


Binary vs Trinary

Robin Dawes

September 16, 2021

N this assignment you are tasked with comparing the binary search algorithm to a potentially superior version.

Binary Search

THE BINARY SEARCH algorithm for searching a sorted array is well known.

Here is a recursive version:

```
bin_search(A,first,last,target):  
    # returns index of target in A, if present  
    # returns -1 if target is not present in A  
    if first > last:  
        return -1  
    else:  
        mid = (first+last)//2  
        if A[mid] == target:  
            return mid  
        elif A[mid] > target:  
            return bin_search(A,first,mid-1,target)  
        else:  
            return bin_search(A,mid+1,last,target)
```

and here is an iterative version:

```
bin_search(A,target):
    # returns index of target in A, if present
    # returns -1 if target is not present in A
    done = False
    location = -1
    first = 0
    last = len(A)-1
    while not done:
        if first > last:
            done = True
        else:
            mid = (first+last)//2
            if A[mid] == target:
                location = mid
                done = True
            elif A[mid] > target:
                last = mid-1
            else:
                first = mid+1
    return location
```

Binary search reduces the number of possible locations for the target value by (about) half each time, which makes it quite efficient. But we could eliminate more locations by looking at two values in the range $A[first \dots last]$: If we look at the value $\frac{1}{3}$ of the way from *first* to *last*, and the value $\frac{2}{3}$ of the way from *first* to *last*, we can eliminate about $\frac{2}{3}$ of the locations each time. We can call this algorithm TRINARY SEARCH.



To search for value X, compare X to this and this

These two comparisons tell us which third of the array contains X

Here is a recursive version of the trinary search algorithm:

```

trin_search(A,first,last,target):
    # returns index of target in A, if present
    # returns -1 if target is not present in A
    if first > last:
        return -1
    else:
        one_third = first + (last-first)//3
        if A[one_third] == target:
            return one_third
        elif A[one_third] > target:
            # search the left-hand third
            return trin_search(A,first,one_third-1,target)
        else:
            two_thirds = first + 2*(last-first)//3
            if A[two_thirds] == target:
                return two_thirds
            elif A[two_thirds] > target:
                # search the middle third
                return trin_search(A,one_third+1,two_thirds-1,target)
            else:
                # search the right-hand third
                return trin_search(A,two_thirds+1,last,target)

```

and here is an iterative version:

```

trin_search(A,target):
    # returns index of target in A, if present
    # returns -1 if target is not present in A
    done = False
    location = -1
    first = 0
    last = len(A)-1
    while not done:
        if first > last:
            done = True
        else:
            one_third = first + (last-first)//3
            if A[one_third] == target:
                location = one_third
                done = True
            elif A[one_third] > target:
                # search the left-hand third
                last = one_third-1
            else:
                two_thirds = first + 2*(last-first)//3
                if A[two_thirds] == target:
                    location = two_thirds
                    done = True
                elif A[two_thirds] > target:
                    # search the middle third
                    first = one_third + 1
                    last = two_thirds - 1
                else:
                    # search the right-hand third
                    first = two_thirds + 1
    return location

```

Your assignment is to empirically evaluate the efficiency of these two search algorithms. As usual, in lieu of actual time measurements we will count operations. But to simplify the process we will only count the number of times a value in the array is compared to *target*.

Coding

IMPLEMENT BOTH binary search and trinary search in the language of your choice. You are not required to follow the pseudocode given above.

Not a completely free choice: you must choose one of C, C++, Java or Python.

Modify the algorithms so that they count the number of times values in the array are compared to *target*.

Experiment 1

For $n = 1000$, $n = 2000$, $n = 4000$, $n = 8000$, $n = 16000$ complete the following steps.

1. Generate an array (or list, in Python) of n integers in ascending order.
2. Use binary search to search the array for each of the values in the array. Record the average number of "comparison to *target*" operations required to conduct the binary searches.
3. Use trinary search to search the array for each of the values in the array. Record the average number of "comparison to *target*" operations required to conduct the trinary searches.

Filling the set with random numbers may make the experiment superficially seem more scientific, but in fact it is not necessary. You could simply fill the array with consecutive integers in order. However, in Experiment 2 you will want target numbers that fall between the numbers in the set ... so you may want to fill the array with consecutive even numbers.

Experiment 2

Repeat Experiment 1, but this time search only for values that are not present in the array. (One easy way to do this is to fill your array with even values, then search for odd values.) Search for one value that is too small, one value that is too large, and one value that falls between each pair of consecutive values in the array (a total of $n+1$ search values).

Presentation of Results

CREATE TABLES OR GRAPHS for the results of the two experiments.

A table might look like this (the numbers shown here are completely random):

Experiment 1		
n	Binary Search	Trinary Search
1000	7.5	6.189
2000	9.488	9.374
...

Based on the results of your experiments, answer the following questions:

1. Binary search and trinary search both fall into the $O(\log n)$ complexity class. Do your experiments show growth in the average number of comparisons that is consistent with this classification?
2. Compare the average number of counted comparisons for the two search algorithms, for different values of n . For what values of n (if any) does either of the search algorithms use $\leq 90\%$ as many comparisons as the other?
3. On the basis of your experiments, can you conclude that either binary search or trinary search is preferable over the other?

For an algorithm in the $O(\log n)$ class, doubling the size of the input should result in an approximately constant growth of the number of comparisons. For example, if the input size goes from 1000 to 2000 and then to 4000, the average number of comparisons might go from 8 to 9.5 to 10.9 ... increasing by about 1.5 each time n doubles.

For example, you might find that for $n = 4000$, binary search uses an average of 24 comparisons and trinary search uses an average of 36 comparisons. $24 \leq 0.9 * 36$

Logistics

YOU MAY COMPLETE the programming part of this assignment in Python, Java, C or C++.

You must submit your source code, properly formatted and documented according to standards established in CISC-121 and CISC-124. You must also submit a PDF file containing your answers to the questions. Both files must contain your name and student number, and must contain the following statement: "I confirm that this submission is my own work and is consistent with the Queen's regulations on Academic Integrity."

You are required to work individually on this assignment. You may discuss the problem in general terms with others and brainstorm ideas, but you may not share code. This includes letting others read your code or your written conclusions.

If you find code or other material on the internet that helps you understand how to complete this assignment and you want to use what you have learned, you must name the source as a reference. Also, you must make sure that what you submit is the result of your own work. In other words you cannot simply cut and paste from a source. My best advice is to study what you find, learn from it, then close the source and use what you have learned to build your own solution. If you do that, you will not be in danger of plagiarism.

The due date for this assignment is September 24, 2021, 11:59 PM

Submission will be through onQ.