Teaching the Power of Randomization Using a Simple Game

Yana Kortsarts
Computer Science Department,
Widener University, Chester, PA 19013, USA
yanako@cs.widener.edu

Jeffrey Rufinus
Computer Science Department,
Widener University, Chester, PA 19013, USA
rufinus@cs.widener.edu

ABSTRACT

Any deterministic algorithm can be viewed as a game between the algorithm player and the input player. A randomized algorithm can be viewed as a mixed strategy for the first player, used to minimize the disadvantage of being the first to reveal its move. We suggest a simple and accessible guessing game that can serve as both a way to explain notions in algorithms (like worst case input) to students and also to illustrate the power of randomization, presented in an intuitive way.

Categories and Subject Descriptors

K.3.2 [**Computer and Information Science Education**]: Computer science education

General Terms

Algorithms, Theory

Keywords

Randomized algorithms, randomization

1. INTRODUCTION

1.1 Motivation

We consider a simple game with the idea of illustrating the power of randomization. The design of an algorithm for a combinatorial problem can be seen as a game. The algorithm player designs the algorithm. The input player selects the test input for the selected algorithm. The object of the first player is to minimize the running time of the algorithm. The object of the second player is to maximize the running time of the algorithm. Naturally, the best strategy for the second player is finding the worst input for the algorithm produced by the first player. The problem facing the algorithm player is that if it uses a deterministic strategy, then since in a sense it "moves first", the second player can indeed pick the worst example for the suggested algorithm.

However, a randomized algorithm can be seen as a distribution over all possible deterministic algorithms. The algorithm player does not "reveal his cards" fully in advance. It only tells the second player the probability by which it selects any one of the possible deterministic algorithms (the sum of probabilities over all algorithms is 1). The coins have not fallen yet, and the game only begins after the input player chooses its adversarial input. Then, the algorithm to be used is chosen using the distribution defined by the second player.

The advantage to this approach where teaching is concerned is that it eases students' understanding of why randomization is required in algorithms. It is our experience that the idea of a randomized algorithm can be difficult to understand. However, the idea of a randomized algorithm is clearer for students when presented as a game. Indeed, the notion of a game is common in everyday life. When the algorithm is presented as part of a game, the students tend to understand the drawback of a deterministic algorithm that reveals all the cards in advance versus the advantage of a randomized algorithm that does not. A "bad" input for a randomized strategy has to be an input which is bad for several algorithms simultaneously (depending on how the coins fall).

The classic example used to explain the need for randomized algorithms is randomized Quicksort. Deterministic Quicksort (in most of its variations) performs poorly on a sorted or sorted in reverse input (see [3]). The partition that results has one or no elements in one side of the partition and almost all the rest of the elements in the other side of the partition. Randomized Quicksort selects the pivot at random (see [3]). Hence, it is not possible for the adversary to locate the minimum in the array in such a way that the minimum is always selected, etc.

The randomized Quicksort example is important but entails many small and technical points (avoiding infinite loops, discussing the partition algorithm, etc) which may hide some of the ideas we try to teach. In this paper, we suggest a simple game that (unlike other examples) can actually be played in class, and can help in teaching these complicated concepts.

1.2 Why should we teach randomized algorithms in a basic course?

Unlike fundamental subjects like parallel algorithms, online algorithms, approximation algorithms and cryptography, randomization is a general tool that applies in all the above areas and is not just a subject by itself. The significance of randomization in algorithms cannot be overstated. Many of the breakthroughs in various algorithmic areas have used randomization. Just to give a few examples: In approximation algorithms, a breakthrough in proving lower bounds for hard problems stems from the celebrated PCP theorem [1] that gave a new characterization of NP in the language of probabilistically checkable proofs. Randomization is an essential tool in parallel algorithms, for example in finding matchings in parallel [10] and in fast routing [2]. In cryptography, ever since the RSA [12], most cryptographic protocols use randomization. The use of randomization led to the important idea of zero-knowledge protocols [6]. In online algorithms, randomization can bring a competitive ratio for fundamental problems not achievable by deterministic algorithms [5]. Randomization has brought breakthroughs in finding fast combinatorial algorithms for linear programming [9]; examples are abundant. A student who is to learn any of the above subjects is likely to need to know about randomized algorithms. For its basic nature, it seems that at least some examples should be given in the basic algorithms course.

2. THE GAME AND ITS ANALYSIS

2.1 The game

We consider the following simple game:

- Player 1 decides on a positive integral number x, and writes it on a paper.
- Player 2 has to try a sequence of integral number guesses.
- The goal is for player 2 to find a number y_i so that $y_i \geq x$. When such a number is found the game ends.
- The sequence must be strictly increasing (to force termination).
- On a guess y, player 1 either says: "smaller than x", or says "at least as large as x", and reveals x (stopping the game).
- Let the guesses be $y_1, y_2, \ldots, y_n$, so that $y_n \geq x$ and for all $j \leq n-1$, $y_j < x$. The optimization criteria is the performance ratio: $\sum_i y_i/x$.

The game is well-motivated from the point of view of modern scheduling research. Even though this specific game seems not to have been studied before, the techniques illustrated here have been used in a series of papers on approximating scheduling problems, see for example [11, 7, 8, 4]. These papers study the fast scheduling of conflicting jobs with the goal of minimizing the sum of finish times of these jobs. Hence, the suggested game is at the heart of modern research.

In this paper we fully analyze the game. We do not necessarily suggest presenting all details to the students (see later). However, we do provide all details as to how a gap between deterministic and randomized algorithms can be proved in this simple case.

2.2 Teaching the game

The game is so simple it can be shown even to students in basic courses. It is also related to fundamental subjects like binary search.

We played this game with the students in class. The students were the second player. It is our experience that the students have no problem in understanding and playing the game. At first the students just threw numbers out randomly. But, with some guidance, they reached some more systematic strategies. Interestingly, our experience shows that at first, when the students chose increasing numbers with no specific order in their choices, the performance ratio was usually around 3, as predicted by the analysis of the random strategy (see below).

In our opinion, the students should be given the opportunity to play a few examples before discussing general strategies. Also, we recommend discussing the objective function selected. For example, why did we not select just y_i/x with y_i the first larger than x integer suggested by the second player? Clearly, if y_i/x is the objective function then the simple strategy of selecting, 1,2,3,.... is optimal. It is our experience that the students themselves arrive at this conclusion when given enough time. Discussing possible objective functions enhances student understanding of why the selected objective function makes sense. Indeed, the $1, 2, 3, \ldots$ simple strategy has very bad performance for the game with $\sum y_j/x$ optimization value.

When discussing possible strategies, we recommend asking such questions as : why not start with some "large" number (in hope of immediately getting to a larger than x number)? Why do we not benefit from increasing the next guess only a little compared to previous guess? What is the disadvantage of making the next guess, say, 100 times larger than the previous guess? And so on. Such a discussion helps encourage systematic reasoning in students.

2.3 The powers of 2 strategy for the second player

It turns out that the simple strategy that selects $y_1 = 1$, and $y_{i+1} = 2 \cdot y_i$ (the strategy that selects powers of 2) is an optimal deterministic strategy for this game. We provide a proof for that in the next subsection. The worst case for the strategy is when the number selected by the first player is $x = 2^j + 1$ for some large j. The game then is played until the second player suggests 2^j, and then 2^{j+1}, and the game stops. Thus, as

$$\sum_{i \leq j} 2^i = 2^{j+1} - 1$$

and the last guess is 2^{j+1}, the sum of guesses is $2^{j+2} - 1$, and the ratio of the sum over x is close to 4.

The students should be encouraged to find by themselves the worst case for the powers of 2 strategy. This is typically not too difficult for students and it resembles the construction of a bad input for a deterministic algorithm. However, in some cases the students at first had a hard time seeing that the performance ratio of the strategy is 4. Some of them suggest the performance ratio of 2, etc. A concrete example using reasonably large powers of 2 helps.

The above example serves well in illustrating the strict notion of worst case input. The bad instance for the doubling strategy is a very specific and rare number. It is usually well understood by students that if x is some random number, the doubling strategy performs much better. This may also be a good place to discuss the difference between random strategy and random inputs. One should emphasize that the input is sometimes not within our control, while the randomized algorithm is within our control as the designers of the algorithms.

3. A LOWER BOUND FOR ANY DETERMINISTIC STRATEGY

3.1 Explaining the goal to students

Let $\epsilon > 0$ be a small as desired constant. We show that any deterministic strategy has examples with performance ratio at least $4 - \epsilon$. Thus, the doubling strategy is optimal.

We doubt if all the details to follow should be taught in class. This hides the concepts behind technical details. We would suggest talking instead about the philosophy of what we try to do here.

However, one point should be made clear. We are not proving a lower bound for a *specific strategy*, like the powers of 2 strategy. The proof given shows that *any* deterministic strategy has examples where it has performance ratio arbitrarily close to 4. To illustrate this point to students, we suggest discussing several deterministic strategies and exhibiting their respective bad examples. The bad example for *any* strategy can be deduced from the very general lower bound presented here.

3.2 An adversarial strategy

We now present a strategy for an adversary as follows. We denote the ith choice of the second player by y_i and note that $y_{i+1} > y_i$. Define δ_i, μ_i so that:

$$y_i = \delta_i \cdot \sum_{j=1}^{i-1} y_j,\ y_{i+1} = \mu_i y_i.$$

We may assume:

- **The start point assumption:** $y_1 > \max\{4, 1/\epsilon^2\}$. Otherwise, ignore some of the first integers. This gives a new equivalent strategy with large y_1.
- **The bounded ratio assumption:** We may assume $\delta_i \le 4$ for every i. Otherwise, in the worst case $x = y_i + 1$. The ratio will be at least:

$$\frac{4y_i + y_i}{y_i + 1} = 5 - \frac{5}{y_i + 1} \ge 4.$$

The last inequality holds as $y_i \ge 4$.

A stopping condition: In any iteration, if

$$\mu_i \ge (1 - \epsilon/8)(1 + \delta_i)$$

the adversary chooses

$$x = y_i + 1 < y_{i+1}$$

stopping the game. We now consider the case that the stopping condition does not apply. Let $r = y_2/y_1$ and $\nu = \lceil 8/\epsilon \ln(3/r) \rceil$. If for $\nu + 2$ consecutive times the stopping condition does not apply, let y_q be the last number by player 2. Then, the adversary replies that $x = y_q$, stopping the game.

When teaching the details of the adversarial strategy to students, we found out that the concept of choosing x "against" the choices of $\{y_i\}$ is hard to understand for students. Some ask: are we changing x as the second player makes its selections? Some examples using specific deterministic strategies can help in explaining that the adversarial strategy is a proof for the *existence* of a bad x. Such a bad x is to be considered when talking about the worst case.

3.3 Analysis

If in one of the $\nu + 2$ iterations,

$$\mu_i \ge \left(1 - \frac{\epsilon}{8}\right)(1 + \delta_i)$$

then the ratio is given by:

$$\frac{\left(\frac{1}{\delta_i} + 1 + \mu_i\right) \cdot y_i}{y_i + 1} \ge 4 - \epsilon.$$

The last inequality is proven using

1. ϵ small enough,
2. $\delta_i \le 4$,
3. $y_1 \ge 1/\epsilon^2$ and
4. $2 + 1/x + x \ge 4$ for every x.

The second case is that the stopping condition fails $\nu + 2$ consecutive times. In every such case:

$$\delta_{i+1} = \frac{\mu_i \cdot \delta_i}{\delta_i + 1} < \left(1 - \frac{\epsilon}{8}\right)\delta_i.$$

And so, since the stopping condition does not apply $\nu + 2$ times:

$$\delta_{\nu+2} < \left(1 - \frac{\epsilon}{8}\right)^{\nu} \cdot r < 1/3.$$

In other words: $3y_q < \sum_{j=1}^{q-1} y_j$. And so, the ratio is at least:

$$\frac{(y_q + 3 \cdot y_q)}{y_q} = 4.$$

3.4 An intuitive explanation for the students

While the details of the analysis are simple enough and involve only basic algebra, we do believe that discussing these details can hide the intuition behind the proof. Hence, we recommend giving the students a printed proof and only discussing the high level ideas of the proof in class.

The high level ideas are as follows:

- The second player cannot choose y_{i+1} to be "too large" compared to y_i as, if this is the case, it may be that $x = y_i + 1$ and so $(y_{i+1} + y_i)/x$ is already a large number.
- Thus, the second player always selects y_{i+1} to be not much larger than y_i. This is in order to avoid the bad example presented above.
- However, if after many times the second player makes such choices, we get to a stage so that for some y_j, $\sum_{i \le j-1} y_i$ is much larger than y_j. This holds true as the choices are "slow" to increase. Hence, the sum of previous numbers is more significant than the last number. Hence, in this case the choice $x = y_j$ is bad for the second player as

$$(\sum_{i \le j-1} y_i + y_j)/y_j$$

is large.

3.5 A randomized strategy

The following simple randomized strategy gives an improved expected value. Let $\alpha \in_R [0,1)$. Namely, α is randomly and uniformly chosen from the interval $[0,1)$.

We define our randomized strategy by taking

$$y_j = \lfloor exp(j+\alpha) \rfloor.$$

We now analyze the expected performance ratio of the randomized strategy.

Let i so that:

$$\lfloor exp(i-1+\alpha) \rfloor < x \le \lfloor exp(i+\alpha) \rfloor$$

and let us compute $E(exp(i+\alpha))$. Consider the random variable:

$$j(\alpha) = i + \alpha - \ln x.$$

Let

$$\mu = \ln x - \lfloor \ln x \rfloor.$$

It is not hard to see that: If

$$\alpha \le \mu,$$

then

$$j(\alpha) = 1 + \alpha - \mu.$$

Else,

$$j(\alpha) = \alpha - \mu.$$

It follows that $j(\alpha) = j_x(\alpha)$ is $1-1$ and onto $[0,1)$, hence invertible.

We recommend not getting into these details and simply stating that it is possible to show that $j(\alpha)$ is a $1-1$ and onto $[0,1)$. Thus, since α is random so is $j(\alpha)$.

Hence: $j(\alpha) \in_R [0,1)$. Hence: $E(exp(i+\alpha-\ln x)) = e-1$. Observe that the computation of the expectation here is by integrals (because the distribution is continuous). Some of these details can be avoided if necessary (depending on the students' understanding of the material).

Thus: $E(exp(i+\alpha)) = (e-1)\cdot x$. The sum corresponding to the strategy is:

$$\sum_{j \le i} exp(j+\alpha) = exp(\alpha)\cdot((exp(i)-1))/(e-1) + exp(i+\alpha).$$

The expected value of the strategy is thus:

$$(e \cdot x - 1)/x < e$$

.

The useful issue with the game is that if one wants to avoid the technical details in the proof, one can simply run the game with the random strategy on a few examples. It is very simple to write a short program to produce the above described strategy. One can use the random seed of the computer in order to select a random number between 0 and 1. This may illustrate the power of randomization in a practical setting. By the law of large numbers, if the teacher runs the algorithm many times on the same x, the performance ratio approaches $(e \cdot x - 1)/x$.

4. REFERENCES

[1] S. Arora and C. Lund and R. Motwani and M. Sudan and M. Szegedy. Proof verification and the hardness of approximation problems. Journal of ACM, 45(3):501-555, 1998.

[2] G. J. Brebner and L. G. Valiant, Universal schemes for parallel communication. Proceedings of the thirteenth annual ACM symposium on Theory of computing, Pages: 263 - 277, 1981

[3] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. Introduction to algorithms. *The MIT Press, 2nd edition*, 2001.

[4] S. Chakrabarti, C. A. Phillips, A. S. Schulz, D. B. Shmoys, C. Stein and J. Wein. Improved scheduling algorithms for minsum criteria. *ICALP '96*, 875–886.

[5] A. Fiat, R. M. Karp, M. Luby, L. A. McGeoch, D. D. Sleator, N. E. Young, Competitive paging algorithms. Journal of Algorithms archive Volume 12(4): 685 - 699 1991

[6] O. Goldreich, S. Micali, A. Wigderson. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. *Journal of the ACM*, 38(3):690 - 728, 1991

[7] L. A. Hall, D. B. Shmoys, and J. Wein. Scheduling to minimize average completion time: Off-line and on-line algorithms. *SODA'96*, 142–151. 42–151, Jan 1996.

[8] L. A. Hall, A. Schulz, D. B. Shmoys, and J. Wein. Scheduling to minimize average completion time: Off-line nd on-line approximation algorithms. *Math. Operations Research* 22:513–544, 1997.

[9] G. Kalai, A subexponential randomized simplex algorithm, Proceedings of the twenty-fourth annual ACM symposium on Theory of computing, 475 - 482, 1992

[10] R. M. Karp, E. Upfal and A. Wigderson. Constructing a perfect matching is in random NC. Combinatorica Volume 6(1):35–48, 1986

[11] M. Queyranne, M. Sviridenko. Approximation algorithms for shop scheduling problems with minsum objective. *J. Scheduling* 5:287-305, 2002.

[12] R. L. Rivest, A. Shamir, L. M. Adleman, A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. Commun. ACM 21(2):120-126, 1978