Lecture #1

CS170

Spring 2021

Lecture # 1 – 2021.01.19

Welcome to CS170!

Tuesdays and Thursdays at 09:00 - 10:30

Always the same Zoom link, available on Piazza.

Instructors: Alessandro Chiesa & James Demmel

Resources: Piazza and webpage (https://cs170.org)

if you have trouble accessing any resources please contact your GSI

Contact: please use Piazza to contact staff

if you need to use email then use cs170@berkeley.edu (rather than personal emails)

Prerequisites are enforced.

Homework policy is no late submissions. You can drop 2 HWs and ≥90% is full credit.

Please read collaboration policy on website.

Please note dates and times of midterms and final

(they are proctored live online and we do not provide alternative options).

We have an amazing course team this semester and look forward to guide you through this course.

We encourage you to get to know the staff at discussion sections and office hours.

Let's get started!

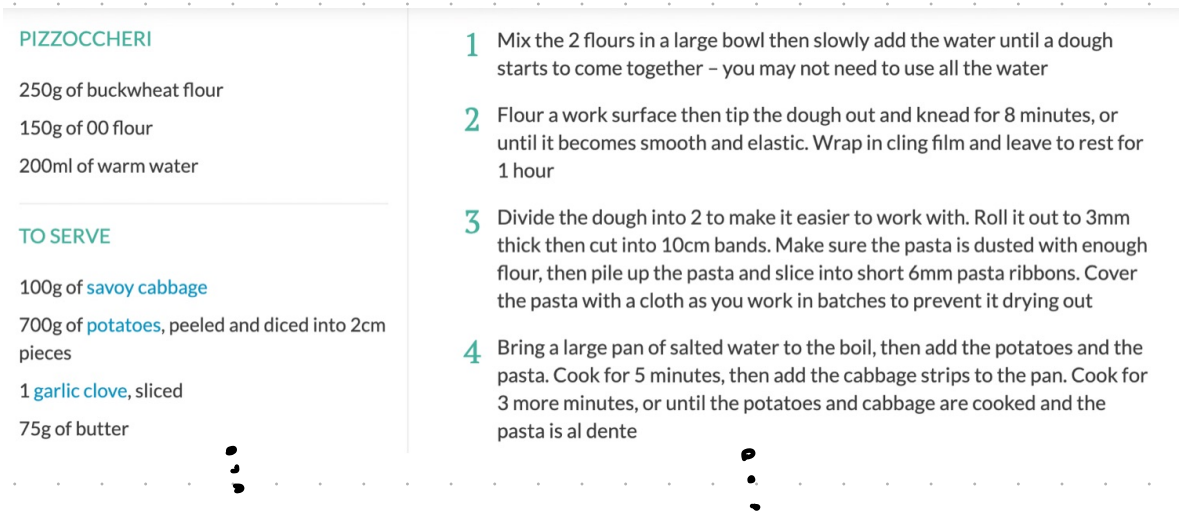
This course is (mostly*) about algorithms.

* you will also learn about intractability

What are algorithms?

An algorithm is a well-defined procedure for carrying out a given task.

Example: a food recipe



PIZZOCCHERI

250g of buckwheat flour

150g of 00 flour

200ml of warm water

TO SERVE

100g of savoy cabbage

700g of potatoes, peeled and diced into 2cm pieces

1 garlic clove, sliced

75g of butter

1 Mix the 2 flours in a large bowl then slowly add the water until a dough starts to come together – you may not need to use all the water

2 Flour a work surface then tip the dough out and knead for 8 minutes, or until it becomes smooth and elastic. Wrap in cling film and leave to rest for 1 hour

3 Divide the dough into 2 to make it easier to work with. Roll it out to 3mm thick then cut into 10cm bands. Make sure the pasta is dusted with enough flour, then pile up the pasta and slice into short 6mm pasta ribbons. Cover the pasta with a cloth as you work in batches to prevent it drying out

4 Bring a large pan of salted water to the boil, then add the potatoes and the pasta. Cook for 5 minutes, then add the cabbage strips to the pan. Cook for 3 more minutes, or until the potatoes and cabbage are cooked and the pasta is al dente

What about algorithms in this class?

arithmetic, graphs, scheduling, games, hashing, ...

You will develop the skills to design algorithms that (halt and) are:

① correct (return the correct answer on every input)

② efficient (in time, but will also consider space, randomness, ...)

Rest of today: algorithms for integer arithmetic

We focus on two basic operations: addition and multiplication.

That is we are searching for algorithms A_{add} and A_{mul} s.t.

- $\forall x,y \quad A_{add}(x,y) = x+y$ and A_{add} is efficient
- $\forall x,y \quad A_{mul}(x,y) = x \cdot y$ and A_{mul} is efficient

Q: how are the integer inputs represented?

Over time people have devised different numeral systems.

E.g. Roman numerals – III = 3, IX = 9, XL = 40, CI = 101, ...

E.g. Arabic numerals – 3, 9, 40, 101 ← positional numeral system

$$1 \cdot 10^2 + 0 \cdot 10^1 + 1 \cdot 10^0$$

More generally the digits $a_{n-1} \dots a_1 a_0$ represent the number $\sum_{i=0}^{n-1} a_i \cdot \underbrace{10}_{\text{base}}^{i}$.

So n digits can represent any number between 0 and $10^n - 1$.

E.g. for n = 3 we have 001, 002, ..., 010, 010, ..., 998, 999.

We will use this representation throughout today and in this course.

ADDITION $(x, y \mapsto x+y)$

First idea: "count by fingers"

$7+5 \to 8+4 \to 9+3 \to 10+2 \to 11+1 \to \underline{12}+0$

that is, add 1 to x, for y times.

Correctness: ✓

Efficiency: if x, y are each at most n digits the time is
$y \cdot (\text{time to increment}) \leq y \cdot n \leq (10^n - 1) \cdot n$ [a better bound of $\sim 10^n$ can be shown]
This is SLOOOOOW!!

Second idea: "carry method"

$$\begin{array}{r} {\scriptstyle +1} \\ 2754 \\ 0129 \\ \hline 2883 \end{array}$$

For each digit location $i = 0, 1, \ldots, n-1$:
sum digits at i plus any carry from location $i-1$.

Correctness: ✓ Efficiency: $n \cdot (\text{time to sum two digits}) = n \cdot c \ll 10^n$ (some constant)

Can we do better than n? No. Every digit of the input "matters".
Also, writing the output takes n ops.

The algorithm you studied in school is the best one can do for addition.

Multiplication $(x, y \mapsto x \cdot y)$

First idea: do $+x$ for y times (e.g. $5 \cdot 3 = \overbrace{5+5+5}^{3}$)

Correctness: ✓ Efficiency: $y \cdot (\text{cost to add } x) = y \cdot n \leq (10^n - 1) \cdot n$ ☹

Second idea: "long multiplication"

$$\begin{array}{r} 3517 \\ \times\ 143 \end{array} = 3517 \times 100 + 3517 \times 40 + 3517 \times 3 = \ldots = 502931.$$

In general: $x \cdot y = x \cdot \sum_{i=0}^{n-1} y_i \cdot 10^i = \sum_{i=0}^{n-1} (x \cdot y_i) \cdot 10^i$.

Computing $\{x \cdot y_{n-1}, x \cdot y_{n-2}, \ldots, x \cdot y_0\}$ takes n^2 ops, and adding all up (with shifts) also $\approx n^2$ ops.
Overall, the efficiency of this algorithm is $C \cdot n^2$.

You should ask: Can we do better?

Intuitively it seems that the answer is NO.
Each digit of x needs to "interact" with each digit of y.
That is n^2 terms that we need to compute.

But ... the plot thickens.

Interlude

We do not wish to count exact number of elementary operations of an algorithm. Instead we study growth rates as a function of input size n.

The addition algorith ran in time $T(n) = O(n)$.
By this we mean:

$$\exists\, c, N \quad \forall\, n \geq N \quad T(n) \leq c \cdot n.$$

Similarly the multiplication algorithm ran in time $T(n) = O(n^2)$.
We similarly mean:

$$\exists\, c, N \quad \forall\, n \geq N \quad T(n) \leq c \cdot n^2.$$

Notice that $n = O(n^2)$ but a better bound is $O(n)$.

More generally: $f(n) = O(g(n))$ if for large enough n we have $f(n) \leq c\, g(n)$.

There is other notation that you will practice in discussions & HWs:

$$\Omega(g(n)) = \text{"at least } c \cdot g(n)\text{"} \qquad \Theta(g(n)) = \text{"} O(g(n)) \wedge \Omega(g(n))\text{"}$$

Karatsuba Multiplication

Illustrative example of a powerful technique: divide & conquer

$$X = \underbrace{35}_{X_H}\underbrace{17}_{X_L} \qquad Y = \underbrace{01}_{Y_H}\underbrace{43}_{Y_L}$$

$$X = X_H \cdot 10^{n/2} + X_L \qquad Y = Y_H \cdot 10^{2} + Y_L$$

As multiplication distributes over addition:

$$X \cdot Y = X_H \cdot Y_H \cdot \underbrace{10^{n}}_{\text{shift}} + \underbrace{(X_H \cdot Y_L + X_L \cdot Y_H)}_{O(n)} \cdot \underbrace{10^{n/2}}_{\text{shift}} + X_L \cdot Y_L .$$

4 multiplications of $n/2$-digit integers

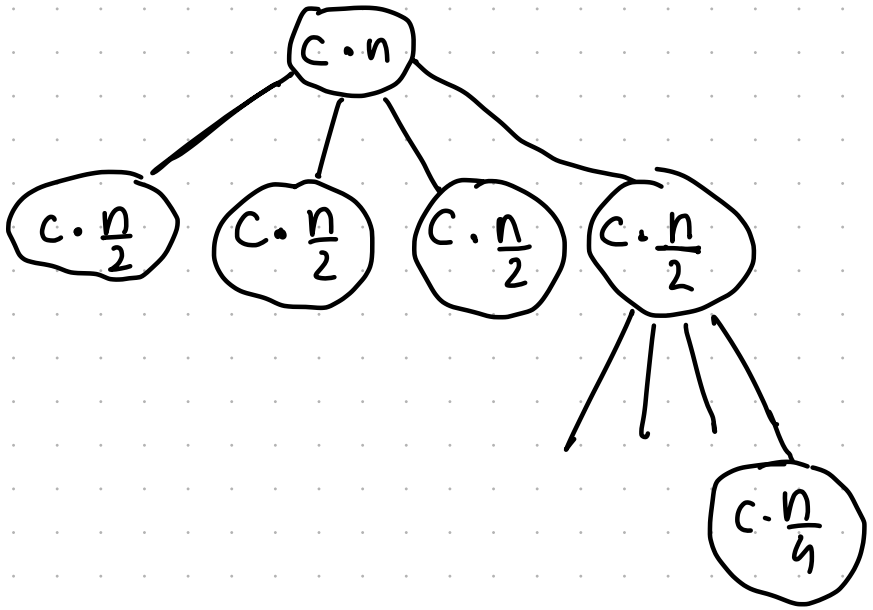
The time of running this algorithm is:

$$T(n) = 4 \cdot T\left(\frac{n}{2}\right) + C \cdot n$$

$$T(1) = 1$$

↑ look up in 10x10 multiplication table

Better than n^2?
Worse than n^2?
Exactly n^2?



$1 \cdot cn$

$4 \cdot c \cdot \frac{n}{2} = 2 \cdot cn$

$16 \cdot c \cdot \frac{n}{4} = 4 \cdot cn$

$\dots\dots$ $4^i \cdot c \cdot \frac{n}{2^i} = 2^i \cdot cn$

total time is

$cn \cdot (1 + 2 + \dots + 2^k)$

for $k := \log_2 n$.

$$cn \cdot \frac{2^{k+1} - 1}{2 - 1} \leq cn \cdot 2^{\log_2 n + 1} = cn \cdot 2n = O(n^2)$$

This would be even slower than long multiplication because of recursion overheads. ☹

Recall how to sum geometric series:

$$A = 1 + b + b^2 + \dots + b^k$$

$$\Rightarrow bA = b + b^2 + b^3 + \dots + b^{k+1}$$

$$\Rightarrow bA - A = b^{k+1} - 1$$

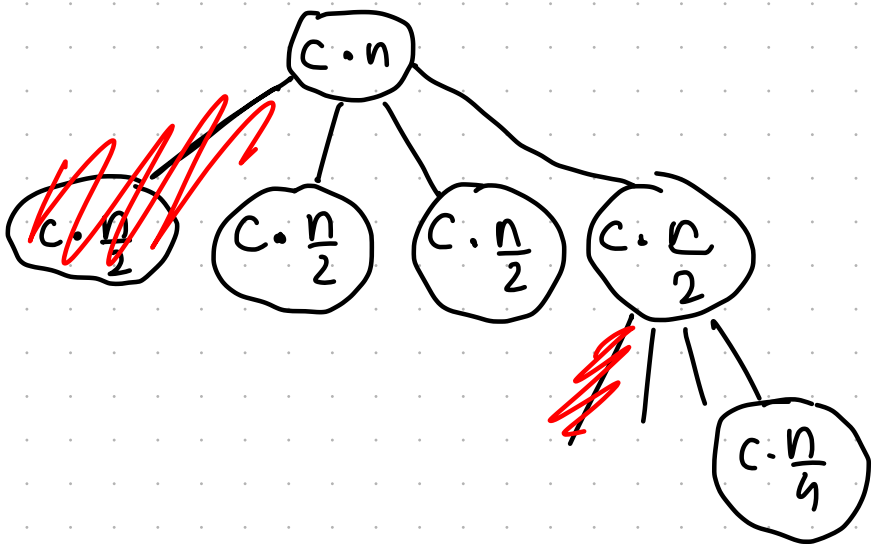
$$\Rightarrow A = \frac{b^{k+1} - 1}{b - 1} \quad (b \neq 1)$$

We can use a more clever recursion to beat n^2:

we do **not** need to compute $x_H y_L$ and $x_L y_H$. **we only need their sum**

Karatsuba $(x, y) :=$
1. $A :=$ Karatsuba (x_H, y_H)
2. $B :=$ Karatsuba (x_L, y_L)
3. $C :=$ Karatsuba $(x_H + x_L, y_H + y_L)$
4. return $A \cdot 10^n + (C - A - B) \cdot 10^{n/2} + B$

The new recurrence relation is: $T(n) = 3 \cdot T\left(\frac{n}{2}\right) + O(n)$



$1 \cdot cn$

$3 \cdot c\frac{n}{2}$

$9 \cdot c\frac{n}{4}$

$$cn \cdot \left(1 + \frac{3}{2} + \left(\frac{3}{2}\right)^2 + \cdots + \left(\frac{3}{2}\right)^k\right) \qquad k = \log_2 n$$

$$= cn \cdot \frac{\left(\frac{3}{2}\right)^{k+1} - 1}{\frac{3}{2} - 1} \leq cn\, 4 \left(\frac{3}{2}\right)^{\log_2 n}$$

$$= 4c \cdot 3^{\log_2 n} = 4c \cdot \left(2^{\log_2 3}\right)^{\log_2 n}$$

$$= 4c \cdot n^{\log_2 3} = O\left(n^{\log_2 3}\right) \quad !$$

Recall that $(a^b)^c = (a^c)^b$.

And in practice beats n^2 algo when inputs are a few tens of digits.

On algorithms for integer multiplication

$\log^*(a)$ = # of times to app $\log_2$ to get 1

$\log^*(2)=1$, $\log^*(2^2)=2$, $\log^*(2^{2^2})=3$, ...

div & conq

- 1960 Karatsuba $O(n^{\log_2 3})$ break into 2 parts & do 3 muls instead of $2^2=4$
- 1963 Toom & Cook $\cdot\ O(n^{\log_3 5})$ break into 3 parts & do 5 muls instead of $3^2=9$
- 1966 Cook $\cdot\ \forall k\ O_k(n^{\log_k(2k-1)})$ k parts, 2k-1 muls instead of k^2 [idea is polynomial interpolation]

FFT

- 1971 Schönhage & Strassen $O(n \log n \log\log n)$
- 2007 Fürer $O(n \log n\, 2^{\Theta(\log^* n)})$
- 2019 Harvey van der Hoeven $O(n \log n)$

believed to be optimal

Quantamagazine

NUMBER THEORY

Mathematicians Discover the Perfect Way to Multiply

By KEVIN HARTNETT

April 11, 2019

By chopping up large numbers into smaller ones, researchers have rewritten a fundamental mathematical speed limit.