A discussion on algorithms

R. Ramanujam

The Institute of Mathematical Sciences, Chennai, India

email: jam@imsc.res.in

FDP on Algorithms, SSNCE

January 25, 2016

First words

- I thank SSNCE for providing me with this opportunity.
- The talk owes a great deal to long discussions over many years with my colleague Venkatesh Raman.
- The presentation follows style and material from many online sources, especially notes by Erik Demaine, Erikson, Kozen, Parberry.
- Please feel free to interrupt any time.

A quote

Some words well worth listening to:

> *We should explain, before proceeding, that it is not our object to consider this programme with reference to the actual arrangement of the data on the Variables of the engine, but simply as an abstract question of the nature and number of the operations required to be perfomed during its complete solution.*
>
> *Ada Augusta Byron King, Countess of Lovelace (1843)*

A few questions

Some questions we need answers to:

- Why should we study algorithms ?

A few questions

Some questions we need answers to:

- Why should we study algorithms ?
- Why should we study algorithms ?

A few questions

Some questions we need answers to:

- Why should we study algorithms ?
- Why should we study algorithms ?
- What can an algorithms course expect to teach a (reasonably sincere) student ?

A few questions

Some questions we need answers to:

- Why should we study algorithms ?
- Why should we study algorithms ?
- What can an algorithms course expect to teach a (reasonably sincere) student ?
- How well do we achieve these objectives ? Why ?

A science ?

Is computer science really a science ?

A science ?

Is computer science really a science ?

- What are the elements of any machine science ?

A science ?

Is computer science really a science ?

- What are the elements of any machine science ?
- A proposal for a pressure cooker science !

A science ?

Is computer science really a science ?

- What are the elements of any machine science ?
- A proposal for a pressure cooker science !
- The two paradigms of computer science.

What is easy?

What is the *hardest* problem you have ever solved ?

What is easy?

What is the *hardest* problem you have ever solved ?

- What is hard for one may be easy for another.

What is easy?

What is the *hardest* problem you have ever solved ?

- What is hard for one may be easy for another.
- What is hard today may be easy tomorrow.

What is easy?

What is the *hardest* problem you have ever solved ?

- What is hard for one may be easy for another.
- What is hard today may be easy tomorrow.
- Difficulty vs Hardness as discussed here.

An exercise

What happens when you login ?

An exercise

What happens when you login ?

- One way functions.

History

Algorithm, named after the 9th century Persian mathematician Abu Abd Allah Muhammad ibn Musa al-Khwarizmi.

History

Algorithm, named after the 9th century Persian mathematician Abu Abd Allah Muhammad ibn Musa al-Khwarizmi.

- Until recently, the word algorithm referred exclusively to pencil-and-paper methods for numerical calculations. People trained in the reliable execution of these methods were called

History

Algorithm, named after the 9th century Persian mathematician Abu Abd Allah Muhammad ibn Musa al-Khwarizmi.

- Until recently, the word algorithm referred exclusively to pencil-and-paper methods for numerical calculations. People trained in the reliable execution of these methods were called *computers* !

History

Algorithm, named after the 9th century Persian mathematician Abu Abd Allah Muhammad ibn Musa al-Khwarizmi.

- Until recently, the word algorithm referred exclusively to pencil-and-paper methods for numerical calculations. People trained in the reliable execution of these methods were called *computers* !
- The word computer comes from the Latin word *putare* which means 'to trim/prune', 'to clean', 'to arrange', 'to value', 'to judge', and 'to consider/suppose'.

History

Algorithm, named after the 9th century Persian mathematician Abu Abd Allah Muhammad ibn Musa al-Khwarizmi.

- Until recently, the word algorithm referred exclusively to pencil-and-paper methods for numerical calculations. People trained in the reliable execution of these methods were called *computers* !
- The word computer comes from the Latin word *putare* which means ‘to trim/prune’, ‘to clean’, ‘to arrange’, ‘to value’, ‘to judge’, and ‘to consider/suppose’.
- An algorithm from the Rhind papyrus (19th century BCE).

Correctness

The Rhind papyrus algorithm breaks the difficult task of general multiplication into four simpler operations:

- determining parity (even or odd),
- addition,
- doubling, and
- mediation (halving a number, rounding down).

Correctness

The Rhind papyrus algorithm breaks the difficult task of general multiplication into four simpler operations:

- determining parity (even or odd),
- addition,
- doubling, and
- mediation (halving a number, rounding down).
- Its correctness follows from the recursive identity: for non-negative integers x, y, $x \cdot y = 0$ if $x = 0$. When x is even, $x \cdot y = \lfloor x/2 \rfloor \cdot (y + y)$. When x is odd, $x \cdot y = \lfloor x/2 \rfloor \cdot (y + y) + y$.

An algorithm?

Here is a recipé to become a millionaire (or pauper).

- You place some amount of money, say x, on the betting table. A fair coin is tossed. If it comes up Heads, you lose your x. If it comes up Tails, you get your x and another x.

An algorithm?

Here is a recipé to become a millionaire (or pauper).

- You place some amount of money, say x, on the betting table. A fair coin is tossed. If it comes up Heads, you lose your x. If it comes up Tails, you get your x and another x.
- Here is a rich man's strategy. Initially he will place Re. 1.00 on the table. At any stage, if he loses x, he will play again with $2x$. However, at any stage if he wins, he will stop playing and go home.

An algorithm?

Here is a recipé to become a millionaire (or pauper).

- You place some amount of money, say x, on the betting table. A fair coin is tossed. If it comes up Heads, you lose your x. If it comes up Tails, you get your x and another x.
- Here is a rich man's strategy. Initially he will place Re. 1.00 on the table. At any stage, if he loses x, he will play again with $2x$. However, at any stage if he wins, he will stop playing and go home.
- Does this constitute an algorithm for the rich man ?

An algorithm?

Here is a recipé to become a millionaire (or pauper).

- You place some amount of money, say x, on the betting table. A fair coin is tossed. If it comes up Heads, you lose your x. If it comes up Tails, you get your x and another x.
- Here is a rich man's strategy. Initially he will place Re. 1.00 on the table. At any stage, if he loses x, he will play again with $2x$. However, at any stage if he wins, he will stop playing and go home.
- Does this constitute an algorithm for the rich man ?
- What is the probability that the gambling does eventually halt?

Correctness

We require algorithms that are correct for **all** possible inputs.

Correctness

We require algorithms that are correct for **all** possible inputs.

- We must *prove* that our algorithms are correct.

Correctness

We require algorithms that are correct for **all** possible inputs.

- We must *prove* that our algorithms are correct.
- Correctness proofs almost always involve induction.

Correctness

We require algorithms that are correct for **all** possible inputs.

- We must *prove* that our algorithms are correct.
- Correctness proofs almost always involve induction.
- We have to formally *state* what it's supposed to do.

Correctness

We require algorithms that are correct for **all** possible inputs.

- We must *prove* that our algorithms are correct.
- Correctness proofs almost always involve induction.
- We have to formally *state* what it's supposed to do.
- It is important to remember the distinction between a problem and an algorithm.

Correctness

We require algorithms that are correct for **all** possible inputs.

- We must *prove* that our algorithms are correct.
- Correctness proofs almost always involve induction.
- We have to formally *state* what it's supposed to do.
- It is important to remember the distinction between a problem and an algorithm.
- Often, the hardest part of answering any question is figuring out the right way to ask it !

Analysis

Ideally, we want the fastest possible algorithm for any particular problem.

Analysis

Ideally, we want the fastest possible algorithm for any particular problem.

- We require algorithms that always run efficiently, even in the *worst case*.

Analysis

Ideally, we want the fastest possible algorithm for any particular problem.

- We require algorithms that always run efficiently, even in the *worst case*.
- How do we determine which is the worst case instance ?

Analysis

Ideally, we want the fastest possible algorithm for any particular problem.

- We require algorithms that always run efficiently, even in the *worst case*.
- How do we determine which is the worst case instance ?
- How do we measure running time ?

Analysis

Ideally, we want the fastest possible algorithm for any particular problem.

- We require algorithms that always run efficiently, even in the *worst case*.
- How do we determine which is the worst case instance ?
- How do we measure running time ?
- Sometimes we are also interested in other computational resources: space, randomness, inter-process messages, and so forth. But the techniques are similar.

An example

An online matching service "We Match You" ! Give your preferences, and we will find the Right One for you !

An example

An online matching service "We Match You" ! Give your preferences, and we will find the Right One for you !
For simplicity assume:

- n men, n women.
- Every woman ranks all men, no ties.
- Every man ranks all women, no ties.

Quality of solution

We are looking for a bijection between two sets of the same size, obviously there are lots of them. But how do we say a matching is good ?

Quality of solution

We are looking for a bijection between two sets of the same size, obviously there are lots of them. But how do we say a matching is good ?

- Suppose there exist women 1 and 2, and men A and B such that:
  - We match 1 with A, 2 with B.
  - But 1 prefers B over A; B prefers 1 over 2.
- Surely 1 and B would prefer beng matched with each other over the current assignment.
- Thus we can define a good matching to be one where such a thing would not happen, but it is no longer clear that a good matching exists !

An algorithm

"We Match You" goes about solving the problem as follows:

- An arbitrary unassigned man A makes an offer to the best woman u (according to the man's preference list) who has not already rejected it.

An algorithm

"We Match You" goes about solving the problem as follows:

- An arbitrary unassigned man A makes an offer to the best woman u (according to the man's preference list) who has not already rejected it.
- Each woman ultimately accepts the best offer that she receives, according to her preference list. Thus, if u is currently unassigned, she (tentatively) accepts the offer from A. If u already has an assignment but prefers A, she rejects her existing assignment and (tentatively) accepts the new offer from A. Otherwise, u rejects the new offer.

An algorithm

"We Match You" goes about solving the problem as follows:

- An arbitrary unassigned man A makes an offer to the best woman u (according to the man's preference list) who has not already rejected it.
- Each woman ultimately accepts the best offer that she receives, according to her preference list. Thus, if u is currently unassigned, she (tentatively) accepts the offer from A. If u already has an assignment but prefers A, she rejects her existing assignment and (tentatively) accepts the new offer from A. Otherwise, u rejects the new offer.
- An instance of the problem.

The solution

We have a good matching. Just how good is it ?

- No woman was matched with her favourite man.
- No man was matched with his favourite woman.
- At least one man was matched with his least favourite woman.
- The assignment is stable, not subject to deviation.

The solution

We have a good matching. Just how good is it ?

- No woman was matched with her favourite man.
- No man was matched with his favourite woman.
- At least one man was matched with his least favourite woman.
- The assignment is stable, not subject to deviation.
- This is not the only one; the matching $(A, r), (B, s), (C, q), (D, t)$ is also stable.

Running time

Each man makes an offer to each woman at most once, so the algorithm requires at most n^2 rounds.

Running time

Each man makes an offer to each woman at most once, so the algorithm requires at most n^2 rounds.

- We can use representations like $M[i,j]$ and $W[i,j]$ where $M[i,j]$ gives the j^{th} man in woman i's list.

Running time

Each man makes an offer to each woman at most once, so the algorithm requires at most n^2 rounds.

- We can use representations like $M[i,j]$ and $W[i,j]$ where $M[i,j]$ gives the j^{th} man in woman i's list.
- A somewhat harder exercise is to prove that there are inputs (and choices of who makes offers when) that force n^2 rounds before the algorithm terminates.

Running time

Each man makes an offer to each woman at most once, so the algorithm requires at most n^2 rounds.

- We can use representations like $M[i,j]$ and $W[i,j]$ where $M[i,j]$ gives the j^{th} man in woman i's list.
- A somewhat harder exercise is to prove that there are inputs (and choices of who makes offers when) that force n^2 rounds before the algorithm terminates.
- Thus, the upper bound on the worst-case running time cannot be improved; in this case, we say our analysis is tight.

Correctness

This algorithm is often misattributed to David Gale and Lloyd Shapley, who formally analyzed the algorithm and first proved that it computes a stable matching in 1962.

Correctness

This algorithm is often misattributed to David Gale and Lloyd Shapley, who formally analyzed the algorithm and first proved that it computes a stable matching in 1962.

- The algorithm continues as long as there is at least one unfilled position.

Correctness

This algorithm is often misattributed to David Gale and Lloyd Shapley, who formally analyzed the algorithm and first proved that it computes a stable matching in 1962.

- The algorithm continues as long as there is at least one unfilled position.
- Conversely, when the algorithm terminates, every position is filled.

Correctness

This algorithm is often misattributed to David Gale and Lloyd Shapley, who formally analyzed the algorithm and first proved that it computes a stable matching in 1962.

- The algorithm continues as long as there is at least one unfilled position.
- Conversely, when the algorithm terminates, every position is filled.
- No man can make an offer to more than one woman, and no woman accepts an offer from more than one man.

Correctness

This algorithm is often misattributed to David Gale and Lloyd Shapley, who formally analyzed the algorithm and first proved that it computes a stable matching in 1962.

- The algorithm continues as long as there is at least one unfilled position.
- Conversely, when the algorithm terminates, every position is filled.
- No man can make an offer to more than one woman, and no woman accepts an offer from more than one man.
- So the algorithm does compute a matching. How do we show it is stable ?

Proof of stability

The argument is surprisingly simple.

- Suppose woman u is assigned to man A in the final matching, but prefers B.
- Because every woman accepts the best offer she receives, u received no offer she liked more than A.
- In particular, B never made an offer to u.
- On the other hand, B made offers to every woman he likes more than v.
- Thus, B prefers v to u, and so there is no instability.

More to analysis

We are not done yet. Does it matter who makes the offer in the first round ?

More to analysis

We are not done yet. Does it matter who makes the offer in the first round ?

- We can show that no matter which unassigned man makes an offer in each round, the algorithm always computes the same matching!

More to analysis

We are not done yet. Does it matter who makes the offer in the first round ?

- We can show that no matter which unassigned man makes an offer in each round, the algorithm always computes the same matching!
- Let us say that u is a feasible choice for A if there is a stable matching that assigns u to A.

More to analysis

We are not done yet. Does it matter who makes the offer in the first round ?

- We can show that no matter which unassigned man makes an offer in each round, the algorithm always computes the same matching!
- Let us say that u is a feasible choice for A if there is a stable matching that assigns u to A.
- Lemma: According to the algorithm, each man A is rejected only by women who are infeasible for A.

Proof of Lemma

Lemma: According to the algorithm, each man A is rejected only by women who are infeasible for A.

- We prove the lemma by induction. Consider an arbitrary round of the algorithm, in which woman u rejects man A for B.

Proof of Lemma

Lemma: According to the algorithm, each man A is rejected only by women who are infeasible for A.

- We prove the lemma by induction. Consider an arbitrary round of the algorithm, in which woman u rejects man A for B.
- The rejection implies that u prefers B to A.

Proof of Lemma

Lemma: According to the algorithm, each man A is rejected only by women who are infeasible for A.

- We prove the lemma by induction. Consider an arbitrary round of the algorithm, in which woman u rejects man A for B.
- The rejection implies that u prefers B to A.
- Every woman that appears higher than u in B's preference list has already rejected B and therefore, by the inductive hypothesis, is infeasible for B.

Proof of Lemma

Lemma: According to the algorithm, each man A is rejected only by women who are infeasible for A.

- We prove the lemma by induction. Consider an arbitrary round of the algorithm, in which woman u rejects man A for B.
- The rejection implies that u prefers B to A.
- Every woman that appears higher than u in B's preference list has already rejected B and therefore, by the inductive hypothesis, is infeasible for B.
- Now consider an arbitrary matching that assigns u to A. We have already established that u prefers B to A. If B prefers u over his partner, the matching is unstable.

Proof of Lemma

Lemma: According to the algorithm, each man A is rejected only by women who are infeasible for A.

- We prove the lemma by induction. Consider an arbitrary round of the algorithm, in which woman u rejects man A for B.
- The rejection implies that u prefers B to A.
- Every woman that appears higher than u in B's preference list has already rejected B and therefore, by the inductive hypothesis, is infeasible for B.
- Now consider an arbitrary matching that assigns u to A. We have already established that u prefers B to A. If B prefers u over his partner, the matching is unstable.
- On the other hand, if B prefers his partner over u, then the partner is infeasible, and again the matching is unstable. We conclude that there is no stable matching that assigns u to A.

A corollary

Let $best(A)$ denote the highest-ranked feasible woman on A's preference list.

A corollary

Let $best(A)$ denote the highest-ranked feasible woman on A's preference list.

- Lemma implies that every woman that A prefers to its final assignment is infeasible for A.

A corollary

Let $best(A)$ denote the highest-ranked feasible woman on A's preference list.

- Lemma implies that every woman that A prefers to its final assignment is infeasible for A.
- On the other hand, the final matching is stable, so the woman assigned to A is feasible for A. Thus we have:

A corollary

Let $best(A)$ denote the highest-ranked feasible woman on A's preference list.

- Lemma implies that every woman that A prefers to its final assignment is infeasible for A.
- On the other hand, the final matching is stable, so the woman assigned to A is feasible for A. Thus we have:
- Corollary: The algorithm assigns $best(A)$ to A, for every A.

Another consequence

The algorithm does the best for every man A, but also does the worst possible from a woman's point of view !

Another consequence

The algorithm does the best for every man A, but also does the worst possible from a woman's point of view !

- Let $worst(u)$ denote the lowest-ranked feasible man on u's preference list.
- Lemma: According to the algorithm, each woman u is assigned $worst(u)$.

Another consequence

The algorithm does the best for every man A, but also does the worst possible from a woman's point of view !

- Let $worst(u)$ denote the lowest-ranked feasible man on u's preference list.
- Lemma: According to the algorithm, each woman u is assigned $worst(u)$.
- Suppose the algorithm matches u with A. Consider an arbitrary stable matching where A is matched with $v \neq u$. By previous corollary, $best(A) = v$. But the matching is stable, so u prefers her assigned man to A.

Another consequence

The algorithm does the best for every man A, but also does the worst possible from a woman's point of view !

- Let *worst*(u) denote the lowest-ranked feasible man on u's preference list.
- Lemma: According to the algorithm, each woman u is assigned *worst*(u).
- Suppose the algorithm matches u with A. Consider an arbitrary stable matching where A is matched with $v \neq u$. By previous corollary, *best*$(A) = v$. But the matching is stable, so u prefers her assigned man to A.
- This works for every stable assignment, so u prefers *every* assigned match over A; that is, $A =$ *worst*(u).

Main reason

The study of algorithms is ultimately about learning two skills that are crucial for all computer scientists.

Main reason

The study of algorithms is ultimately about learning two skills that are crucial for all computer scientists.

- **Intuition**: How to think about abstract computation.
- **Language**: How to talk about abstract computation.
- As teachers, our main role is to share the conviction that thinking and talking about abstract computation is crucial for computer science students.

Intuition

What does it mean to develop algorithmic intuition ?

Intuition

What does it mean to develop algorithmic intuition ?

- How do various algorithms really work ?
- When you see a problem for the first time, how should you attack it ?
- How do you tell which techniques will work at all, and which ones will work best ?
- How do you judge whether one algorithm is better than another ?
- How do you tell whether you have the best possible solution ?

Intuition

What does it mean to develop algorithmic intuition ?

- How do various algorithms really work ?
- When you see a problem for the first time, how should you attack it ?
- How do you tell which techniques will work at all, and which ones will work best ?
- How do you judge whether one algorithm is better than another ?
- How do you tell whether you have the best possible solution ?
- These are not easy questions.

Algorithmic facts

Along the way, we also pick up a bunch of algorithmic facts.

- Mergesort runs in $\Theta(n\ log\ n)$ time.
- The amortized time to search in a splay tree is $O(logn)$.
- Greedy algorithms don't always produce optimal solutions.
- The traveling salesman problem is NP-hard.

Algorithmic facts

Along the way, we also pick up a bunch of algorithmic facts.

- Mergesort runs in $\Theta(n \; log \; n)$ time.
- The amortized time to search in a splay tree is $O(logn)$.
- Greedy algorithms don't always produce optimal solutions.
- The traveling salesman problem is NP-hard.
- But these are not the main point; they can be learnt from wikipedia.

Algorithmic facts

Along the way, we also pick up a bunch of algorithmic facts.

- Mergesort runs in $\Theta(n \; log \; n)$ time.
- The amortized time to search in a splay tree is $O(logn)$.
- Greedy algorithms don't always produce optimal solutions.
- The traveling salesman problem is NP-hard.
- But these are not the main point; they can be learnt from wikipedia.
- The point is to provide enough intuition and experience to know what to look for.

Algorithmic skills

The study does aim to provide practice in a lot of algorithm design and analysis skills.

Algorithmic skills

The study does aim to provide practice in a lot of algorithm design and analysis skills.

- Finding useful examples and counterexamples
- Developing induction proofs
- Solving recurrences
- Using big-Oh notation
- Using probability
- Giving problems crisp mathematical descriptions, and so on.

Algorithmic skills

The study does aim to provide practice in a lot of algorithm design and analysis skills.

- Finding useful examples and counterexamples
- Developing induction proofs
- Solving recurrences
- Using big-Oh notation
- Using probability
- Giving problems crisp mathematical descriptions, and so on.
- This is all incredibly useful for developing intuition, but this is not the main point either.

A good reason

Then what is indeed the main reason to study algorithms ?

A good reason

Then what is indeed the main reason to study algorithms ?

- Good algorithms are extremely useful, elegant, surprising, deep, even beautiful.

A good reason

Then what is indeed the main reason to study algorithms ?

- Good algorithms are extremely useful, elegant, surprising, deep, even beautiful.
- But, most importantly, algorithms are fun !

Discussion time

Thank you.

Questions, comments, suggestions welcome; also, please write to jam@imsc.res.in.