CISC/CMPE-365
Fall 2019

Week 3 Lab Problem: Comparing Search Algorithms

Not to be handed in for grading

Consider the problem of searching a sorted array A of n values for a target value x. If x is in the array, the algorithm should return x's location in A. If x is not in the array, the algorithm should return -1.

Here is the well-known binary search algorithm:

```
Bin_Search(A, x):                    # A is indexed from 0 to n-1
                                     # Bin_Search returns the index of x in A, if found
                                     # Bin_Search returns -1 if x is not in A
    first =  0
    last = length(A) - 1

    while first <= last:
        mid = (first + last) / 2
        if A[mid] == x:
            return mid
        else if A[mid] > x:
            last = mid - 1
        else:
            first = mid + 1
    return -1
```

Here is a similar algorithm that divides the array into three parts and reduces the search to whichever third the target value would be in. We can call this trinary search:

```
Trin_Search(A,x):
        first = 0
        last = length(A) - 1
        while first <= last:
                t1 = first + (last – first)/3
                if A[t1] == x:
                        return t1
                else if A[t1] > x:              # reduce to the first third
                        last = t1 – 1
                else:
                        first = t1 + 1
                        if first > last:
                                return -1
                        mid = (first+last)/2
                        if A[mid] == x:
                                return mid
                        else if A[mid] > x:             # reduce to the middle third
                                last = mid-1
                        else:
                                first = mid+1           # reduce to the last third
        return -1
```

Since Trin_Search reduces the number of possible locations for x by a factor of 3 on each iteration, whereas Bin_Search reduces the number of possible locations by a factor of 2 on each iteration, it is plausible to hypothesize that Trin_Search should terminate faster.

Your task in this lab is to implement both algorithms and test them for efficiency.

Part 1:

Implement both algorithms in a language of your choice.

Test for correctness by letting A = [2, 4, 6, 8, 10] and searching for the x values

x = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11

Note that this set of search values explores every possible search path.

Part 2:

When we measure the efficiency of search algorithms, we only count the operations that involve comparing the target value x to an element of A. Each iteration of Bin_Search contains two such potential comparisons, while each iteration of Trin_Search contains four potential comparisons.

Modify your code so that instead of returning the location of x, the algorithms return the number of times x is compared to some element of A (this might be as small as 1 if x is in the first location checked, or as high as ... well, let's not give the game away!)

As in Part 1, test your code on a small array to make sure it is working correctly.

Part 3:

Compare the two algorithms on arrays of size 1000, 2000, 4000, 8000, 16000. In this experiment we are interested in the average-case behaviour of the algorithms. Use the test array from Part 1 to guide you in constructing tests that will explore all possible search paths.

For each size of array, record the average number of comparisons for searching the array with each of the two algorithms. Based on your observations, predict the average number of comparisons required for searching an array of 32000 elements.

How would you describe the growth rate of the average number of comparisons made by these two algorithms?

Reminder: This lab will NOT be graded.