Individual Project 2

CS627

Aidan Polivka

January 14, 2024

My team has been tasked with building a search algorithm to find a song in a sorted list of song titles. We've decided to explore an implementation of the binary search algorithm with a twist: instead of using the divide and conquer approach on 2 sub-arrays, we'll be splitting the original array into 3 sub-arrays. This effectively could be called a "Ternary Search" algorithm. This is the implementation we've built in C#:

```
/// <summary>
/// Ternary Searching Algorithm - Splits array into three segments <br />
/// Recursively calls TernarySearch on the segment the search term should be in
/// </summary>
/// <param name="arr">pre-sorted list of strings</param>
/// <param name="searchTerm">search term</param>
/// <param name="first">optional left boundary</param>
/// <param name="last">optional right boundary</param>
/// <returns>index of search term in array, or -1 if not found</returns>
private static int TernarySearch(
    string[] arr,
    string searchTerm,
    int first = 0,
    int? last = null)
{
    // Set operating variables
    last = last ?? arr.Length - 1;
    int left = first + ((int)(last - first) / 3);
    int right = first + ((int)Math.Ceiling((double)(last - first) * 2 / 3));

    // Recursion ending cases
    if (first == last && arr[first] != searchTerm)
    {
        return -1;
    }else if (arr[left] == searchTerm)
    {
        return left;
    } else if (arr[right] == searchTerm)
    {
        return right;
    }

    // Recursive cases
    if (string.Compare(searchTerm, arr[right]) > 0)
    {
        return TernarySearch(arr, searchTerm, right + 1, arr.Length - 1);
    }
    else if (string.Compare(searchTerm, arr[left]) < 0)
    {
        return TernarySearch(arr, searchTerm, 0, left - 1);
    }

    return TernarySearch(arr, searchTerm, left + 1, right - 1);
}
```

So, the question is, what is the worst-case time complexity of this algorithm, and how does it compare to binary search? This algorithm's time complexity is $\log_3 n$, which boiled down to its dominant factor is equal to log(n). Therefore, the complexity of Ternary Search is the same as Binary Search. In my mind, the follow up question would be: Does this statement remain true in practice? And to that I'd say no. Because Ternary Search must perform so many more comparisons than Binary Search, it makes it measurably slower than Binary Search. Therefore, it's impractical to implement Ternary Search over Binary Search.