Timothy Allec 1/12/2024

Name, SID, Date ....

In Class Assignment 2: Binary Search

Benjamin Sanders, MS November 25, 2020

1 Introduction

You will need to work individually to complete this assignment. Write your name at the top of all pages for this assignment. Turn in all work to Blackboard on or before the deadline to receive credit.

You may use additional libraries and online resources, if you get them approved in writing, over email, from the instructor first. If you have received approval from the instructor, write the approved libraries and any references in the space below.

no libraries used

....

....

....

....

2 Assignment Description

2.1 Big Picture

Binary search finds a value in a sorted array more quickly than considering every value in order.

2.2 Algorithm Implementation

Implement the following algorithm in Java, using the Vector data structure for any 1-D array, 2-D array, or linear algebra purposes.

BINARY-SEARCH(A, n, T)

$L = 0$
$R = n - 1$
while $L \leq R$
    $m = \lfloor (L + R)/2 \rfloor$
    if $A[m] < T$
        $L = m + 1$
    elseif $A[m] > T$
        $R = m - 1$
    else
        return m
return -1 // invalid, not found

```
public static int function(int[] array, int n, int T){
    int leftSide = 0;
    int rightSide = n -1;

    while(leftSide < rightSide ){
      int midPoint = (leftSide+rightSide)/2;
      if(array[midPoint] < T){
        leftSide = midPoint+1;
      }
      else if(array[midPoint]> T){
        rightSide = midPoint-1;
      }
      else{
        return midPoint;
      }
    }
    return -1;
}
```

Note that A is an array of values, of length n. Note that T is the value to search for.

2.3 Time Complexity Analysis

Where n is the number of data points in A, analyze the time complexity of the given algorithm with respect to n. Write the result of your analysis in big-O notation, i.e. $O(n^2)$ in the space below.

O(log(n))

....

2.4 Space Complexity Analysis

Where n is the number of data points in A, analyze the space complexity of the given algorithm with respect to n. Write the result of your analysis in big-O notation, i.e. $O(n \cdot log(n))$ in the space below.

O(n)

....

Name, SID, Date ....

2.5 Optimize the Algorithm for a Purpose

Choose an optimization, either in time or in space, for the given algorithm. Write your intended big-O notation, i.e. $O(1)$, $O(n)$, etc., in the space below, and write N/A in the other space.

- Time Complexity: create a ternary search where you split the array into 1/3s
- Space Complexity: ....

What application would benefit from the purpose of the above optimization? Why? Write two sentences to answer these questions in the space below.

An application where this type of search would be beneficial would be attempting to locate the vowels within the English alphabet. This is because the majority of vowels exist in the first third of the alphabet sequence, therefore, locating them would be more efficiently done with a ternary search rather than a binary search.

2.6 New Algorithm Design and Implementation

In the space below, design an algorithm that achieves the same purpose of the given algorithm, but includes the optimization you have specified above. Use pseudocode written in a style similar to the given algorithm, and implement it in Java. You may use as many additional pages as necessary for this purpose.

```
public static int function(int[] array, int n, int T) {
   int leftSide = 0;
   int rightSide = n - 1;
   while (leftSide <= rightSide) {
      int mid1 = leftSide + (rightSide - leftSide) / 3;
      int mid2 = rightSide - (rightSide - leftSide) / 3;
      if (array[mid1] == T) {
         return mid1;
      }
      if (array[mid2] == T) {
         return mid2;
      }
      if (array[mid1] > T) {
         rightSide = mid1 - 1;
      } else if (array[mid2] < T) {
         leftSide = mid2 + 1;
      } else {
         leftSide = mid1 + 1;
         rightSide = mid2 - 1;
      }
   }
   return -1;
}
```

PSEUDOCODE

```
function search(array, n, T):
   leftSide  0
   rightSide  n - 1

   while leftSide  rightSide:
      mid1  leftSide + (rightSide - leftSide) / 3
      mid2  rightSide - (rightSide - leftSide) / 3

      if array[mid1] = T:
         return mid1

      if array[mid2] = T:
         return mid2

      if array[mid1] > T:
         rightSide  mid1 - 1
      else if array[mid2] < T:
         leftSide  mid2 + 1
      else:
         leftSide  mid1 + 1
         rightSide  mid2 - 1

   return -1
```

Both my Binary Search and the Optimization (Ternary Search) work as intended, with no parts needing to be fixed or debugged.

3 What to Turn In

Turn in one PDF or Word document on Blackboard, containing the following items.

1. All pages scanned or photographed of the In Class Assignment completed document.
2. Any additional pages you used to complete the assignment.
3. All code created for the assignment, along with test cases.
4. One statement indicating which parts of your implementation(s) are working, and which parts are not.
5. Screenshots demonstrating the code working, if it is working.