

# Measuring the Efficiency of an Algorithm by the Worst Input

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, that is, what would be a “fastest” algorithm for each problem.

But how do we measure “fast”? One approach—the one we follow in this course—is to look at the algorithm’s cost on the **worst** input, as we will formalize in this lecture.

## Cost on the Worst Input

Define  $\mathcal{I}_n$ , where  $n$  is an integer, to be the set of all inputs to a problem that have the same **problem size**  $n$ .

Given an input  $I \in \mathcal{I}_n$ , the cost  $X_{\mathcal{A}}(I)$  of an algorithm  $\mathcal{A}$  is the length of its execution on  $I$ .

- The **worst-case cost** of  $\mathcal{A}$  under the problem size  $n$  is the maximum  $X_{\mathcal{A}}(I)$  of all  $I \in \mathcal{I}_n$ .
- The **worst expected cost** of  $\mathcal{A}$  under the problem size  $n$  is the maximum  $\mathbf{E}[X_{\mathcal{A}}(I)]$  of all  $I \in \mathcal{I}_n$ .

## Example: Dictionary Search

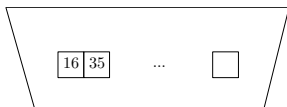
**Problem Input:** In the memory, a set  $S$  of  $n$  integers have been arranged in **ascending** order at the memory cells from address 1 to  $n$ . The value of  $n$  has been placed in Register 1 of the CPU. Another integer  $v$  has been placed in Register 2 of the CPU.

- $n$  is the problem size.
- $\mathcal{I}_n$  is the set of all possible  $(S, v)$ .

**Goal:** Determine **whether  $v$  exists in  $S$ .**

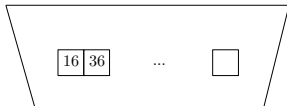
## Example: Dictionary Search

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																				
---	---	----	----	----	----	----	----	----	----	----	----	----	----	----	----	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

## Example 1: Dictionary Search

The worst-case cost of the **binary search algorithm** is  $O(\log n)$ .

In other words, on any input in  $\mathcal{I}_n$ , the maximum number  $f(n)$  of atomic operations performed by the algorithm grows no faster than  $\log_2 n$ .

Note: This does **not** mean  $f(n) = \log_2 n$ .

" $f(n) = O(\log n)$ " only says that  $f(n)$  could be functions like  $10(1 + \log_2 n)$ ,  $352 \log_3 n$ ,  $\sqrt{\log n} + 78 \log_2(n^{83})$ , etc.

## Example 2

Consider the following randomized algorithm:

```
/* A is an array of size  $n$  that contains at least one 0 */
```

1. **do**
2.      $r = \text{RANDOM}(1, n)$
3. **until**  $A[r] = 0$
4. **return**  $r$

What is the expected cost of the algorithm? The answer is “it depends”:

- If all numbers in  $A$  are 0, the algorithm finishes in  $O(1)$  time.
- If  $A$  has only one 0, the algorithm finishes in  $O(n)$  expected time because
  - $A[r]$  has  $1/n$  probability of being 0.
  - In expectation, we need to repeat  $n$  times to find the 0.

## Example 2 (cont.)

/\*  $A$  is an array of size  $n$  that contains at least one 0 \*/

1. **do**
2.      $r = \text{RANDOM}(1, n)$
3. **until**  $A[r] = 0$
4. **return**  $r$

Worst-case cost of the algorithm =  $\infty$

Worst expected cost of the algorithm =  $O(n)$



We will finish the lecture by tapping into the power of randomization. We will see a problem where randomized algorithms are **provably faster** than deterministic ones in expected cost.

Before proceeding, think: what is the “expected cost” of a deterministic algorithm?

## Power of Randomization

**Problem “Find-a-Zero”:** Let  $A$  be an array of  $n$  integers, among which **half** of them are 0. Design an algorithm to report an arbitrary position of  $A$  that contains a 0.

For example, suppose  $A = (9, 18, 0, 0, 15, 0, 33, 0)$ . An algorithm can report 3, 4, 6, or 8.

## Power of Randomization

1. **do**
2.      $r = \text{RANDOM}(1, n)$
3. **until**  $A[r] = 0$
4. **return**  $r$

The algorithm finishes in  $O(1)$  expected time on **every input**  $A$ !

In contrast, **any** deterministic algorithm must probe at least  $n/2$  integers of  $A$  in the worst case! In other words, any deterministic algorithm must have a worst case time of  $\Theta(n)$ —provably slower than the above randomized algorithm (in expectation).