ECE 2400 Computer Systems Programming

Topic 8: Complexity Analysis

http://www.csl.cornell.edu/courses/ece2400
School of Electrical and Computer Engineering
Cornell University

revision: 2021-10-14-10-41

Please do not ask for solutions. Students should compare their solutions to solutions from their fellow students, discuss their solutions with the instructors during lab/office hours, and/or post their solutions on Ed for discussion.

List of Problems

1 Short Answer **2**

2 K-ary and Hybrid Search **3**

2.A K-ary Search Algorithm 3

2.B Hybrid Search Algorithm 4

2.C Comparing Search Algorithms 5

Note that there are also problems related to complexity analysis in earlier topics!

Problem 1. Short Answer

Carefully plan your solution before starting to write your response. Please be brief and to the point; if at all possible, limit your answers to the space provided.

Problem 2. K-ary and Hybrid Search

In lecture we learned about linear and binary search. In this problem, you will explore two more advanced searching algorithms: k-ary search and hybrid search. You will then compare and contrast these algorithms with linear and binary search.

Part 2.A K-ary Search Algorithm

A binary search algorithm assumes the input array is sorted. The algorithm recursively divides this array into two halves, compares the search value to the midpoint, and then decides whether to recursively search in the lower or higher half of the array. We can generalize binary search into a K-ary search in which we divide the array into K partitions and then compare the search value to decide which partition to recursively search. A generalized K-ary search algorithm is a little complicated, so in this part we will focus on implementing the 3-ary search algorithm. The algorithm should divide the array into three partitions and then compare the search value to decide which of these three partitions to recursively search.

Develop and implement a ternary (3-ary) search algorithm. Implement your algorithm in a `ternary_search` function which takes as input an array `x`, the size of the array `n`, and a search value `v`. The function should return 1 if the search value is found and 0 if not found. You should use a recursive function and you are free to include a helper function if you wish. Your implementation cannot modify the input array. Your implementation should be correct and efficient in terms of both execution time and space usage. While you are welcome to use pseudo-code to plan your approach, your final solution must be written using valid C syntax.

```
int ternary_search( int* x, int n, int v ) {
```

Part 2.B Hybrid Search Algorithm

A hybrid search will switch from binary search to linear search once the partition is reasonably small. We will use the key parameter M to indicate how small the partition needs to be in order to switch to linear search.

Develop and implement a hybrid search algorithm. Implement your algorithm in a `hybrid_search` function which has the same interface as `ternary_search`. *Your hybrid search should use binary search, not K-ary or 3-ary search for the recursive part of the algorithm!* Your implementation cannot modify the input array. Your implementation should be correct and efficient in terms of both execution time and space usage. While you are welcome to use pseudo-code to plan your approach, your final solution must be written using valid C syntax.

```
const int M = 4;

int hybrid_search( int* x, int n, int v ) {
```

Part 2.C Comparing Search Algorithms

In this problem, you will be qualitatively comparing various search algorithms. **Begin by filling in the following table.** Your analysis for K-ary search should be generalized for any value of K. Your analysis for hybrid search should be generalized for any value of M. *Hint: For K-ary search carefully consider what computation increases and decreases with K!* Your analysis should be for the worst case. *Note that this does not mean the worst case values of K or M.* This means the worst case array data and/or search value which produce the worst case function of N.

	Worst Case Execution Time ($T(N)$)	Worst Case Time Complexity (big-O)
Linear Search		
Binary Search		
Ternary Search		
K-ary Search		
Hybrid Search		

Use these results along with deeper insights to perform a comparative analysis of these search algorithms, with the ultimate goal of **making a compelling argument for which algorithm will perform better across a large number of usage scenarios.** While you are free to use whatever approach you like, we recommend you structure your response in several paragraphs. The **first paragraph** might discuss the performance of linear and binary search using time complexity analysis summarizing what we learned in lecture. Justify the entries in the table. The **second paragraph** might start by discussing the performance of 3-ary search using time complexity analysis before generalizing this analysis for an arbitrary value of K. Justify the entries in the table. Remember that asymptotic big-O time complexity analysis is not the entire story; it is just the starting point for understanding execution time. *Consider how the execution time varies as a function of K. Think critically about what might be an optimum value of K.* The **third paragraph** might discuss the performance of the hybrid search algorithm using time complexity analysis. Justify the entries in the table. Remember that asymptotic big-O time complexity analysis is not the entire story; it is just the starting point for understanding execution time. *Consider how the execution time varies as a function of M. Think critically about what kind of overheads a hybrid search algorithm is trying to optimize.* The **fourth paragraph** might discuss other qualitative metrics such as generality, maintainability, and design complexity. The **final paragraph** can conclude with a compelling argument for which search algorithm will perform better in the general case, or if you cannot strongly argue for any algorithm explain why. Your answer will be assessed on how well you argue your position.