Algorithms: COMP3121/3821/9101/9801

Aleks Ignjatović, ignjat@cse.unsw.edu.au
office: 504 (CSE building); phone: 5-6659
Course Admin: Anahita Namvar, comp3121.unsw@gmail.com

School of Computer Science and Engineering
University of New South Wales Sydney

INTRODUCTION

Introduction

What is this course about?

It is about **designing algorithms** for solving problems.

Why should you study algorithms design?

Can you find every algorithm you might need using Google?

Our goal:

To learn **techniques** which can be used to solve **new, unfamiliar** problems that arise in a rapidly changing field.

Course content:

- a survey of algorithms **design techniques**
- particular algorithms will be mostly used to illustrate design techniques
- emphasis on development of **algorithm design skills**

Textbooks

Textbook:

Kleinberg and Tardos: *Algorithm Design*
paperback edition available at the UNSW book store
good: very readable (and very pleasant to read!); an excellent textbook;
not so good: as a reference manual for later use.

An alternative textbook:

Cormen, Leiserson, Rivest and Stein: *Introduction to Algorithms*
preferably the third edition, should also be available at the bookstore
excellent: to be used later as a reference manual;
not so good: quite formalistic and written in a rather dry style.

The role of proofs in algorithm design

When do we need to **prove** that an algorithm we have just designed terminates and returns a solution to the problem at hand?

The role of proofs in algorithm design

When do we need to **prove** that an algorithm we have just designed terminates and returns a solution to the problem at hand?

Only when this is not clear by common sense.

Example: MERGESORT

Merge-Sort(A,p,r) *sorting A[p..r]*

1. **if** $p < r$
2. **then** $q \leftarrow \lfloor \frac{p+r}{2} \rfloor$
3. $\textsc{Merge-Sort}(A, p, q)$
4. $\textsc{Merge-Sort}(A, q+1, r)$
5. $\textsc{Merge}(A, p, q, r)$

Example: MERGESORT

Merge-Sort(A,p,r) *sorting A[p..r]*

1. **if** $p < r$
2. **then** $q \leftarrow \lfloor \frac{p+r}{2} \rfloor$
3. $\textsc{Merge-Sort}(A, p, q)$
4. $\textsc{Merge-Sort}(A, q+1, r)$
5. $\textsc{Merge}(A, p, q, r)$

1. The depth of recursion in MERGE-SORT is $\log_2 n$.

Example: MergeSort

Merge-Sort(A,p,r) *sorting A[p..r]*

1. **if** $p < r$
2. **then** $q \leftarrow \lfloor \frac{p+r}{2} \rfloor$
3. $\textsc{Merge-Sort}(A, p, q)$
4. $\textsc{Merge-Sort}(A, q + 1, r)$
5. $\textsc{Merge}(A, p, q, r)$

1. The depth of recursion in Merge-Sort is $\log_2 n$.
2. On each level of recursion merging intermediate arrays takes $O(n)$ steps.

Example: MergeSort

Merge-Sort(A,p,r) *sorting A[p..r]*

1. **if** $p < r$
2. **then** $q \leftarrow \lfloor \frac{p+r}{2} \rfloor$
3. $\textsc{Merge-Sort}(A, p, q)$
4. $\textsc{Merge-Sort}(A, q+1, r)$
5. $\textsc{Merge}(A, p, q, r)$

1. The depth of recursion in Merge-Sort is $\log_2 n$.
2. On each level of recursion merging intermediate arrays takes $O(n)$ steps.
3. Thus, MergeSort always terminates and, in fact, it terminates in $O(n \log_2 n)$ many steps.

Example: MergeSort

Merge-Sort(A,p,r) *sorting A[p..r]*

1. **if** $p < r$
2. **then** $q \leftarrow \lfloor \frac{p+r}{2} \rfloor$
3. Merge-Sort(A, p, q)
4. Merge-Sort$(A, q + 1, r)$
5. Merge(A, p, q, r)

1. The depth of recursion in Merge-Sort is $\log_2 n$.
2. On each level of recursion merging intermediate arrays takes $O(n)$ steps.
3. Thus, MergeSort always terminates and, in fact, it terminates in $O(n \log_2 n)$ many steps.
4. Merging two sorted arrays always produces a sorted array, thus, the output of MergeSort will be a sorted array.

Example: MERGESORT

Merge-Sort(A,p,r) *sorting A[p..r]*

1. **if** $p < r$
2. **then** $q \leftarrow \lfloor \frac{p+r}{2} \rfloor$
3. MERGE-SORT(A, p, q)
4. MERGE-SORT$(A, q+1, r)$
5. MERGE(A, p, q, r)

1. The depth of recursion in MERGE-SORT is $\log_2 n$.
2. On each level of recursion merging intermediate arrays takes $O(n)$ steps.
3. Thus, MERGESORT always terminates and, in fact, it terminates in $O(n \log_2 n)$ many steps.
4. Merging two sorted arrays always produces a sorted array, thus, the output of MERGESORT will be a sorted array.
5. The above is essentially a proof by induction, but we will never bother formalizing proofs of (essentially) obvious facts.

The role of proofs in algorithm design

- However, sometimes it is **NOT** clear from a description of an algorithm that such an algorithm will not enter an infinite loop and fail to terminate;

The role of proofs in algorithm design

- However, sometimes it is **NOT** clear from a description of an algorithm that such an algorithm will not enter an infinite loop and fail to terminate;
- Sometimes it is not clear that an algorithm will not run in exponentially many steps (in the size of the input), which is essentially as bad as never terminating;

The role of proofs in algorithm design

- However, sometimes it is **NOT** clear from a description of an algorithm that such an algorithm will not enter an infinite loop and fail to terminate;
- Sometimes it is not clear that an algorithm will not run in exponentially many steps (in the size of the input), which is essentially as bad as never terminating;
- Sometimes it is not clear from a description of an algorithm why such an algorithm, after it terminates, produces a desired solution.

The role of proofs in algorithm design

- However, sometimes it is **NOT** clear from a description of an algorithm that such an algorithm will not enter an infinite loop and fail to terminate;
- Sometimes it is not clear that an algorithm will not run in exponentially many steps (in the size of the input), which is essentially as bad as never terminating;
- Sometimes it is not clear from a description of an algorithm why such an algorithm, after it terminates, produces a desired solution.
- Proofs are needed for such circumstances; thus, proofs are **NOT** academic embellishments - in lots of cases they are **the only way** to know that the algorithm will always work!

The role of proofs in algorithm design

- However, sometimes it is **NOT** clear from a description of an algorithm that such an algorithm will not enter an infinite loop and fail to terminate;
- Sometimes it is not clear that an algorithm will not run in exponentially many steps (in the size of the input), which is essentially as bad as never terminating;
- Sometimes it is not clear from a description of an algorithm why such an algorithm, after it terminates, produces a desired solution.
- Proofs are needed for such circumstances; thus, proofs are **NOT** academic embellishments - in lots of cases they are **the only way** to know that the algorithm will always work!
- For that reason we will **NEVER** prove the obvious (your CLRS textbook sometimes does just that, being too pedantic!) and will prove only what is genuinely nontrivial.

The role of proofs in algorithm design

- However, sometimes it is **NOT** clear from a description of an algorithm that such an algorithm will not enter an infinite loop and fail to terminate;
- Sometimes it is not clear that an algorithm will not run in exponentially many steps (in the size of the input), which is essentially as bad as never terminating;
- Sometimes it is not clear from a description of an algorithm why such an algorithm, after it terminates, produces a desired solution.
- Proofs are needed for such circumstances; thus, proofs are **NOT** academic embellishments - in lots of cases they are **the only way** to know that the algorithm will always work!
- For that reason we will **NEVER** prove the obvious (your CLRS textbook sometimes does just that, being too pedantic!) and will prove only what is genuinely nontrivial.
- **However, be very careful what you call trivial!!**

Role of proofs - example

Stable Matching Problem

Role of proofs - example

Stable Matching Problem

Assume that you are running a dating agency and have n men and n women as customers;

Role of proofs - example

Stable Matching Problem

Assume that you are running a dating agency and have n men and n women as customers;

They all attend a dinner party; after the party

- every man gives you a list with his ranking of all women present, **and**
- every woman gives you a list with her ranking of all men present;

Role of proofs - example

Stable Matching Problem

Assume that you are running a dating agency and have n men and n women as customers;

They all attend a dinner party; after the party

- every man gives you a list with his ranking of all women present, **and**
- every woman gives you a list with her ranking of all men present;

Design an algorithm which produces a *stable matching*:

Role of proofs - example

Stable Matching Problem

Assume that you are running a dating agency and have n men and n women as customers;

They all attend a dinner party; after the party

- every man gives you a list with his ranking of all women present, **and**
- every woman gives you a list with her ranking of all men present;

Design an algorithm which produces a *stable matching*:
a set of n pairs $p = (m, w)$ of a man m and a woman w so that
the following situation never happens:

Role of proofs - example

Stable Matching Problem

Assume that you are running a dating agency and have n men and n women as customers;

They all attend a dinner party; after the party

- every man gives you a list with his ranking of all women present, **and**
- every woman gives you a list with her ranking of all men present;

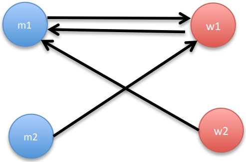
Design an algorithm which produces a *stable matching*:
a set of n pairs $p = (m, w)$ of a man m and a woman w so that the following situation never happens:

for two pairs $p = (m, w)$ and $p' = (m', w')$:

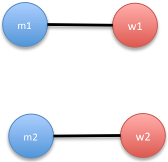
- man m prefers woman w' to woman w, **and**
- woman w' prefers man m to man m'.

Stable Matching Problem

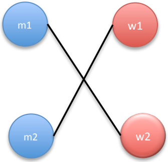
Preferences:



<u>Case1:</u>
stable matching



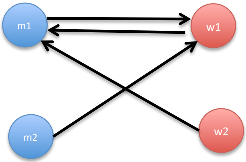
not stable



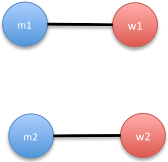
Stable Matching Problem

Case1:

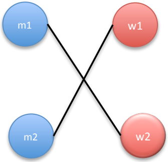
Preferences:



stable matching

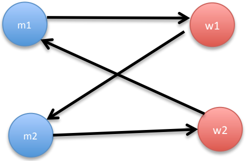


not stable

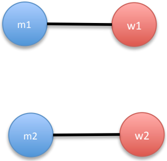


Case2:

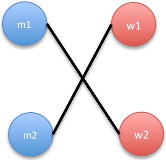
Preferences:



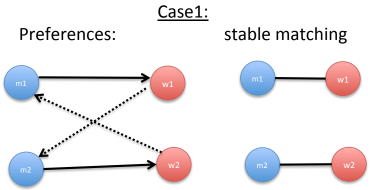
stable matching



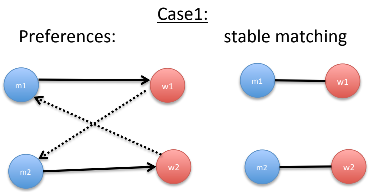
also stable!

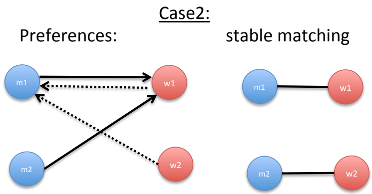


Is there always a stable matching for any preferences of two pairs?



Is there always a stable matching for any preferences of two pairs?





Stable Matching Problem: Gale - Shapley algorithm (invented to pair newly graduated physicians with US hospitals for residency training)

Question 1: Is it true that for every possible collection of n lists of preferences provided by all men, and n lists of preferences provided by all women, a stable matching exists?

Stable Matching Problem: Gale - Shapley algorithm (invented to pair newly graduated physicians with US hospitals for residency training)

Question 1: Is it true that for every possible collection of n lists of preferences provided by all men, and n lists of preferences provided by all women, a stable matching exists?

Answer: **YES**, but this is **NOT** obvious!

Stable Matching Problem: Gale - Shapley algorithm (invented to pair newly graduated physicians with US hospitals for residency training)

Question 1: Is it true that for every possible collection of n lists of preferences provided by all men, and n lists of preferences provided by all women, a stable matching exists?

Answer: **YES**, but this is **NOT** obvious!

Question 2: Given n men and n women, how many ways are there to match them, i.e., just to form n couples?

Stable Matching Problem: Gale - Shapley algorithm (invented to pair newly graduated physicians with US hospitals for residency training)

Question 1: Is it true that for every possible collection of n lists of preferences provided by all men, and n lists of preferences provided by all women, a stable matching exists?

Answer: **YES**, but this is **NOT** obvious!

Question 2: Given n men and n women, how many ways are there to match them, i.e., just to form n couples?

Answer: $n!$

Stable Matching Problem: Gale - Shapley algorithm (invented to pair newly graduated physicians with US hospitals for residency training)

Question 1: Is it true that for every possible collection of n lists of preferences provided by all men, and n lists of preferences provided by all women, a stable matching exists?

Answer: **YES**, but this is **NOT** obvious!

Question 2: Given n men and n women, how many ways are there to match them, i.e., just to form n couples?

Answer: $n! \approx (n/e)^n$ - more than exponentially many in n ($e \approx 2.71$);

Stable Matching Problem: Gale - Shapley algorithm (invented to pair newly graduated physicians with US hospitals for residency training)

Question 1: Is it true that for every possible collection of n lists of preferences provided by all men, and n lists of preferences provided by all women, a stable matching exists?

Answer: **YES**, but this is **NOT** obvious!

Question 2: Given n men and n women, how many ways are there to match them, i.e., just to form n couples?

Answer: $n! \approx (n/e)^n$ - more than exponentially many in n ($e \approx 2.71$);

Can we find a stable matching in a reasonable amount of time??

Stable Matching Problem: Gale - Shapley algorithm (invented to pair newly graduated physicians with US hospitals for residency training)

Question 1: Is it true that for every possible collection of n lists of preferences provided by all men, and n lists of preferences provided by all women, a stable matching exists?

Answer: **YES**, but this is **NOT** obvious!

Question 2: Given n men and n women, how many ways are there to match them, i.e., just to form n couples?

Answer: $n! \approx (n/e)^n$ - more than exponentially many in n ($e \approx 2.71$);

Can we find a stable matching in a reasonable amount of time??

Answer: **YES**, using the **Gale - Shapley algorithm**.

Stable Matching Problem: Gale - Shapley algorithm

Stable Matching Problem: Gale - Shapley algorithm

- Produces pairs in stages, with possible revisions;

Stable Matching Problem: Gale - Shapley algorithm

- Produces pairs in stages, with possible revisions;
- A man who has not been paired with a woman will be called *free*.

Stable Matching Problem: Gale - Shapley algorithm

- Produces pairs in stages, with possible revisions;
- A man who has not been paired with a woman will be called *free*.
- Men will be proposing to women.Women will decide if they accept a proposal or not.

Stable Matching Problem: Gale - Shapley algorithm

- Produces pairs in stages, with possible revisions;
- A man who has not been paired with a woman will be called *free*.
- Men will be proposing to women.Women will decide if they accept a proposal or not.
- Start with all men free;

Stable Matching Problem: Gale - Shapley algorithm

- Produces pairs in stages, with possible revisions;
- A man who has not been paired with a woman will be called *free*.
- Men will be proposing to women.Women will decide if they accept a proposal or not.
- Start with all men free;

While there exists a free man who has not proposed to all women

Stable Matching Problem: Gale - Shapley algorithm

- Produces pairs in stages, with possible revisions;
- A man who has not been paired with a woman will be called *free*.
- Men will be proposing to women.Women will decide if they accept a proposal or not.
- Start with all men free;

While there exists a free man who has not proposed to all women
    pick such a free man m and have him propose to the highest
    ranking woman w on his list to whom he has not proposed yet;

Stable Matching Problem: Gale - Shapley algorithm

- Produces pairs in stages, with possible revisions;
- A man who has not been paired with a woman will be called *free*.
- Men will be proposing to women.Women will decide if they accept a proposal or not.
- Start with all men free;

While there exists a free man who has not proposed to all women
  pick such a free man m and have him propose to the highest
  ranking woman w on his list to whom he has not proposed yet;
  If no one has proposed to w yet

Stable Matching Problem: Gale - Shapley algorithm

- Produces pairs in stages, with possible revisions;
- A man who has not been paired with a woman will be called *free*.
- Men will be proposing to women.Women will decide if they accept a proposal or not.
- Start with all men free;

While there exists a free man who has not proposed to all women
    pick such a free man m and have him propose to the highest
    ranking woman w on his list to whom he has not proposed yet;
    If   no one has proposed to w yet
        she always accepts and a pair $p = (m, w)$ is formed;

Stable Matching Problem: Gale - Shapley algorithm

- Produces pairs in stages, with possible revisions;
- A man who has not been paired with a woman will be called *free*.
- Men will be proposing to women.Women will decide if they accept a proposal or not.
- Start with all men free;

While there exists a free man who has not proposed to all women
    pick such a free man m and have him propose to the highest ranking woman w on his list to whom he has not proposed yet;
    If no one has proposed to w yet
        she always accepts and a pair $p = (m, w)$ is formed;
    Else she is already in a pair $p' = (m', w)$;

Stable Matching Problem: Gale - Shapley algorithm

- Produces pairs in stages, with possible revisions;
- A man who has not been paired with a woman will be called *free*.
- Men will be proposing to women.Women will decide if they accept a proposal or not.
- Start with all men free;

While there exists a free man who has not proposed to all women
    pick such a free man m and have him propose to the highest ranking woman w on his list to whom he has not proposed yet;
    If no one has proposed to w yet
        she always accepts and a pair $p = (m, w)$ is formed;
    Else she is already in a pair $p' = (m', w)$;
        If m is higher on her preference list than m'

Stable Matching Problem: Gale - Shapley algorithm

- Produces pairs in stages, with possible revisions;
- A man who has not been paired with a woman will be called *free*.
- Men will be proposing to women.Women will decide if they accept a proposal or not.
- Start with all men free;

While there exists a free man who has not proposed to all women
    pick such a free man m and have him propose to the highest ranking woman w on his list to whom he has not proposed yet;
    If no one has proposed to w yet
        she always accepts and a pair $p = (m, w)$ is formed;
    Else she is already in a pair $p' = (m', w)$;
        If m is higher on her preference list than m'
            the pair $p' = (m', w)$ is deleted;

Stable Matching Problem: Gale - Shapley algorithm

- Produces pairs in stages, with possible revisions;
- A man who has not been paired with a woman will be called *free*.
- Men will be proposing to women. Women will decide if they accept a proposal or not.
- Start with all men free;

While there exists a free man who has not proposed to all women
    pick such a free man m and have him propose to the highest ranking woman w on his list to whom he has not proposed yet;
    If no one has proposed to w yet
        she always accepts and a pair $p = (m, w)$ is formed;
    Else she is already in a pair $p' = (m', w)$;
        If m is higher on her preference list than m'
            the pair $p' = (m', w)$ is deleted;
            m' becomes a free man;

Stable Matching Problem: Gale - Shapley algorithm

- Produces pairs in stages, with possible revisions;
- A man who has not been paired with a woman will be called *free*.
- Men will be proposing to women. Women will decide if they accept a proposal or not.
- Start with all men free;

While there exists a free man who has not proposed to all women
    pick such a free man m and have him propose to the highest
    ranking woman w on his list to whom he has not proposed yet;
    If no one has proposed to w yet
        she always accepts and a pair $p = (m, w)$ is formed;
    Else she is already in a pair $p' = (m', w)$;
        If m is higher on her preference list than m'
            the pair $p' = (m', w)$ is deleted;
            m' becomes a free man;
            a new pair $p = (m, w)$ is formed;

Stable Matching Problem: Gale - Shapley algorithm

- Produces pairs in stages, with possible revisions;
- A man who has not been paired with a woman will be called *free*.
- Men will be proposing to women. Women will decide if they accept a proposal or not.
- Start with all men free;

While there exists a free man who has not proposed to all women
    pick such a free man m and have him propose to the highest
    ranking woman w on his list to whom he has not proposed yet;
    If no one has proposed to w yet
        she always accepts and a pair $p = (m, w)$ is formed;
    Else she is already in a pair $p' = (m', w)$;
        If m is higher on her preference list than m'
            the pair $p' = (m', w)$ is deleted;
            m' becomes a free man;
            a new pair $p = (m, w)$ is formed;
        Else m is lower on her preference list than m';
            the proposal is rejected and m remains free.

Claim 1: Algorithm terminates after $\leq n^2$ rounds of the *While* loop

Stable Matching Problem: Gale - Shapley algorithm

Claim 1: Algorithm terminates after $\leq n^2$ rounds of the *While* loop
Proof:

- In every round of the *While* loop one man proposes to one woman;

Stable Matching Problem: Gale - Shapley algorithm

Claim 1: Algorithm terminates after $\leq n^2$ rounds of the *While* loop

Proof:

- In every round of the *While* loop one man proposes to one woman;
- every man can propose to a woman at most once;

Stable Matching Problem: Gale - Shapley algorithm

Claim 1: Algorithm terminates after $\leq n^2$ rounds of the *While* loop
Proof:

- In every round of the *While* loop one man proposes to one woman;
- every man can propose to a woman at most once;
- thus, every man can make at most n proposals;

Stable Matching Problem: Gale - Shapley algorithm

Claim 1: Algorithm terminates after $\leq n^2$ rounds of the *While* loop
Proof:

- In every round of the *While* loop one man proposes to one woman;
- every man can propose to a woman at most once;
- thus, every man can make at most n proposals;
- there are n men, so in total they can make $\leq n^2$ proposals

Stable Matching Problem: Gale - Shapley algorithm

Claim 1: Algorithm terminates after $\leq n^2$ rounds of the *While* loop

Proof:

- In every round of the *While* loop one man proposes to one woman;
- every man can propose to a woman at most once;
- thus, every man can make at most n proposals;
- there are n men, so in total they can make $\leq n^2$ proposals

Thus the *While* loop can be executed no more than n^2 many times.

Stable Matching Problem: Gale - Shapley algorithm

Claim 1: Algorithm terminates after $\leq n^2$ rounds of the *While* loop
Proof:

- In every round of the *While* loop one man proposes to one woman;
- every man can propose to a woman at most once;
- thus, every man can make at most n proposals;
- there are n men, so in total they can make $\leq n^2$ proposals

Thus the *While* loop can be executed no more than n^2 many times.

Claim 2: Algorithm produces a matching, i.e., every man is eventually paired with a woman (and thus also every woman is paired to a man)

Stable Matching Problem: Gale - Shapley algorithm

Claim 1: Algorithm terminates after $\leq n^2$ rounds of the *While* loop
Proof:

- In every round of the *While* loop one man proposes to one woman;
- every man can propose to a woman at most once;
- thus, every man can make at most n proposals;
- there are n men, so in total they can make $\leq n^2$ proposals

Thus the *While* loop can be executed no more than n^2 many times.

Claim 2: Algorithm produces a matching, i.e., every man is eventually paired with a woman (and thus also every woman is paired to a man)
Proof:

Claim 1: Algorithm terminates after $\leq n^2$ rounds of the *While* loop
Proof:

- In every round of the *While* loop one man proposes to one woman;
- every man can propose to a woman at most once;
- thus, every man can make at most n proposals;
- there are n men, so in total they can make $\leq n^2$ proposals

Thus the *While* loop can be executed no more than n^2 many times.

Claim 2: Algorithm produces a matching, i.e., every man is eventually paired with a woman (and thus also every woman is paired to a man)
Proof:

- Assume that the while *While* loop has terminated, but m is still free.

Stable Matching Problem: Gale - Shapley algorithm

Claim 1: Algorithm terminates after $\leq n^2$ rounds of the *While* loop
Proof:

- In every round of the *While* loop one man proposes to one woman;
- every man can propose to a woman at most once;
- thus, every man can make at most n proposals;
- there are n men, so in total they can make $\leq n^2$ proposals

Thus the *While* loop can be executed no more than n^2 many times.

Claim 2: Algorithm produces a matching, i.e., every man is eventually paired with a woman (and thus also every woman is paired to a man)
Proof:

- Assume that the while *While* loop has terminated, but m is still free.
- This means that m has already proposed to every woman.

Stable Matching Problem: Gale - Shapley algorithm

Claim 1: Algorithm terminates after $\leq n^2$ rounds of the *While* loop
Proof:

- In every round of the *While* loop one man proposes to one woman;
- every man can propose to a woman at most once;
- thus, every man can make at most n proposals;
- there are n men, so in total they can make $\leq n^2$ proposals

Thus the *While* loop can be executed no more than n^2 many times.

Claim 2: Algorithm produces a matching, i.e., every man is eventually paired with a woman (and thus also every woman is paired to a man)
Proof:

- Assume that the while *While* loop has terminated, but m is still free.
- This means that m has already proposed to every woman.
- Thus, every woman is paired with a man, because a woman is not paired with anyone only if no one has made a proposal to her.

Stable Matching Problem: Gale - Shapley algorithm

Claim 1: Algorithm terminates after $\leq n^2$ rounds of the *While* loop
Proof:

- In every round of the *While* loop one man proposes to one woman;
- every man can propose to a woman at most once;
- thus, every man can make at most n proposals;
- there are n men, so in total they can make $\leq n^2$ proposals

Thus the *While* loop can be executed no more than n^2 many times.

Claim 2: Algorithm produces a matching, i.e., every man is eventually paired with a woman (and thus also every woman is paired to a man)
Proof:

- Assume that the while *While* loop has terminated, but m is still free.
- This means that m has already proposed to every woman.
- Thus, every woman is paired with a man, because a woman is not paired with anyone only if no one has made a proposal to her.
- But this would mean that n women are paired with all of n men so m cannot be free.

Stable Matching Problem: Gale - Shapley algorithm

Claim 1: Algorithm terminates after $\leq n^2$ rounds of the *While* loop
Proof:

- In every round of the *While* loop one man proposes to one woman;
- every man can propose to a woman at most once;
- thus, every man can make at most n proposals;
- there are n men, so in total they can make $\leq n^2$ proposals

Thus the *While* loop can be executed no more than n^2 many times.

Claim 2: Algorithm produces a matching, i.e., every man is eventually paired with a woman (and thus also every woman is paired to a man)
Proof:

- Assume that the while *While* loop has terminated, but m is still free.
- This means that m has already proposed to every woman.
- Thus, every woman is paired with a man, because a woman is not paired with anyone only if no one has made a proposal to her.
- But this would mean that n women are paired with all of n men so m cannot be free. **Contradiction!**

Claim 3: The matching produced by the algorithm is stable.

Claim 3: The matching produced by the algorithm is stable.

Proof:

Claim 3: The matching produced by the algorithm is stable.

Proof: Note that during the *While* loop:

- a woman is paired with men of increasing ranks on her list;

Stable Matching Problem: Gale - Shapley algorithm

Claim 3: The matching produced by the algorithm is stable.

Proof: Note that during the *While* loop:

- a woman is paired with men of increasing ranks on her list;
- a man is paired with women of decreasing ranks on his list.

Stable Matching Problem: Gale - Shapley algorithm

Claim 3: The matching produced by the algorithm is stable.
Proof: Note that during the *While* loop:

- a woman is paired with men of increasing ranks on her list;
- a man is paired with women of decreasing ranks on his list.

Assume now the opposite, that the matching is not stable;

Claim 3: The matching produced by the algorithm is stable.

Proof: Note that during the *While* loop:

- a woman is paired with men of increasing ranks on her list;
- a man is paired with women of decreasing ranks on his list.

Assume now the opposite, that the matching is not stable;
thus, there are two pairs $p = (m, w)$ and $p' = (m', w')$ such that:

Stable Matching Problem: Gale - Shapley algorithm

Claim 3: The matching produced by the algorithm is stable.
Proof: Note that during the *While* loop:

- a woman is paired with men of increasing ranks on her list;
- a man is paired with women of decreasing ranks on his list.

Assume now the opposite, that the matching is not stable;
thus, there are two pairs $p = (m, w)$ and $p' = (m', w')$ such that:

$$m \text{ prefers } w' \text{ over } w;$$
$$w' \text{ prefers } m \text{ over } m'.$$

Stable Matching Problem: Gale - Shapley algorithm

Claim 3: The matching produced by the algorithm is stable.
Proof: Note that during the *While* loop:

- a woman is paired with men of increasing ranks on her list;
- a man is paired with women of decreasing ranks on his list.

Assume now the opposite, that the matching is not stable;
thus, there are two pairs $p = (m, w)$ and $p' = (m', w')$ such that:

$$m \text{ prefers } w' \text{ over } w;$$
$$w' \text{ prefers } m \text{ over } m'.$$

- Since m prefers w' over w, he must have proposed to w' before proposing to w;

Claim 3: The matching produced by the algorithm is stable.
Proof: Note that during the *While* loop:

- a woman is paired with men of increasing ranks on her list;
- a man is paired with women of decreasing ranks on his list.

Assume now the opposite, that the matching is not stable;
thus, there are two pairs $p = (m, w)$ and $p' = (m', w')$ such that:

$$m \text{ prefers } w' \text{ over } w;$$
$$w' \text{ prefers } m \text{ over } m'.$$

- Since m prefers w' over w, he must have proposed to w' before proposing to w;
- Since he is paired with w, woman w' must have either:

Claim 3: The matching produced by the algorithm is stable.

Proof: Note that during the *While* loop:

- a woman is paired with men of increasing ranks on her list;
- a man is paired with women of decreasing ranks on his list.

Assume now the opposite, that the matching is not stable;
thus, there are two pairs $p = (m, w)$ and $p' = (m', w')$ such that:

$$m \text{ prefers } w' \text{ over } w;$$
$$w' \text{ prefers } m \text{ over } m'.$$

- Since m prefers w' over w, he must have proposed to w' before proposing to w;
- Since he is paired with w, woman w' must have either:
  - rejected him because she was already with someone whom she prefers, or

Stable Matching Problem: Gale - Shapley algorithm

Claim 3: The matching produced by the algorithm is stable.
Proof: Note that during the *While* loop:

- a woman is paired with men of increasing ranks on her list;
- a man is paired with women of decreasing ranks on his list.

Assume now the opposite, that the matching is not stable;
thus, there are two pairs $p = (m, w)$ and $p' = (m', w')$ such that:

$$m \text{ prefers } w' \text{ over } w;$$
$$w' \text{ prefers } m \text{ over } m'.$$

- Since m prefers w' over w, he must have proposed to w' before proposing to w;
- Since he is paired with w, woman w' must have either:
  - rejected him because she was already with someone whom she prefers, or
  - dropped him later after a proposal from someone whom she prefers;

Stable Matching Problem: Gale - Shapley algorithm

Claim 3: The matching produced by the algorithm is stable.

Proof: Note that during the *While* loop:

- a woman is paired with men of increasing ranks on her list;
- a man is paired with women of decreasing ranks on his list.

Assume now the opposite, that the matching is not stable; thus, there are two pairs $p = (m, w)$ and $p' = (m', w')$ such that:

$$m \text{ prefers } w' \text{ over } w;$$
$$w' \text{ prefers } m \text{ over } m'.$$

- Since m prefers w' over w, he must have proposed to w' before proposing to w;
- Since he is paired with w, woman w' must have either:
  - rejected him because she was already with someone whom she prefers, or
  - dropped him later after a proposal from someone whom she prefers;
- In both cases she would now be with m' whom she prefers over m.

Stable Matching Problem: Gale - Shapley algorithm

Claim 3: The matching produced by the algorithm is stable.
Proof: Note that during the *While* loop:

- a woman is paired with men of increasing ranks on her list;
- a man is paired with women of decreasing ranks on his list.

Assume now the opposite, that the matching is not stable;
thus, there are two pairs $p = (m, w)$ and $p' = (m', w')$ such that:

$$m \text{ prefers } w' \text{ over } w;$$
$$w' \text{ prefers } m \text{ over } m'.$$

- Since m prefers w' over w, he must have proposed to w' before proposing to w;
- Since he is paired with w, woman w' must have either:
  - rejected him because she was already with someone whom she prefers, or
  - dropped him later after a proposal from someone whom she prefers;
- In both cases she would now be with m' whom she prefers over m.
- **Contradiction!**

Puzzles!!!

Puzzles!!!

Why puzzles? It is a fun way to practice problem solving!

Puzzles!!!

Why puzzles? It is a fun way to practice problem solving!

Problem : Tom and his wife Mary went to a party where nine more couples were present.

Puzzles!!!

Why puzzles? It is a fun way to practice problem solving!

Problem : Tom and his wife Mary went to a party where nine more couples were present.

- Not every one knew everyone else, so people who did not know each other introduced themselves and shook hands.

Puzzles!!!

Why puzzles? It is a fun way to practice problem solving!

Problem : Tom and his wife Mary went to a party where nine more couples were present.

- Not every one knew everyone else, so people who did not know each other introduced themselves and shook hands.
- People who knew each other from before did not shake hands.

Puzzles!!!

Why puzzles? It is a fun way to practice problem solving!

Problem : Tom and his wife Mary went to a party where nine more couples were present.

- Not every one knew everyone else, so people who did not know each other introduced themselves and shook hands.
- People who knew each other from before did not shake hands.
- Later that evening Tom got bored, so he walked around and asked all other guests (including his wife) how many hands they had shaken that evening, and got 19 different answers.

Puzzles!!!

Why puzzles? It is a fun way to practice problem solving!

Problem : Tom and his wife Mary went to a party where nine more couples were present.

- Not every one knew everyone else, so people who did not know each other introduced themselves and shook hands.
- People who knew each other from before did not shake hands.
- Later that evening Tom got bored, so he walked around and asked all other guests (including his wife) how many hands they had shaken that evening, and got 19 different answers.
- How many hands did Mary shake?

Puzzles!!!

Why puzzles? It is a fun way to practice problem solving!

Problem : Tom and his wife Mary went to a party where nine more couples were present.

- Not every one knew everyone else, so people who did not know each other introduced themselves and shook hands.
- People who knew each other from before did not shake hands.
- Later that evening Tom got bored, so he walked around and asked all other guests (including his wife) how many hands they had shaken that evening, and got 19 different answers.
- How many hands did Mary shake?
- How many hands did Tom shake?

Puzzles!!!

Problem : We are given 27 coins of the same denomination; we know that one of them is counterfeit and that it is lighter than the others. Find the counterfeit coin by weighing coins on a pan balance only three times.

Puzzles!!!

Problem : We are given 27 coins of the same denomination; we know that one of them is counterfeit and that it is lighter than the others. Find the counterfeit coin by weighing coins on a pan balance only three times.

Note: this method is called "divide-and-conquer".

Puzzles!!!

Problem : We have nine coins and three of them are heavier than the remaining six. Can you find the heavier coins by weighing coins on a pan balance only four times?

Puzzles!!!

Problem : We have nine coins and three of them are heavier than the remaining six. Can you find the heavier coins by weighing coins on a pan balance only four times?

- How many outcomes of 4 weighings are there in total?

Puzzles!!!

Problem : We have nine coins and three of them are heavier than the remaining six. Can you find the heavier coins by weighing coins on a pan balance only four times?

- How many outcomes of 4 weighings are there in total?

$$3^4 = 81$$

Puzzles!!!

Problem : We have nine coins and three of them are heavier than the remaining six. Can you find the heavier coins by weighing coins on a pan balance only four times?

- How many outcomes of 4 weighings are there in total?

$$3^4 = 81$$

- How many ways are there to hide three heavier coins among six good coins?

Puzzles!!!

Problem : We have nine coins and three of them are heavier than the remaining six. Can you find the heavier coins by weighing coins on a pan balance only four times?

- How many outcomes of 4 weighings are there in total?

$$3^4 = 81$$

- How many ways are there to hide three heavier coins among six good coins?

$$\binom{9}{3} = \frac{9!}{3!6!} = \frac{7 \times 8 \times 9}{3!} = 84$$

Puzzles!!!

Problem : We have nine coins and three of them are heavier than the remaining six. Can you find the heavier coins by weighing coins on a pan balance only four times?

- How many outcomes of 4 weighings are there in total?

$$3^4 = 81$$

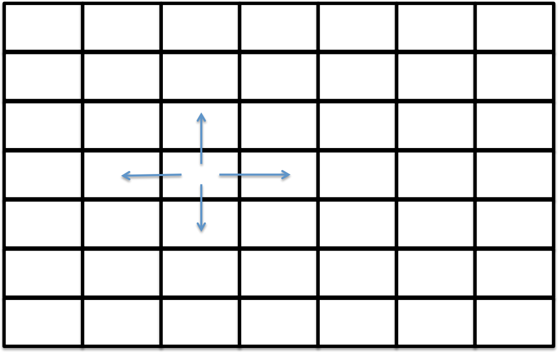
- How many ways are there to hide three heavier coins among six good coins?

$$\binom{9}{3} = \frac{9!}{3!6!} = \frac{7 \times 8 \times 9}{3!} = 84$$

- More ways to hide than the number of all possible weighting outcomes! Thus, it is impossible to do it!

Puzzles!!!

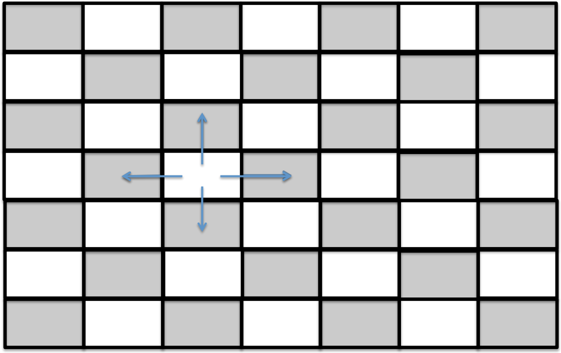
Problem: Consider a block of 7 X 7 houses:



The inhabitant of each house thinks that all four houses around him (to the left, right, top and bottom) are nicer than his house and would like to move to any of the four. Can you move the inhabitants around to make them all happier?

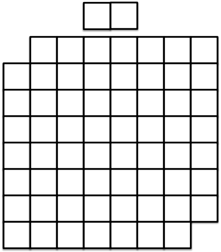
Puzzles!!!

Hint:



Puzzles!!!

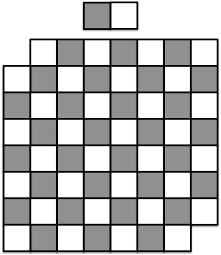
Problem: Consider an 8×8 board with two diagonal squares missing, and an 1×2 domino:



Can you cover the entire board with 31 such dominoes?

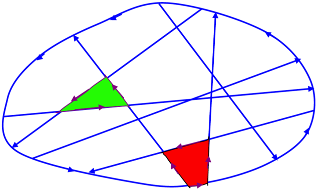
Puzzles!!!

Hint:



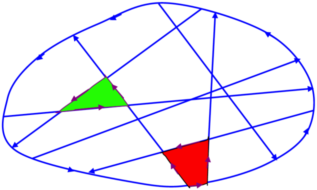
Puzzles!!!

Problem: In Elbonia all cities have a circular one-way highway around the city; see the map. All streets in the cities are one-way, and they all start and end on the circular highway (see the map).



Puzzles!!!

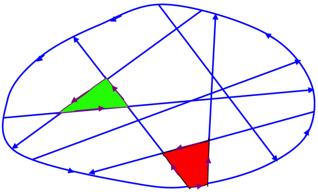
Problem: In Elbonia all cities have a circular one-way highway around the city; see the map. All streets in the cities are one-way, and they all start and end on the circular highway (see the map).



- A block is a part of the city that is not intersected by any street.

Puzzles!!!

Problem: In Elbonia all cities have a circular one-way highway around the city; see the map. All streets in the cities are one-way, and they all start and end on the circular highway (see the map).



- A block is a part of the city that is not intersected by any street.
- Design an algorithm that, given a map of a city, finds a block (just one such block, not all such blocks) that can be circumnavigated while respecting all one-way signs.

(for example, the green block has such property, but not the red one)

Basics revisited: how do we add two numbers?

Basics revisited: how do we add two numbers?

```
 C C C C C     carry
   X X X X X    first integer
+  X X X X X    second integer
  -----------
 X X X X X X    result
```

Basics revisited: how do we add two numbers?

```
 C C C C C     carry
   X X X X X    first integer
+  X X X X X    second integer
  -----------
 X X X X X X    result
```

- adding 3 bits can be done in constant time;

Basics revisited: how do we add two numbers?

```
 C C C C C      carry
   X X X X X     first integer
+  X X X X X     second integer
  -----------
 X X X X X X     result
```

- adding 3 bits can be done in constant time;
- the whole algorithm runs in linear time i.e., $O(n)$ many steps.

Basics revisited: how do we add two numbers?

```
 C C C C C     carry
   X X X X X    first integer
+  X X X X X    second integer
  -----------
 X X X X X X    result
```

- adding 3 bits can be done in constant time;
- the whole algorithm runs in linear time i.e., $O(n)$ many steps.

can we do it faster than in linear time?

Basics revisited: how do we add two numbers?

```
 C C C C C      carry
   X X X X X     first integer
+  X X X X X     second integer
  -----------
 X X X X X X     result
```

- adding 3 bits can be done in constant time;
- the whole algorithm runs in linear time i.e., $O(n)$ many steps.

can we do it faster than in linear time?

- no, because we have to read every bit of the input

Basics revisited: how do we add two numbers?

```
 C C C C C     carry
   X X X X X    first integer
+  X X X X X    second integer
  -----------
 X X X X X X    result
```

- adding 3 bits can be done in constant time;
- the whole algorithm runs in linear time i.e., $O(n)$ many steps.

can we do it faster than in linear time?

- no, because we have to read every bit of the input
- no asymptotically faster algorithm

Basics revisited: how do we multiply two numbers?

```
        X X X X  <- first input integer
      * X X X X  <- second input integer
        -------
        X X X X  \
      X X X X     \ O(n^2) intermediate operations:
    X X X X       / O(n^2) elementary  multiplications
  X X X X        /    + O(n^2) elementary additions
---------------
X X X X X X X X  <-  result of length 2n
```

Basics revisited: how do we multiply two numbers?

```
        X X X X  <- first input integer
      * X X X X  <- second input integer
        -------
        X X X X  \
      X X X X     \ O(n^2) intermediate operations:
    X X X X       / O(n^2) elementary  multiplications
  X X X X        /    + O(n^2) elementary additions
---------------
X X X X X X X X  <-  result of length 2n
```

- We assume that two `X`'s can be multiplied in $O(1)$. time (each `X` could be a bit or a digit in some other base).

Basics revisited: how do we multiply two numbers?

```
          X X X X  <- first input integer
        * X X X X  <- second input integer
          -------
          X X X X  \
        X X X X     \ O(n^2) intermediate operations:
      X X X X       / O(n^2) elementary  multiplications
    X X X X        /    + O(n^2) elementary additions
---------------
X X X X X X X X  <-  result of length 2n
```

- We assume that two `X`'s can be multiplied in $O(1)$. time (each `X` could be a bit or a digit in some other base).
- Thus the above procedure runs in time $O(n^2)$.

Basics revisited: how do we multiply two numbers?

```
      X X X X  <- first input integer
    * X X X X  <- second input integer
      -------
      X X X X  \
    X X X X     \ O(n^2) intermediate operations:
  X X X X       / O(n^2) elementary  multiplications
X X X X        /    + O(n^2) elementary additions
---------------
X X X X X X X X  <-  result of length 2n
```

- We assume that two `X`'s can be multiplied in $O(1)$. time (each `X` could be a bit or a digit in some other base).
- Thus the above procedure runs in time $O(n^2)$.
- Can we do it in **LINEAR** time, like addition?

Basics revisited: how do we multiply two numbers?

```
        X X X X  <- first input integer
      * X X X X  <- second input integer
        -------
        X X X X  \
      X X X X     \ O(n^2) intermediate operations:
    X X X X       / O(n^2) elementary  multiplications
  X X X X        /    + O(n^2) elementary additions
---------------
X X X X X X X X  <-  result of length 2n
```

- We assume that two `X`'s can be multiplied in $O(1)$. time (each `X` could be a bit or a digit in some other base).
- Thus the above procedure runs in time $O(n^2)$.
- Can we do it in **LINEAR** time, like addition?
- **No one knows!**

Basics revisited: how do we multiply two numbers?

```
        X X X X  <- first input integer
      * X X X X  <- second input integer
        -------
        X X X X  \
      X X X X     \ O(n^2) intermediate operations:
    X X X X       / O(n^2) elementary  multiplications
  X X X X        /    + O(n^2) elementary additions
---------------
X X X X X X X X  <-  result of length 2n
```

- We assume that two `X`'s can be multiplied in $O(1)$. time (each `X` could be a bit or a digit in some other base).
- Thus the above procedure runs in time $O(n^2)$.
- Can we do it in **LINEAR** time, like addition?
- **No one knows!**
- "Simple" problems can actually turn out to be difficult!

Can we do multiplication faster than $O(n^2)$?

Can we do multiplication faster than $O(n^2)$?

Let us try a divide-and-conquer algorithm:

Can we do multiplication faster than $O(n^2)$?

Let us try a divide-and-conquer algorithm:
take our two input numbers A and B, and split them into two halves:

Can we do multiplication faster than $O(n^2)$?

Let us try a divide-and-conquer algorithm:
take our two input numbers A and B, and split them into two halves:

$$A = A_1 2^{\frac{n}{2}} + A_0 \qquad \underbrace{XX\dots X}_{\frac{n}{2}}\underbrace{XX\dots X}_{\frac{n}{2}}$$

$$B = B_1 2^{\frac{n}{2}} + B_0$$

Can we do multiplication faster than $O(n^2)$?

Let us try a divide-and-conquer algorithm:
take our two input numbers A and B, and split them into two halves:

$$A = A_1 2^{\frac{n}{2}} + A_0 \qquad \underbrace{XX\ldots X}_{\frac{n}{2}}\underbrace{XX\ldots X}_{\frac{n}{2}}$$

$$B = B_1 2^{\frac{n}{2}} + B_0$$

- $A_0,\ B_0$ - the least significant bits; $A_1,\ B_1$ the most significant bits.

Can we do multiplication faster than $O(n^2)$?

Let us try a divide-and-conquer algorithm:
take our two input numbers A and B, and split them into two halves:

$$A = A_1 2^{\frac{n}{2}} + A_0 \qquad \underbrace{XX \dots X}_{\frac{n}{2}}\underbrace{XX \dots X}_{\frac{n}{2}}$$

$$B = B_1 2^{\frac{n}{2}} + B_0$$

- $A_0, \ B_0$ - the least significant bits; $A_1, \ B_1$ the most significant bits.
- AB can now be calculated as follows:

$$AB = A_1 B_1 2^n + (A_1 B_0 + B_1 A_0) 2^{\frac{n}{2}} + A_0 B_0 \tag{1}$$

Can we do multiplication faster than $O(n^2)$?

Let us try a divide-and-conquer algorithm:
take our two input numbers A and B, and split them into two halves:

$$A = A_1 2^{\frac{n}{2}} + A_0 \qquad \underbrace{XX\ldots X}_{\frac{n}{2}}\underbrace{XX\ldots X}_{\frac{n}{2}}$$

$$B = B_1 2^{\frac{n}{2}} + B_0$$

- $A_0,\ B_0$ - the least significant bits; $A_1,\ B_1$ the most significant bits.
- AB can now be calculated as follows:

$$AB = A_1 B_1 2^n + (A_1 B_0 + B_1 A_0) 2^{\frac{n}{2}} + A_0 B_0 \tag{1}$$

What we mean is that the product AB can be calculated recursively by the following program:

function $\textsc{Mult}(A, B)$
  if $|A| = |B| = 1$ **then return** AB
  else
    $A_1 \leftarrow \text{MoreSignificantPart}(A)$;
    $A_0 \leftarrow \text{LessSignificantPart}(A)$;
    $B_1 \leftarrow \text{MoreSignificantPart}(B)$;
    $B_0 \leftarrow \text{LessSignificantPart}(B)$;
    $X \leftarrow \textsc{Mult}(\mathrm{A}_0, \mathrm{B}_0)$;
    $Y \leftarrow \textsc{Mult}(\mathrm{A}_0, \mathrm{B}_1)$;
    $Z \leftarrow \textsc{Mult}(\mathrm{A}_1, \mathrm{B}_0)$;
    $W \leftarrow \textsc{Mult}(\mathrm{A}_1, \mathrm{B}_1)$;
    return $W\,2^n + (Y + Z)\,2^{n/2} + X$
  end if
end function

How many steps does this algorithm take?

How many steps does this algorithm take?

Each multiplication of two n digit numbers is replaced by four multiplications of $n/2$ digit numbers: A_1B_1, A_1B_0, B_1A_0, A_0B_0,

How many steps does this algorithm take?

Each multiplication of two n digit numbers is replaced by four multiplications of $n/2$ digit numbers: A_1B_1, A_1B_0, B_1A_0, A_0B_0, plus we have a **linear** overhead to shift and add:

How many steps does this algorithm take?

Each multiplication of two n digit numbers is replaced by four multiplications of $n/2$ digit numbers: A_1B_1, A_1B_0, B_1A_0, A_0B_0, plus we have a **linear** overhead to shift and add:

$$T(n) = 4T\left(\frac{n}{2}\right) + c\,n \tag{2}$$

Claim: if $T(n)$ satisfies

$$T(n) = 4T\left(\frac{n}{2}\right) + c\,n \tag{3}$$

then

$$T(n) = n^2(c+1) - c\,n$$

Can we do multiplication faster than $O(n^2)$?

Claim: if $T(n)$ satisfies

$$T(n) = 4T\left(\frac{n}{2}\right) + c\,n \tag{3}$$

then

$$T(n) = n^2(c+1) - c\,n$$

Proof: By "fast" induction. We assume it is true for $\lfloor n/2 \rfloor$:

Can we do multiplication faster than $O(n^2)$?

Claim: if $T(n)$ satisfies

$$T(n) = 4T\left(\frac{n}{2}\right) + c\,n \tag{3}$$

then

$$T(n) = n^2(c+1) - c\,n$$

Proof: By "fast" induction. We assume it is true for $\lfloor n/2 \rfloor$:

$$T\left(\frac{n}{2}\right) = \left(\frac{n}{2}\right)^2 (c+1) - c\,\frac{n}{2}$$

Can we do multiplication faster than $O(n^2)$?

Claim: if $T(n)$ satisfies

$$T(n) = 4T\left(\frac{n}{2}\right) + c\,n \tag{3}$$

then

$$T(n) = n^2(c+1) - c\,n$$

Proof: By "fast" induction. We assume it is true for $\lfloor n/2 \rfloor$:

$$T\left(\frac{n}{2}\right) = \left(\frac{n}{2}\right)^2 (c+1) - c\,\frac{n}{2}$$

and prove that it is also true for n:

Can we do multiplication faster than $O(n^2)$?

Claim: if $T(n)$ satisfies

$$T(n) = 4T\left(\frac{n}{2}\right) + c\,n \tag{3}$$

then

$$T(n) = n^2(c+1) - c\,n$$

Proof: By "fast" induction. We assume it is true for $\lfloor n/2 \rfloor$:

$$T\left(\frac{n}{2}\right) = \left(\frac{n}{2}\right)^2 (c+1) - c\,\frac{n}{2}$$

and prove that it is also true for n:

$$T\,(n) = 4\,T\left(\tfrac{n}{2}\right) + c\,n = 4\left(\left(\tfrac{n}{2}\right)^2 (c+1) - \tfrac{n}{2}\,c\right) + c\,n$$

Can we do multiplication faster than $O(n^2)$?

Claim: if $T(n)$ satisfies

$$T(n) = 4T\left(\frac{n}{2}\right) + c\,n \tag{3}$$

then

$$T(n) = n^2(c+1) - c\,n$$

Proof: By "fast" induction. We assume it is true for $\lfloor n/2 \rfloor$:

$$T\left(\frac{n}{2}\right) = \left(\frac{n}{2}\right)^2 (c+1) - c\,\frac{n}{2}$$

and prove that it is also true for n:

$$\begin{aligned} T\left(n\right) &= 4\,T\left(\tfrac{n}{2}\right) + c\,n = 4\left(\left(\tfrac{n}{2}\right)^2 (c+1) - \tfrac{n}{2}\,c\right) + c\,n \\ &= n^2(c+1) - 2c\,n + c\,n = n^2(c+1) - c\,n \end{aligned}$$

Can we do multiplication faster than $O(n^2)$?

Thus, if $T(n)$ satisfies

$$T(n) = 4\,T\left(\frac{n}{2}\right) + c\,n$$

Can we do multiplication faster than $O(n^2)$?

Thus, if $T(n)$ satisfies

$$T(n) = 4\,T\left(\frac{n}{2}\right) + c\,n$$

then

$$T(n) = n^2(c+1) - c\,n = O(n^2)$$

Can we do multiplication faster than $O(n^2)$?

Thus, if $T(n)$ satisfies

$$T(n) = 4\,T\left(\frac{n}{2}\right) + c\,n$$

then

$$T(n) = n^2(c+1) - c\,n = O(n^2)$$

i.e., we gained **nothing** with our divide-and-conquer!

Can we do multiplication faster than $O(n^2)$?

Thus, if $T(n)$ satisfies

$$T(n) = 4\,T\left(\frac{n}{2}\right) + c\,n$$

then

$$T(n) = n^2(c+1) - c\,n = O(n^2)$$

i.e., we gained **nothing** with our divide-and-conquer!

Some history: In 1952, one of the most famous mathematicians of the 20^{th} century, Andrey Kolmogorov, conjectured that you cannot multiply in less than $\Omega(n^2)$ elementary operations. In 1960, Karatsuba, then a 23-year-old student, found an algorithm (later it was called "divide and conquer") that multiplies two n-digit numbers in $\Theta(n^{\log_2 3}) \approx \Theta(n^{1.58\ldots})$ elementary steps, thus disproving the conjecture!! Kolmogorov was shocked!

The Karatsuba trick

How did Karatsuba do it??

The Karatsuba trick

How did Karatsuba do it??

Take again our two input numbers A and B, and split them into two halves:

$$A = A_1 2^{\frac{n}{2}} + A_0 \qquad \underbrace{XX\dots X}_{\frac{n}{2}}\underbrace{XX\dots X}_{\frac{n}{2}}$$

$$B = B_1 2^{\frac{n}{2}} + B_0$$

How did Karatsuba do it??

Take again our two input numbers A and B, and split them into two halves:

$$A = A_1 2^{\frac{n}{2}} + A_0 \qquad \underbrace{XX\ldots X}_{\frac{n}{2}}\underbrace{XX\ldots X}_{\frac{n}{2}}$$

$$B = B_1 2^{\frac{n}{2}} + B_0$$

- AB can now be calculated as follows:

The Karatsuba trick

How did Karatsuba do it??

Take again our two input numbers A and B, and split them into two halves:

$$A = A_1 2^{\frac{n}{2}} + A_0 \qquad \underbrace{XX\ldots X}_{\frac{n}{2}}\underbrace{XX\ldots X}_{\frac{n}{2}}$$

$$B = B_1 2^{\frac{n}{2}} + B_0$$

- AB can now be calculated as follows:

$$AB = A_1B_12^n + (A_1B_0 + A_0B_1)2^{\frac{n}{2}} + A_0B_0$$

How did Karatsuba do it??

Take again our two input numbers A and B, and split them into two halves:

$$A = A_1 2^{\frac{n}{2}} + A_0 \qquad \underbrace{XX\ldots X}_{\frac{n}{2}}\underbrace{XX\ldots X}_{\frac{n}{2}}$$

$$B = B_1 2^{\frac{n}{2}} + B_0$$

- AB can now be calculated as follows:

$$AB = A_1 B_1 2^n + (A_1 B_0 + A_0 B_1) 2^{\frac{n}{2}} + A_0 B_0$$

$$= A_1 B_1 2^n + ((A_1 + A_0)(B_1 + B_0) - A_1 B_1 - A_0 B_0) 2^{\frac{n}{2}} + A_0 B_0$$

How did Karatsuba do it??

Take again our two input numbers A and B, and split them into two halves:

$$A = A_1 2^{\frac{n}{2}} + A_0 \qquad \underbrace{XX\ldots X}_{\frac{n}{2}}\underbrace{XX\ldots X}_{\frac{n}{2}}$$

$$B = B_1 2^{\frac{n}{2}} + B_0$$

- AB can now be calculated as follows:

$$AB = A_1 B_1 2^n + (A_1 B_0 + A_0 B_1) 2^{\frac{n}{2}} + A_0 B_0$$

$$= A_1 B_1 2^n + ((A_1 + A_0)(B_1 + B_0) - A_1 B_1 - A_0 B_0) 2^{\frac{n}{2}} + A_0 B_0$$

Thus, the algorithm will look like this:

function $\textsc{Mult}(A, B)$
    if $|A| = |B| = 1$ **then return** AB
    else
        $A_1 \leftarrow \text{MoreSignificantPart}(A);$
        $A_0 \leftarrow \text{LessSignificantPart}(A);$
        $B_1 \leftarrow \text{MoreSignificantPart}(B);$
        $B_0 \leftarrow \text{LessSignificantPart}(B);$
        $U \leftarrow A_0 + A_1;$
        $V \leftarrow B_0 + B_1;$
        $X \leftarrow \textsc{Mult}(\mathrm{A}_0, \mathrm{B}_0);$
        $W \leftarrow \textsc{Mult}(\mathrm{A}_1, \mathrm{B}_1);$
        $Y \leftarrow \textsc{Mult}(\mathrm{U}, \mathrm{V});$
        return $W\,2^n + (Y - X - W)\,2^{n/2} + X$
    end if
end function

The Karatsuba trick

Since

$$T(n) = 3\,T\left(\frac{n}{2}\right) + c\,n$$

The Karatsuba trick

Since

$$T(n) = 3\,T\left(\frac{n}{2}\right) + c\,n$$

implies

$$T\left(\frac{n}{2}\right) = 3\,T\left(\frac{n}{2^2}\right) + c\,\frac{n}{2}$$

The Karatsuba trick

Since

$$T(n) = 3\,T\left(\frac{n}{2}\right) + c\,n$$

implies

$$T\left(\frac{n}{2}\right) = 3\,T\left(\frac{n}{2^2}\right) + c\,\frac{n}{2}$$

and

$$T\left(\frac{n}{2^2}\right) = 3\,T\left(\frac{n}{2^3}\right) + c\,\frac{n}{2^2}$$

The Karatsuba trick

Since

$$T(n) = 3\,T\left(\frac{n}{2}\right) + c\,n$$

implies

$$T\left(\frac{n}{2}\right) = 3\,T\left(\frac{n}{2^2}\right) + c\,\frac{n}{2}$$

and

$$T\left(\frac{n}{2^2}\right) = 3\,T\left(\frac{n}{2^3}\right) + c\,\frac{n}{2^2}$$

$$\ldots$$

The Karatsuba trick

Since

$$T(n) = 3\,T\left(\frac{n}{2}\right) + c\,n$$

implies

$$T\left(\frac{n}{2}\right) = 3\,T\left(\frac{n}{2^2}\right) + c\,\frac{n}{2}$$

and

$$T\left(\frac{n}{2^2}\right) = 3\,T\left(\frac{n}{2^3}\right) + c\,\frac{n}{2^2}$$

$$\ldots$$

we get

The Karatsuba trick

Since

$$T(n) = 3\,T\left(\frac{n}{2}\right) + c\,n$$

implies

$$T\left(\frac{n}{2}\right) = 3\,T\left(\frac{n}{2^2}\right) + c\,\frac{n}{2}$$

and

$$T\left(\frac{n}{2^2}\right) = 3\,T\left(\frac{n}{2^3}\right) + c\,\frac{n}{2^2}$$

$$\ldots$$

we get

$$T(n) = 3\,\underbrace{T\left(\frac{n}{2}\right)} + c\,n = 3\left(\underbrace{3T\left(\frac{n}{2^2}\right) + c\frac{n}{2}}\right) + c\,n$$

The Karatsuba trick

Since

$$T(n) = 3\,T\left(\frac{n}{2}\right) + c\,n$$

implies

$$T\left(\frac{n}{2}\right) = 3\,T\left(\frac{n}{2^2}\right) + c\,\frac{n}{2}$$

and

$$T\left(\frac{n}{2^2}\right) = 3\,T\left(\frac{n}{2^3}\right) + c\,\frac{n}{2^2}$$

$$\dots$$

we get

$$T(n) = 3\,\underbrace{T\left(\frac{n}{2}\right)} + c\,n = 3\left(\underbrace{3T\left(\frac{n}{2^2}\right) + c\frac{n}{2}}\right) + c\,n$$

$$= 3^2\,\underbrace{T\left(\frac{n}{2^2}\right)} + c\frac{3n}{2} + c\,n = 3^2\left(\underbrace{3T\left(\frac{n}{2^3}\right) + c\frac{n}{2^2}}\right) + c\frac{3n}{2} + c\,n$$

The Karatsuba trick

Since

$$T(n) = 3\,T\left(\frac{n}{2}\right) + c\,n$$

implies

$$T\left(\frac{n}{2}\right) = 3\,T\left(\frac{n}{2^2}\right) + c\,\frac{n}{2}$$

and

$$T\left(\frac{n}{2^2}\right) = 3\,T\left(\frac{n}{2^3}\right) + c\,\frac{n}{2^2}$$

$$\ldots$$

we get

$$T(n) = 3\,\underbrace{T\left(\frac{n}{2}\right)} + c\,n = 3\left(\underbrace{3T\left(\frac{n}{2^2}\right) + c\frac{n}{2}}\right) + c\,n$$

$$= 3^2\,\underbrace{T\left(\frac{n}{2^2}\right)} + c\frac{3n}{2} + c\,n = 3^2\left(\underbrace{3T\left(\frac{n}{2^3}\right) + c\frac{n}{2^2}}\right) + c\frac{3n}{2} + c\,n$$

$$= 3^3\,\underbrace{T\left(\frac{n}{2^3}\right)} + c\frac{3^2n}{2^2} + c\frac{3n}{2} + c\,n = 3^3\left(\underbrace{3T\left(\frac{n}{2^4}\right) + c\frac{n}{2^3}}\right) + c\frac{3^2n}{2^2} + c\frac{3n}{2} + c\,n = \ldots$$

The Karatsuba trick

$$T(n) = 3T\left(\tfrac{n}{2}\right) + c\,n = 3\left(3T\left(\tfrac{n}{2^2}\right) + c\tfrac{n}{2}\right) + c\,n = 3^2\,\underbrace{T\left(\frac{n}{2^2}\right)} + c\tfrac{3n}{2} + c\,n$$

$$= 3^2\left(\underbrace{3T\left(\frac{n}{2^3}\right) + c\frac{n}{2^2}}\right) + c\tfrac{3n}{2} + c\,n = 3^3 T\left(\tfrac{n}{2^3}\right) + c\tfrac{3^2 n}{2^2} + c\tfrac{3n}{2} + c\,n$$

$$= 3^3\,\underbrace{T\left(\frac{n}{2^3}\right)} + c\,n\left(\tfrac{3^2}{2^2} + \tfrac{3}{2} + 1\right)$$

$$= 3^3\left(\underbrace{3T\left(\frac{n}{2^4}\right) + c\frac{n}{2^3}}\right) + c\,n\left(\tfrac{3^2}{2^2} + \tfrac{3}{2} + 1\right)$$

The Karatsuba trick

$$T(n) = 3T\left(\tfrac{n}{2}\right) + c\,n = 3\left(3T\left(\tfrac{n}{2^2}\right) + c\tfrac{n}{2}\right) + c\,n = 3^2\,\underbrace{T\left(\frac{n}{2^2}\right)} + c\tfrac{3n}{2} + c\,n$$

$$= 3^2\left(\underbrace{3T\left(\frac{n}{2^3}\right) + c\frac{n}{2^2}}\right) + c\tfrac{3n}{2} + c\,n = 3^3 T\left(\tfrac{n}{2^3}\right) + c\tfrac{3^2 n}{2^2} + c\tfrac{3n}{2} + c\,n$$

$$= 3^3\,\underbrace{T\left(\frac{n}{2^3}\right)} + c\,n\left(\tfrac{3^2}{2^2} + \tfrac{3}{2} + 1\right)$$

$$= 3^3\left(\underbrace{3T\left(\frac{n}{2^4}\right) + c\frac{n}{2^3}}\right) + c\,n\left(\tfrac{3^2}{2^2} + \tfrac{3}{2} + 1\right)$$

$$= 3^4 T\left(\tfrac{n}{2^4}\right) + c\,n\left(\tfrac{3^3}{2^3} + \tfrac{3^2}{2^2} + \tfrac{3}{2} + 1\right)$$

The Karatsuba trick

$$T(n) = 3T\left(\tfrac{n}{2}\right) + c\,n = 3\left(3T\left(\tfrac{n}{2^2}\right) + c\tfrac{n}{2}\right) + c\,n = 3^2\,\underbrace{T\left(\frac{n}{2^2}\right)} + c\tfrac{3n}{2} + c\,n$$

$$= 3^2\left(\underbrace{3T\left(\frac{n}{2^3}\right) + c\frac{n}{2^2}}\right) + c\tfrac{3n}{2} + c\,n = 3^3T\left(\tfrac{n}{2^3}\right) + c\tfrac{3^2n}{2^2} + c\tfrac{3n}{2} + c\,n$$

$$= 3^3\,\underbrace{T\left(\frac{n}{2^3}\right)} + c\,n\left(\tfrac{3^2}{2^2} + \tfrac{3}{2} + 1\right)$$

$$= 3^3\left(\underbrace{3T\left(\frac{n}{2^4}\right) + c\frac{n}{2^3}}\right) + c\,n\left(\tfrac{3^2}{2^2} + \tfrac{3}{2} + 1\right)$$

$$= 3^4T\left(\tfrac{n}{2^4}\right) + c\,n\left(\tfrac{3^3}{2^3} + \tfrac{3^2}{2^2} + \tfrac{3}{2} + 1\right)$$

$\dots$

The Karatsuba trick

$$T(n) = 3T\left(\tfrac{n}{2}\right) + c\,n = 3\left(3T\left(\tfrac{n}{2^2}\right) + c\tfrac{n}{2}\right) + c\,n = 3^2\,\underbrace{T\left(\frac{n}{2^2}\right)} + c\tfrac{3n}{2} + c\,n$$

$$= 3^2\left(\underbrace{3T\left(\frac{n}{2^3}\right) + c\frac{n}{2^2}}\right) + c\tfrac{3n}{2} + c\,n = 3^3T\left(\tfrac{n}{2^3}\right) + c\tfrac{3^2n}{2^2} + c\tfrac{3n}{2} + c\,n$$

$$= 3^3\,\underbrace{T\left(\frac{n}{2^3}\right)} + c\,n\left(\tfrac{3^2}{2^2} + \tfrac{3}{2} + 1\right)$$

$$= 3^3\left(\underbrace{3T\left(\frac{n}{2^4}\right) + c\frac{n}{2^3}}\right) + c\,n\left(\tfrac{3^2}{2^2} + \tfrac{3}{2} + 1\right)$$

$$= 3^4T\left(\tfrac{n}{2^4}\right) + c\,n\left(\tfrac{3^3}{2^3} + \tfrac{3^2}{2^2} + \tfrac{3}{2} + 1\right)$$

$$\ldots$$

$$= 3^{\lfloor \log_2 n \rfloor}T\left(\frac{n}{\lfloor 2^{\log_2 n}\rfloor}\right) + c\,n\left(\left(\tfrac{3}{2}\right)^{\lfloor \log_2 n\rfloor - 1} + \ldots + \tfrac{3^2}{2^2} + \tfrac{3}{2} + 1\right)$$

The Karatsuba trick

$$T(n) = 3T\left(\frac{n}{2}\right) + c\,n = 3\left(3T\left(\frac{n}{2^2}\right) + c\frac{n}{2}\right) + c\,n = 3^2\,\underbrace{T\left(\frac{n}{2^2}\right)} + c\frac{3n}{2} + c\,n$$

$$= 3^2\left(\underbrace{3T\left(\frac{n}{2^3}\right) + c\frac{n}{2^2}}\right) + c\frac{3n}{2} + c\,n = 3^3T\left(\frac{n}{2^3}\right) + c\frac{3^2n}{2^2} + c\frac{3n}{2} + c\,n$$

$$= 3^3\,\underbrace{T\left(\frac{n}{2^3}\right)} + c\,n\left(\frac{3^2}{2^2} + \frac{3}{2} + 1\right)$$

$$= 3^3\left(\underbrace{3T\left(\frac{n}{2^4}\right) + c\frac{n}{2^3}}\right) + c\,n\left(\frac{3^2}{2^2} + \frac{3}{2} + 1\right)$$

$$= 3^4T\left(\frac{n}{2^4}\right) + c\,n\left(\frac{3^3}{2^3} + \frac{3^2}{2^2} + \frac{3}{2} + 1\right)$$

$$\ldots$$

$$= 3^{\lfloor\log_2 n\rfloor}T\left(\frac{n}{\lfloor 2^{\log_2 n}\rfloor}\right) + c\,n\left(\left(\frac{3}{2}\right)^{\lfloor\log_2 n\rfloor - 1} + \ldots + \frac{3^2}{2^2} + \frac{3}{2} + 1\right)$$

$$\approx 3^{\log_2 n}T(1) + c\,n\frac{\left(\frac{3}{2}\right)^{\log_2 n} - 1}{\frac{3}{2} - 1} = 3^{\log_2 n}T(1) + 2c\,n\left(\left(\frac{3}{2}\right)^{\log_2 n} - 1\right)$$

The Karatsuba trick

So we got

$$T(n) \approx 3^{\log_2 n} T(1) + 2c\,n \left(\left(\frac{3}{2}\right)^{\log_2 n} - 1 \right)$$

The Karatsuba trick

So we got

$$T(n) \approx 3^{\log_2 n} T(1) + 2c\,n \left(\left(\frac{3}{2}\right)^{\log_2 n} - 1 \right)$$

We now use $a^{\log_b n} = n^{\log_b a}$ to get:

The Karatsuba trick

So we got

$$T(n) \approx 3^{\log_2 n} T(1) + 2c\,n \left(\left(\frac{3}{2}\right)^{\log_2 n} - 1 \right)$$

We now use $a^{\log_b n} = n^{\log_b a}$ to get:

$$T(n) \approx n^{\log_2 3} T(1) + 2c\,n \left(n^{\log_2 \frac{3}{2}} - 1 \right) = n^{\log_2 3} T(1) + 2c\,n \left(n^{\log_2 3 - 1} - 1 \right)$$

The Karatsuba trick

So we got

$$T(n) \approx 3^{\log_2 n} T(1) + 2c\,n \left(\left(\frac{3}{2} \right)^{\log_2 n} - 1 \right)$$

We now use $a^{\log_b n} = n^{\log_b a}$ to get:

$$T(n) \approx n^{\log_2 3} T(1) + 2c\,n \left(n^{\log_2 \frac{3}{2}} - 1 \right) = n^{\log_2 3} T(1) + 2c\,n \left(n^{\log_2 3 - 1} - 1 \right)$$

$$= n^{\log_2 3} T(1) + 2c\,n^{\log_2 3} - 2c\,n$$

The Karatsuba trick

So we got

$$T(n) \approx 3^{\log_2 n} T(1) + 2c\,n \left(\left(\frac{3}{2}\right)^{\log_2 n} - 1 \right)$$

We now use $a^{\log_b n} = n^{\log_b a}$ to get:

$$\begin{aligned} T(n) &\approx n^{\log_2 3} T(1) + 2c\,n \left(n^{\log_2 \frac{3}{2}} - 1 \right) = n^{\log_2 3} T(1) + 2c\,n \left(n^{\log_2 3 - 1} - 1 \right) \\ &= n^{\log_2 3} T(1) + 2c\,n^{\log_2 3} - 2c\,n \\ &= O(n^{\log_2 3}) = O(n^{1.58\ldots}) \ll n^2 \end{aligned}$$

The Karatsuba trick

So we got

$$T(n) \approx 3^{\log_2 n} T(1) + 2c\,n \left(\left(\frac{3}{2} \right)^{\log_2 n} - 1 \right)$$

We now use $a^{\log_b n} = n^{\log_b a}$ to get:

$$\begin{aligned} T(n) \approx n^{\log_2 3} T(1) + 2c\,n \left(n^{\log_2 \frac{3}{2}} - 1 \right) &= n^{\log_2 3} T(1) + 2c\,n \left(n^{\log_2 3 - 1} - 1 \right) \\ &= n^{\log_2 3} T(1) + 2c\,n^{\log_2 3} - 2c\,n \\ &= O(n^{\log_2 3}) = O(n^{1.58\ldots}) \ll n^2 \end{aligned}$$

Please review the basic properties of logarithms and the asymptotic notation from the review material (the first item at the class webpage under "class resources".)

Next time:

1. Can we multiply large integers faster than $O\left(n^{\log_2 3}\right)$??

Next time:

1. Can we multiply large integers faster than $O\left(n^{\log_2 3}\right)$??
2. Can we avoid messy computations like:

$$
\begin{aligned}
T(n) &= 3T\left(\frac{n}{2}\right) + cn = 3\left(3T\left(\frac{n}{2^2}\right) + c\frac{n}{2}\right) + cn = 3^2T\left(\frac{n}{2^2}\right) + c\frac{3n}{2} + cn \\
&= 3^2\left(3T\left(\frac{n}{2^3}\right) + c\frac{n}{2^2}\right) + c\frac{3n}{2} + cn = 3^3T\left(\frac{n}{2^3}\right) + c\frac{3^2n}{2^2} + c\frac{3n}{2} + cn \\
&= 3^3T\left(\frac{n}{2^3}\right) + cn\left(\frac{3^2}{2^2} + \frac{3}{2} + 1\right) = \\
&= 3^3\left(3T\left(\frac{n}{2^4}\right) + c\frac{n}{2^3}\right) + cn\left(\frac{3^2}{2^2} + \frac{3}{2} + 1\right) = \\
&= 3^4T\left(\frac{n}{2^4}\right) + cn\left(\frac{3^3}{2^3} + \frac{3^2}{2^2} + \frac{3}{2} + 1\right) = \\
&\ldots \\
&= 3^{\lfloor \log_2 n \rfloor}T\left(\frac{n}{\lfloor 2^{\log_2 n} \rfloor}\right) + cn\left(\left(\frac{3}{2}\right)^{\lfloor \log_2 n \rfloor - 1} + \ldots + \frac{3^2}{2^2} + \frac{3}{2} + 1\right) \\
&\approx 3^{\log_2 n}T(1) + cn\frac{\left(\frac{3}{2}\right)^{\log_2 n} - 1}{\frac{3}{2} - 1} \\
&= 3^{\log_2 n}T(1) + 2cn\left(\left(\frac{3}{2}\right)^{\log_2 n} - 1\right)
\end{aligned}
$$

That's All, Folks!!