Binary Search and Worst-Case Analysis

Yufei Tao

Department of Computer Science and Engineering
Chinese University of Hong Kong
A significant part of computer science is devoted to understanding the power of the RAM model in solving specific problems. Every time we discuss a problem in this course, we will learn something new.

Today’s lecture is about the dictionary search problem. We will learn not only a fast algorithm for solving this problem, but also a method called worst-case analysis for measuring the quality of an algorithm.
The Dictionary Search Problem

Problem Input:

In the memory, a set $S$ of $n$ integers have been arranged in ascending order at the memory cells from address 1 to $n$. The value of $n$ has been placed in Register 1 of the CPU. Another integer $v$ has been placed in Register 2 of the CPU.

Goal:

Design an algorithm to determine whether $v$ exists in $S$.

Note that we have not specified how your algorithm should indicate the outcome. This is up to you. For example, you may store 0 in a certain register to signify “no”, and 1 for “yes”.

We will refer to the value of $n$ as the problem size.
A “yes”-input with \( n = 16 \)

A “no”-input with \( n = 16 \)
The First Algorithm

Let $n$ be Register 1, and $v$ be Register 2.

Simply read the memory cell of address $i$, for each $i \in [1, n]$ in turn. If any of those cells equals $v$, return yes. Otherwise, return no.

The above is a concise, but clear, description of the same algorithm as in the pseudocode of the next slide.
The First Algorithm in Pseudocode

1. Let \( n \) be register 1, and \( v \) be register 2
2. register \( i \leftarrow 1 \), register \( one \leftarrow 1 \)
3. repeat
4. read into register \( x \) the memory cell at address \( i \)
5. if \( x = v \) then
6. return “yes” (by writing 1 to a register)
7. \( i \leftarrow i + one \) (effectively increasing \( i \) by 1)
8. until \( i > n \)
8. return “no” (by writing 0 to a register)
Running Time of the First Algorithm

How much time does the algorithm require? The answer depends on the problem input. Here are two extreme cases:

- If $v$ is the first element in $S$ (i.e., the integer in the memory cell of address 1), the algorithm has running time 5.
- If we are given a “no”-input, then the algorithm has running time $4n + 3$.

In computer science, it is an art to design algorithms with performance guarantees. In our scenario, this amounts to the question: what is the largest running time on the worst input with $n$ integers?

This gives rise to an important notion in the next slide.
Worst-Case Running Time

The **worst-case cost** (or **worst-case time**) of an algorithm under a problem size $n$, is defined to be the largest running time of the algorithm on all the (possibly an infinite number of) inputs of the same size $n$.

Formally, let $S_n$ be the (possibly infinite) set of all possible inputs of size $n$. Fix an algorithm $A$. For each input $I \in S_n$, define $\text{cost}_A(I)$ as the cost of $A$ on the input $I$. Then, the worst-case cost of $A$ is a function of $n$:

$$f_A(n) = \max_{I \in S_n} \text{cost}_A(I).$$
Our algorithm has worst-case time $f_1(n) = 4n + 3$.

In other words, no matter how you design the input set $S$ of $n$ integers, the algorithm always terminates with a cost at most $4n + 3$. This is its performance guarantee on every $n$. 
Next, we will see another algorithm with much better worst-case time, namely, the *binary search* algorithm.
Binary Search

We will utilize the fact that $S$ has been stored in ascending order. Let us compare $v$ to the element $x$ in the middle of $S$ (i.e., the $(n/2)$-th).

- If $v = x$, we have found $v$, and thus, can terminate.
- If $v < x$, we can immediately forget about the second half of $S$.
- If $v > x$, forget about the first half.

In the 2nd and 3rd cases, we have at most $n/2$ elements left. Then, repeat the trick on those elements!
Conceptually discard the second half of $S$. 
Binary Search

Conceptually discard the first half of what is shown.
Conceptually discard the first half of what is shown.
Binary Search

35

Found.
Binary Search in Pseudocode

1. let $n$ be register 1, and $v$ be register 2
2. register $left \leftarrow 1$, $right \leftarrow n$
3. repeat
4. register $mid \leftarrow (left + right)/2$
5. if the memory cell at address $mid = v$ then
6. \hspace{1cm} return “yes”
7. else if the memory cell at address $mid > v$ then
8. \hspace{1cm} $right = mid - 1$
9. else
10. \hspace{1cm} $left = mid + 1$
11. until $left > right$
12. return “no”
Worst-Case Time of Binary Search

Let us call the integers whose memory addresses are from \textit{left} to \textit{right} as active elements.

Refer to Lines 3-10 as an iteration. Each iteration performs at most 7 atomic operations (try verifying this yourself).
Worst-Case Time of Binary Search

How many iterations are there? After the first iteration, the number of active elements is at most $n/2$. After another, the number is at most $n/4$. In general, after $i$ iterations, the number drops to at most $n/2^i$.

Suppose that there are $h$ iterations in total. It holds that $h$ is the smallest integer satisfying (think: why?)

$$\frac{n}{2^h} < 1$$

which gives $h = \lceil \log_2(n + 1) \rceil$. 

In each iteration we perform only a constant number of operations — we will not analyze this constant precisely, except for pointing out a loose upper bound of 10.

The worst-case time of binary search is at most \( f_2(n) = 10(1 + \log_2 n) \).

When \( n \) is large, this running time is much lower than the time \( 4n + 3 \) of our first algorithm.
In this lecture, we have got a taste of what computer science is like. We are seldom satisfied with just finding an algorithm that can correctly solve a problem. Instead, our goal is to design an algorithm with a strong performance guarantee, i.e., you must prove that it runs fast even in the worst case.