RAM with Randomization and Quick Sort

Yufei Tao

ITEE
University of Queensland

So far all our algorithms are deterministic, namely, they do not involve any randomization. In computer science, randomized algorithms play a very important role. Compared to their deterministic counterpart, such algorithms usually have at least one of the following advantages:

- They may be considerably simpler;
- They may achieve a better performance guarantee in expectation.

In this lecture, we will first slightly extend the RAM model to enable the design of randomized algorithms. Then, we will introduce our first such algorithm: the famous quick sort.

RAM with Randomization

Recall that the RAM model defines a set of atomic operations that can be used to design an algorithm. Henceforth, we will formally extend our RAM model with one more atomic operation:

- RANDOM(x, y): Given integers x and y (satisfying $x \leq y$), this operation returns an integer that is chosen uniformly at random in $[x, y]$.

In other words, RANDOM(x, y) returns a random integer in $[x, y]$—any of x, $x+1$, $x+2$, ... y has the same probability of being returned.

None of the other components in RAM is altered. In particular, an algorithm is still a sequence of atomic operations, and its cost is still the length of the sequence.

Recall:

The Sorting Problem

Problem Input:

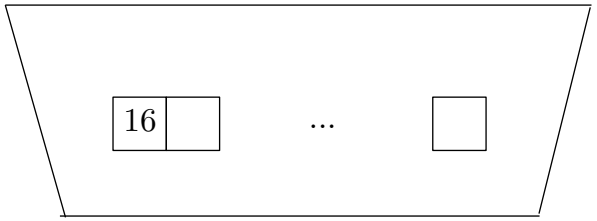
A set S of n integers is given in an array of length n. The value of n is inside the CPU (i.e., in a register).

Goal:

Design an algorithm to store S in an array where the elements have been arranged in ascending order.

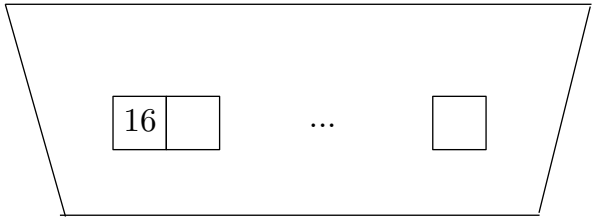
Example

Input:



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

Output:



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

Recall:

The Sorting Problem

Problem Input:

A set S of n integers is given in an array of length n. The value of n is inside the CPU (i.e., in a register).

Goal:

Design an algorithm to store S in an array where the elements have been arranged in ascending order.

Next, we will describe a new algorithm called quick sort to solve this problem with recursion and randomization.

Quick Sort

We will denote the input array as A.

Base Case. If $n = 1$, return directly.

Inductive Case. Otherwise, the algorithm runs the following steps:

1. Randomly pick an integer p in A—call it the pivot.
   - This can be done in $O(1)$ time using RANDOM$(1, n)$.
2. Re-arrange the integers in an array A' such that
   - All the integers smaller than p are positioned before p in A'.
   - All the integers larger than p are positioned after p in A'.
3. Sort the part of A' before p recursively.
4. Sort the part of A' after p recursively.

Example

After Step 1 (suppose that 26 was randomly picked as the pivot):

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

After Step 2:

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

After Steps 3 and 4:

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

Analysis of Quick Sort

As with most randomized algorithms, quick sort's running time is not attractive in the worst case.

- Its worst case time is $O(n^2)$ (verify this by yourself).

However, quick sort is fast in expectation. Notice that the running time of quick sort is a random variable—the cost VARIES, depending on the random choices made. Some choices may lead to a small running time, but others may not. It can be proved that quick sort has expected running time $O(n \log n)$. Intuitively, this means that quick sort runs in $O(n \log n)$ time on average.

> The analysis of quick sort will be covered in the "training camp", and will not be tested in quizzes and exams.