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 RAM's power in solving specific problems. Today, we will discuss the **dictionary search problem**. We will solve the problem with an algorithm called **binary search** and introduce a generic method — called **worst-case analysis** — for measuring the quality of algorithms.

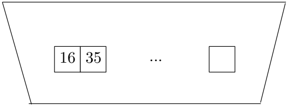
Dictionary Search

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

Goal: Determine whether q exists in S.

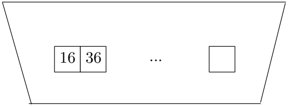
We will refer to n as the **problem size**.

A "yes"-input with $n = 16$



5	9	12	17	26	28	35	38	41	47	52	68	69	72	83	88																

A "no"-input with $n = 16$



5	9	12	17	26	28	35	38	41	47	52	68	69	72	83	88																

The First Algorithm

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

The above is a piece of acceptable description of the same algorithm described by the pseudocode in the next slide.

The First Algorithm in Pseudocode

```
Let n be register 1, and q be register 2
register i ← 1, register one ← 1
repeat
    read into register x the memory cell at address i
    if x = v then
        return "yes" (by writing 1 to a register)
    i ← i + one (effectively increasing i by 1)
until i > n
return "no" (by writing 0 to a register)
```

Running Time of the First Algorithm

The running time depends on the 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 $4n + 3$.

The art of computer science is to design algorithms with performance **guarantees**. In our scenario, what is the **largest** running time on the **worst** input with size n?

Worst-Case Running Time

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

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

$$f_A(n) = \max_{I \in S_n} cost_A(I).$$

Example

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** $4n + 3$.

Binary Search

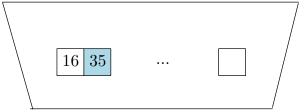
Next, we will see how to solve the problem with a much better worst-case time, utilizing the fact that S has been stored in ascending order.

Let us compare q to the element x in the middle (the $(n/2)$-th) of S.

- If $q = x$, we have found q.
- If $q < x$, we can immediately forget about the second half of S.
- If $q > x$, 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!

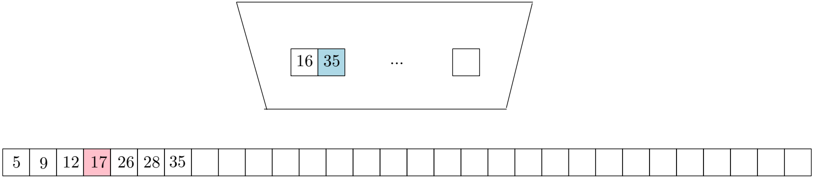
Binary Search



5	9	12	17	26	28	35	38	41	47	52	68	69	72	83	88														

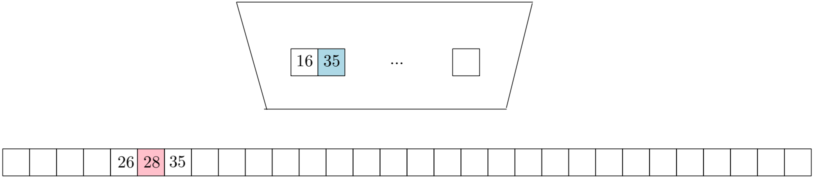
Conceptually discard the second half of S.

Binary Search



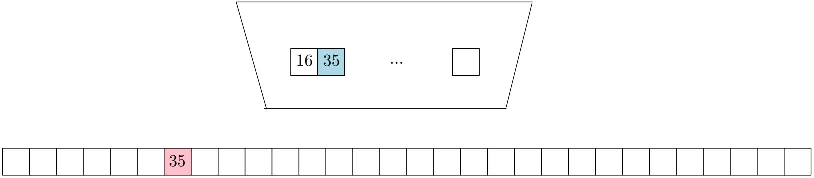
Conceptually discard the first half of what is shown.

Binary Search



Conceptually discard the first half of what is shown.

Binary Search



Found.

Binary Search in Pseudocode

```
let n be register 1 and q be register 2
register left ← 1, right ← n
repeat
    register mid ← (left + right)/2
    read the memory cell at address mid into register x
    if x = q then return "yes"
    else if x > q then right = mid − 1
    else left = mid + 1
until left > right
return "no"
```

Worst-Case Time of Binary Search

Let us use the term **active elements** to refer to the integers stored at memory addresses from *left* to *right*.

Refer to Lines 3-10 as an **iteration**. 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 $h = \lceil \log_2(n+1) \rceil$.

Worst-Case Time of Binary Search (cont.)

In each iteration, we perform only a constant number of operations, for which 10 is a (loose) upper bound.

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.

We have got a taste of what computer science is like. We are seldom satisfied with just finding an algorithm to 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.