }
    const avgComparisons = parseFloat((totalComparisons / searchItems.length).toFixed(2)
        );
    console.log(`Total Comparisons: ${totalComparisons}`);
    console.log(`Average Comparisons: ${avgComparisons}`);

}
```

When searching for all 42 random items through 666 magic items, it took on average 13755.1 comparisons for the whole search, with each individual search taking an average of 328.63 comparisons. The lowest total comparisons I recorded was 12556 while the highest was 15484.

As Linear Search is a simple search algorithm it is best suited for simple arrays with a small length, because of its linear nature it becomes much less viable when the array size increases.

Binary Search

The Binary Search algorithm is a much more efficient search algorithm, it works by starting with the middle value of the array, comparing the target value, then goes in the middle of whichever side the target was on (lower or higher) until the target value is reached.

Due to the divide and conquer methodology used in this algorithm its asymptotic running time is O($\log n$)

```
export function binarySearch(arr: string[], searchItems: string[]): void {
    let totalComparisons = 0;

    for (const item of searchItems) {
        let comparisons = 0;
        let left = 0;
        let right = arr.length - 1;

        while (left <= right) {
            comparisons++;
            const mid = Math.floor((left + right) / 2);

            if (arr[mid] === item) {
                break;
            } else if (arr[mid] < item) {
                left = mid + 1;
            } else {
                right = mid - 1;
            }
        }
        totalComparisons += comparisons;
    }
    const avgComparisons = parseFloat((totalComparisons / searchItems.length).toFixed(2)
        );
    console.log(`Total Comparisons: ${totalComparisons}`);
    console.log(`Average Comparisons: ${avgComparisons}`);
}
```

The Binary Search algorithm boasts an average of 356.6 comparisons to search for all 42 items out of the 666 magic items. The average for each individual search was 8.49. Which is much better than what we had for linear search.

While Binary Search is much more efficient than Linear Search it is important to note that in order for Binary Search to function the input array needs to be sorted.

Results

Search	**AVG Individual Comparisons**	**AVG Total Comparisons**
Linear	328.63	13755.1
Binary	8.49	356.6

As we can see from the table binary search is much more efficient than linear. So much so that there will be cases with our data set that once the linear search has found one item binary might already be done with all of them. With our data set of finding 42 items out of 666 items binary search is 38.6 times faster on average then linear.

While Binary is better in this situation it is important to note that binary is not always the best searching algorithm to use since it requires the array to be sorted and it's a bit harder to work with.

While these results are impressive they can be greatly improved by implementing the leetcode connoisseurs' best friend, hashing.

Part 2: Hashing