Quiz 22: Searching SOLUTION

CSCI 110 Section 1

Monday, October 31, 2016

1) Write a function to determine whether a long exists in an array of long. Your function should run faster than iterating through all of the elements in the array. Oh and one more thing: the array of long is sorted. [70 points]

Return type? Boolean
Parameters? The array to search, and the number we're looking for

```
static boolean binarySearch(long[] nums, long toFind) {
	int low = 0;
	int high = nums.length - 1;
	// Write low <= high instead of low < high.
	//
	// If not, the function will sometimes incorrectly
	// return false if low and high are right next to each
	// other (in that case, mid will get the value of low
	// because of integer division).
	while (low <= high) {
		// Update mid
		int mid = (high + low) / 2;
		// then update either low or high
		if (nums[mid] == toFind) {
			// ...unless we're done
			return true;
		} else if (nums[mid] > toFind) {
			high = mid - 1;
		} else {
			low = mid + 1;
		}
	}
	return false;
}
```

Think about what happens if "toFind" is greater than all of the elements in "nums". Low will keep moving up and up until it's the same as high. On the next iteration, we check whether nums[mid] is the number we're looking for. Then we add one to low, and now "low <= high" is false.

A comment about the types: low, mid and high tell us where to search. They're indices. That's why they're all of type int. Try playing with the solution here: https://repl.it/EKPG/2

2) What's the maximum number of checks your solution does if given an array containing 10 elements? What about an array of 100 elements? [30 points]

By check, we mean "nums[mid] == toFind". You check all of the elements if you iterate through all of them. But with binary search, we only check a few.

For 10 elements, the worst case is 4 checks. One sequence is shown in the table below, looking for the element 3 in the 10-element array {0, 1, 2, 3, 4, 5, 6, 7, 8, 9} where the indices are the same as the values. The worst case being 4 makes sense because we halve the number of elements each time and 2^4 == 16 which is greater than 10.

low	**high**	**mid**
0	9	4
0	3	1
2	3	2
3	3	3

For 100 elements, the worst case is 7 checks (2^7 == 128 which is greater than 100). One sequence is below for the 100-element array where the indices are the same as the values {0, 1, 2, ..., 97, 98, 99}.

low	**high**	**mid**
0	99	49
50	99	74
75	99	87
75	86	80
81	86	83
84	86	85
84	84	84