**Critical Thinking: Module 5**

Braden Heeney

Colorado State University Global

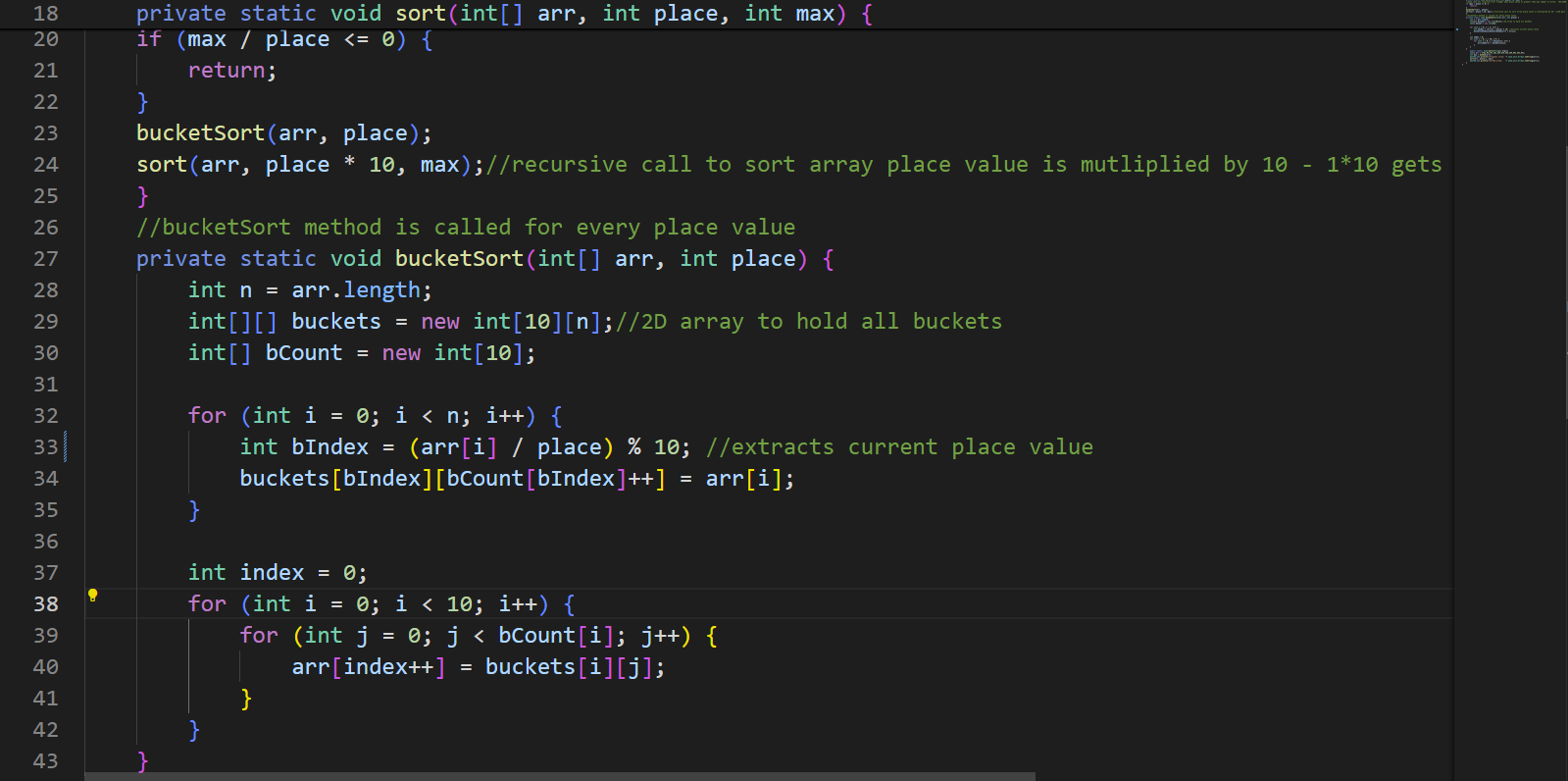
CSC400-1: Graphics & Visualization

Professor Herbert Pensado

1/19/2024

**Mod 5 Recap**

For Module 5, I went with the integer objects for my radix sort. Furthermore, I decided to use bucket sort for my algorithm. I tried a simpler method of utilizing a bubble sort, which worked for the first two iterations of my assignment, but when I got to the hundreds place, my order did not maintain. It was then when I discovered bucket sorting is a more reliable method of sorting due to it retaining proper order through multiple recursions. With that, I utilized a recursive loop to iterate through my array of numbers by place value. The bucket sort would loop through the ones, tens, and hundreds place and result in an ordered list using the radix approach required. the two main functions that made this take place are as follows.



To walk through my program, I first find the max number value in my string. In this case, it was 783. This means that I will have 3 place values to work through. I created a base case in my recursive loop so that when I reach a place value higher than what I have present in my array, I hit my default value and the recursion stops. Looking beyond my base case, I do my first call of bucketSort which accepts my array and the current place value we are working with. Jumping down to the bucketSort, I initialize a bucket for every number possible in a digit, 10 (0-9). I then use modulus division to find the ones digit value of every number in my array and sort the numbers into their perspective buckets. I then combine the buckets back together to get my organized array. I repeat this process for the tens and hundreds place to receive my organized array. Take my array arr = {783,99,472,182,264,543,356,295,692,491,94}. The first iteration through the list would give me [491, 472, 182, 692, 783, 543, 264, 94, 295, 356, 99]. The second time I get [543, 356, 264, 472, 182, 783, 491, 692, 94, 295, 99]. Lastly, I sort the hundreds place a receive this as my final array [94, 99, 182, 264, 295, 356, 472, 491, 543, 692, 783].

Firstly, the recursive loop is called O(d\*n) times. The letter d represents the number of digits in the max number, in this case 3. The letter n represents the number of elements in the array (11). This means the complexity is linear, **so the Big O notation would equal O(n)**.

Secondly, I needed to consider the bucketSort method and its’ big O notation. While there are two for loops, they are not nested so the complexity should be linear as well. This means that every time an iteration is occurring **I have a complexity of O(2n)**. Since in Big O notation we ignore constants, my Big O complexity for my entire program should be Linear, or **O(n)**.

References

WsCube Tech. (n.d.). Bucket sort: Algorithm, time complexity, code, more. WsCube Tech. Retrieved January 19, 2025, from <https://www.wscubetech.com/resources/dsa/bucket-sort>