CS 341: Algorithms

University of Waterloo
Éric Schost
eschost@uwaterloo.ca

Module 2: recurrences - master theorem - divide-and-conquer

An example: merge sort

Merge sort

Goal: sort an array of size n.

Idea: **divide-and-conquer**

- split the array in halves
- sort both halves
- merge

Call $\boldsymbol{t(n)}$ the maximum number of comparisons done for inputs of length n

Remarks (from CS240)

- merging two sorted arrays of size $n/2$ uses at most $n-1$ comparisons
- should not allocate new arrays, work in place

Intro: a useful divide-and-conquer recurrence

The function $t(n)$ satisfies

$$t(1) = 0, \quad t(n) = 2t(n/2) + n - 1 \quad (n \text{ a power of } 2)$$

Remark: $\leq$ is easy, $=$ needs a little care

Let $\boldsymbol{T(n)}$ be such that

$$T(1) = 0, \quad T(n) = 2T(n/2) + n \quad (n \text{ a power of } 2)$$

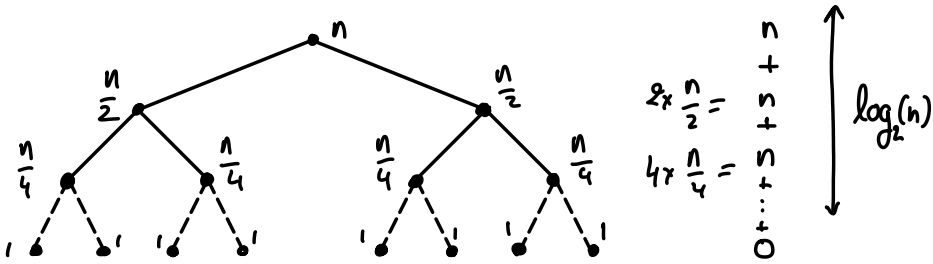
Observation: $t(n) \leq T(n)$, by induction

Unrolling the recurrence

$$
\begin{aligned}
T(n) &= 2T(n/2) + n \\
&= 2(2T(n/4) + n/2) + n \\
&= 4T(n/4) + n + n \\
&= \mathbf{4}T(n/\mathbf{4}) + \mathbf{2}n \\
&= \mathbf{8}T(n/\mathbf{8}) + \mathbf{3}n \\
&= \ldots \\
&= \boldsymbol{n}T(n/\boldsymbol{n}) + \mathbf{log_2(}\boldsymbol{n}\mathbf{)}n \\
&= nT(1) + n\log_2(n) = n\log_2(n)
\end{aligned}
$$

(n a power of two)

Alternative: the recursion tree



Overall, $T(n) = n \log_2(n)$, n a power of 2

Remark: expression for $t(n)$ a bit less nice: $t(n) = n(\log_2(n) - 1) + 1$.

Alternative: guess and prove

Guess: $T(n) = n$

$$n \stackrel{?}{=} 2(n/2) + n$$

Alternative: guess and prove

Guess: $T(n) = n$

$$n \stackrel{?}{=} 2(n/2) + n$$

Guess: $T(n) = kn$, k TBD?

$$kn \stackrel{?}{=} 2(kn/2) + n = kn + n$$

Alternative: guess and prove

Guess: $T(n) = n$

$$n \stackrel{?}{=} 2(n/2) + n$$

Guess: $T(n) = kn$, k TBD?

$$kn \stackrel{?}{=} 2(kn/2) + n = kn + n$$

Guess: $T(n) = kn \log_2(n)$, k TBD?

$$kn \log_2(n) \stackrel{?}{=} 2(k(n/2) \log_2(n/2)) + n$$

RHS is $kn \log_2(n/2) + n = kn \log_2(n) - kn + n$, so OK if $k = 1$.

The master theorem

Master theorem

Consider two integers $a, b \geq 1$, a real number $c \geq 0$ and a function $T(n)$ s.t.

$$T(n) = aT(\frac{n}{b}) + \Theta(n^c), \quad T(1) = C$$

for n a power of b.

Note: for the analysis of a recursive algorithm,

- b is the factor by which we reduce the problem size
- a is the number of recursive calls
- $\Theta(n^c)$ is the cost to prepare the recursive calls and combine their results

Master theorem

Consider two integers $a, b \geq 1$, a real number $c \geq 0$ and a function $T(n)$ s.t.

$$T(n) = aT(\frac{n}{b}) + \Theta(n^c), \quad T(1) = C$$

for n a power of b.

Theorem (clean)

$$T(b^k) = \begin{cases} \Theta((b^k)^c) & \text{if } a < b^c \iff c > \log_b(a) \\ \Theta((b^k)^c \log(b^k)) & \text{if } a = b^c \iff c = \log_b(a) \\ \Theta((b^k)^{\log_b(a)}) & \text{if } a > b^c \iff c < \log_b(a) \end{cases}$$

Master theorem

Consider two integers $a, b \geq 1$, a real number $c \geq 0$ and a function $T(n)$ s.t.

$$T(n) = aT(\frac{n}{b}) + \Theta(n^c), \quad T(1) = C$$

for n a power of b.

Theorem (dirty)

$$T(n) = \begin{cases} \Theta(n^c) & \text{if } a < b^c \iff c > \log_b(a) \\ \Theta(n^c \log(n)) & \text{if } a = b^c \iff c = \log_b(a) \\ \Theta(n^{\log_b(a)}) & \text{if } a > b^c \iff c < \log_b(a) \end{cases}$$

Master theorem

Consider two integers $a, b \geq 1$, a real number $c \geq 0$ and a function $T(n)$ s.t.

$$T(n) = aT(\frac{n}{b}) + \Theta(n^c), \quad T(1) = C$$

for n a power of b.

Theorem (dirty)

$$T(n) = \begin{cases} \Theta(n^c) & \text{root-heavy} \\ \Theta(n^c \log(n)) & a = b^c \\ \Theta(n^{\log_b(a)}) & a > b^c \end{cases}$$

Master theorem

Consider two integers $a, b \geq 1$, a real number $c \geq 0$ and a function $T(n)$ s.t.

$$T(n) = aT(\frac{n}{b}) + \Theta(n^c), \quad T(1) = C$$

for n a power of b.

Theorem (dirty)

$$T(n) = \begin{cases} \Theta(n^c) & a < b^c \\ \Theta(n^c \log(n)) & a = b^c \\ \Theta(n^{\log_b(a)}) & \text{leaf-heavy} \end{cases}$$

Master theorem

Consider two integers $a, b \geq 1$, a real number $c \geq 0$ and a function $T(n)$ s.t.

$$T(n) = aT(\frac{n}{b}) + \Theta(n^c), \quad T(1) = C$$

for n a power of b.

Theorem (dirty)

$$T(n) = \begin{cases} \Theta(n^c) & a < b^c \\ \Theta(n^c \log(n)) & a = b^c \\ \Theta(n^{\log_b(a)}) & \text{leaf-heavy} \end{cases}$$

Only doing the proof for $C = 1$ and $\Theta(n^c) = n^c$, general case is just a bit longer.

Remark: similar results with big-O and big-Omega instead of Θ

Examples

$\boldsymbol{T(n) = 4T(n/2) + n}$ multiplying polynomials

- $a = 4$, $b = 2$, $c = 1$ so $\log_b(a) = 2$ and $T(n) = \Theta(n^2)$

Examples

$\boldsymbol{T(n) = 4T(n/2) + n}$ multiplying polynomials

- $a = 4$, $b = 2$, $c = 1$ so $\log_b(a) = 2$ and $T(n) = \Theta(n^2)$

$\boldsymbol{T(n) = 2T(n/2) + n^2}$

- $a = 2$, $b = 2$, $c = 2$ so $\log_b(a) = 1$ and $T(n) = \Theta(n^2)$

Examples

$T(n) = 4T(n/2) + n$ multiplying polynomials

- $a = 4$, $b = 2$, $c = 1$ so $\log_b(a) = 2$ and $T(n) = \Theta(n^2)$

$T(n) = 2T(n/2) + n^2$

- $a = 2$, $b = 2$, $c = 2$ so $\log_b(a) = 1$ and $T(n) = \Theta(n^2)$

$T(n) = 2T(n/4) + 1$ kd-trees

- $a = 2$, $b = 4$, $c = 0$ so $\log_b(a) = 1/2$ and $T(n) = \Theta(\sqrt{n})$

Examples

$\boldsymbol{T(n) = T(n/2) + 1}$ binary search

- $a = 1$, $b = 2$, $c = 0$ so $\log_b(a) = 0$ and $T(n) = \Theta(\log(n))$

Examples

$T(n) = T(n/2) + 1$ binary search

- $a = 1$, $b = 2$, $c = 0$ so $\log_b(a) = 0$ and $T(n) = \Theta(\log(n))$

$T(n) = T(n/2) + n$

- $a = 1$, $b = 2$, $c = 1$ so $\log_b(a) = 0$ and $T(n) = \Theta(n)$

Examples

$T(n) = T(n/2) + 1$ binary search

- $a = 1$, $b = 2$, $c = 0$ so $\log_b(a) = 0$ and $T(n) = \Theta(\log(n))$

$T(n) = T(n/2) + n$

- $a = 1$, $b = 2$, $c = 1$ so $\log_b(a) = 0$ and $T(n) = \Theta(n)$

$T(n) = T(n/2)$

- ?

Examples

$\boldsymbol{T(n) = T(n/2) + 1}$ binary search

- $a = 1$, $b = 2$, $c = 0$ so $\log_b(a) = 0$ and $T(n) = \Theta(\log(n))$

$\boldsymbol{T(n) = T(n/2) + n}$

- $a = 1$, $b = 2$, $c = 1$ so $\log_b(a) = 0$ and $T(n) = \Theta(n)$

$\boldsymbol{T(n) = T(n/2)}$

- ?

$\boldsymbol{T(n) = 2T(n/2) + n \log(n)}$

- ?

Proof

1: a formula for $n = b^k$. The definition becomes

$$\begin{aligned} T(b^k) &= aT(b^{k-1}) + b^{kc} \\ &= a\big(aT(b^{k-2}) + b^{(k-1)c}\big) + b^{kc} \\ &= a^2T(b^{k-2}) + ab^{(k-1)c} + b^{kc} \\ &= \dots \\ &= a^k + a^{k-1}b^c + a^{k-2}b^{2c} + \dots + ab^{(k-1)c} + b^{kc}. \end{aligned}$$

Note: recursion tree has 1 root, a children, a^2 grand-children, $\dots$, a^k leaves

Proof

1: a formula for $n = b^k$. The definition becomes

$$\begin{aligned} T(b^k) &= aT(b^{k-1}) + b^{kc} \\ &= a\big(aT(b^{k-2}) + b^{(k-1)c}\big) + b^{kc} \\ &= a^2T(b^{k-2}) + ab^{(k-1)c} + b^{kc} \\ &= \dots \\ &= a^k + a^{k-1}b^c + a^{k-2}b^{2c} + \cdots + ab^{(k-1)c} + b^{kc}. \end{aligned}$$

Note: recursion tree has 1 root, a children, a^2 grand-children, $\dots$, a^k leaves

$$\begin{aligned} T(b^k) &= a^k \left(1 + \tfrac{b^c}{a} + \cdots + \left(\tfrac{b^c}{a}\right)^k\right) \\ &= (b^c)^k \left(1 + \tfrac{a}{b^c} + \cdots + \left(\tfrac{a}{b^c}\right)^k\right) \end{aligned}$$

Proof

2: case discussion.

For $n = b^k$,

- $k = \log_b(n)$ so $a^k = a^{\log_b(n)} = n^{\log_b(a)}$
- $(b^c)^k = n^c$

Proof

2: case discussion.

For $n = b^k$,

- $k = \log_b(n)$ so $a^k = a^{\log_b(n)} = n^{\log_b(a)}$
- $(b^c)^k = n^c$

so

- if $a = b^c$, $T(n) = a^k(k+1) = n^c(\log_b(n) + 1)$
- if $a > b^c$, the first sum is in $[1, K_1)$ so $n^{\log_b(a)} \leq T(n) \leq K_1 n^{\log_b(a)}$
- if $a < b^c$, the second sum is in $[1, K_2)$ so $n^c \leq T(n) \leq K_2 n^c$

This is **only** for $n = b^k$.

Application to runtime analysis

Suppose algorithm A does the following on inputs I of size n.

If $1 \leq n < b$, unit cost, else

- a recursive calls in sizes $\lfloor n/b \rfloor$ or $\lfloor n/b \rfloor + 1$ (if b does not divide n), or exactly n/b (if b divides n)
- $X(I)$ extra operations.

Application to runtime analysis

Suppose algorithm A does the following on inputs I of size n.

If $1 \leq n < b$, unit cost, else

- a recursive calls in sizes $\lfloor n/b \rfloor$ or $\lfloor n/b \rfloor + 1$ (if b does not divide n), or exactly n/b (if b divides n)
- $X(I)$ extra operations.

Example:

- merge-sort, counting only comparisons: $a = b = 2$ and

$$\lfloor n/2 \rfloor \leq X(I) \leq n - 1$$

for input I of size n.

Application to runtime analysis

Suppose algorithm A does the following on inputs I of size n.

If $1 \leq n < b$, unit cost, else

- a recursive calls in sizes $\lfloor n/b \rfloor$ or $\lfloor n/b \rfloor + 1$ (if b does not divide n), or exactly n/b (if b divides n)
- $X(I)$ extra operations.

Theorem

Suppose that $\boldsymbol{X(I) \leq h(|I|)}$, with $\boldsymbol{h(n) \in O(n^c)}$. Then

$$T_{\text{worst}}(n) = \begin{cases} O(n^c) & \text{if } a < b^c \iff c > \log_b(a) \\ O(n^c \log(n)) & \text{if } a = b^c \iff c = \log_b(a) \\ O(n^{\log_b(a)}) & \text{if } a > b^c \iff c < \log_b(a) \end{cases}$$

Application to runtime analysis

Suppose algorithm A does the following on inputs I of size n.

If $1 \leq n < b$, unit cost, else

- a recursive calls in sizes $\lfloor n/b \rfloor$ or $\lfloor n/b \rfloor + 1$ (if b does not divide n), or exactly n/b (if b divides n)
- $X(I)$ extra operations.

Theorem

Suppose that $\ell(|I|) \leq X(I)$, with $\ell(n) \in \Omega(n^c)$. Then

$$T_{\text{worst}}(n) = \begin{cases} \Omega(n^c) & \text{if } a < b^c \iff c > \log_b(a) \\ \Omega(n^c \log(n)) & \text{if } a = b^c \iff c = \log_b(a) \\ \Omega(n^{\log_b(a)}) & \text{if } a > b^c \iff c < \log_b(a) \end{cases}$$

Application to runtime analysis

Suppose algorithm A does the following on inputs I of size n.

If $1 \leq n < b$, unit cost, else

- a recursive calls in sizes $\lfloor n/b \rfloor$ or $\lfloor n/b \rfloor + 1$ (if b does not divide n), or exactly n/b (if b divides n)
- $X(I)$ extra operations.

Theorem

Suppose that $\ell(|I|) \leq X(I) \leq h(|I|)$, with $\ell(n), h(n) \in \Theta(n^c)$. Then

$$T_{\text{worst}}(n) = \begin{cases} \Theta(n^c) & \text{if } a < b^c \iff c > \log_b(a) \\ \Theta(n^c \log(n)) & \text{if } a = b^c \iff c = \log_b(a) \\ \Theta(n^{\log_b(a)}) & \text{if } a > b^c \iff c < \log_b(a) \end{cases}$$

Application to runtime analysis

Suppose algorithm A does the following on inputs I of size n.

If $1 \leq n < b$, unit cost, else

- a recursive calls in sizes $\lfloor n/b \rfloor$ or $\lfloor n/b \rfloor + 1$ (if b does not divide n), or exactly n/b (if b divides n)
- $X(I)$ extra operations.

Example:

- merge-sort, counting only comparisons: $a = b = 2$, $\ell(n) = \lfloor n/2 \rfloor$, $h(n) = n - 1$
  Both $\ell(n)$ and $h(n)$ are in $\boldsymbol{\Theta(n)}$, so $c = 1$ and $\boldsymbol{T_{\text{worst}}(n) \in \Theta(n \log(n))}$.

Divide-and-conquer algorithms

The framework

To solve a problem in size n:

Divide

- break the input into **smaller** problems
- ideally few such problems, all of size n/b for some constant b

Conquer

- solve these subproblems recursively

Recombine

- deduce the solution of the main problem from the subproblems

Multiplying polynomials

Goal: given $F = f_0 + \cdots + f_{n-1}x^{n-1}$ and $G = g_0 + \cdots + g_{n-1}x^{n-1}$, compute

$$H = FG = f_0g_0 + (f_0g_1 + f_1g_0)x + \cdots + f_{n-1}g_{n-1}x^{2n-2}$$

Remark: assume all f_i and g_i fit in one word. Then, input and output size $\Theta(n)$, easy algorithm in $\Theta(n^2)$.

```
1.  for i = 0, ..., n − 1 do
2.      for j = 0, ..., n − 1 do
3.          h_{i+j} = h_{i+j} + f_i g_j
```

Divide-and-conquer

Idea: write $F = F_0 + F_1 x^{n/2}, G = G_0 + G_1 x^{n/2}$. Then

$$H = F_0 G_0 + (F_0 G_1 + F_1 G_0) x^{n/2} + F_1 G_1 x^n$$

Analysis:

- 4 recursive calls in size $n/2$
- $\Theta(n)$ additions to compute $F_0 G_1 + F_1 G_0$
- multiplications by $x^{n/2}$ and x^n are free
- $\Theta(n)$ additions to handle overlaps

Recurrence: $T(n) = 4T(n/2) + \Theta(n)$

- $a = 4$, $b = 2$, $c = 1$ so $T(n) = \Theta(n^2)$

Not better than the naive algorithm. We do the same operations.

Karatsuba's algorithm

Idea: use the identity

$$(F_0+F_1x^{n/2})(G_0+G_1x^{n/2}) = \boldsymbol{F_0G_0}+((\boldsymbol{F_0}+\boldsymbol{F_1})(\boldsymbol{G_0}+\boldsymbol{G_1})-\boldsymbol{F_0G_0}-\boldsymbol{F_1G_1})x^{n/2}+\boldsymbol{F_1G_1}x^n$$

Analysis:

- 3 recursive calls in size $n/2$
- $\Theta(n)$ additions to compute $F_0 + F_1$ and $G_0 + G_1$
- multiplications by $x^{n/2}$ and x^n are free
- $\Theta(n)$ additions and subtractions to combine the results

Recurrence: $T(n) = 3T(n/2) + \Theta(n)$

- $a = 3$, $b = 2$, $c = 1$ so $\boldsymbol{T(n) = \Theta(n^{\log_2(3)})}$ $\log_2(3) = 1.58\ldots$

Karatsuba's algorithm

Idea: use the identity

$$(F_0+F_1x^{n/2})(G_0+G_1x^{n/2}) = \boldsymbol{F_0G_0}+((\boldsymbol{F_0}+\boldsymbol{F_1})(\boldsymbol{G_0}+\boldsymbol{G_1})-\boldsymbol{F_0G_0}-\boldsymbol{F_1G_1})x^{n/2}+\boldsymbol{F_1G_1}x^n$$

Analysis:

- 3 recursive calls in size $n/2$
- $\Theta(n)$ additions to compute $F_0 + F_1$ and $G_0 + G_1$
- multiplications by $x^{n/2}$ and x^n are free
- $\Theta(n)$ additions and subtractions to combine the results

Recurrence: $T(n) = 3T(n/2) + \Theta(n)$

- $a = 3$, $b = 2$, $c = 1$ so $\boldsymbol{T(n) = \Theta(n^{\log_2(3)})}$ $\log_2(3) = 1.58\ldots$

Remark: key idea = a formula for degree-1 polymomials that does **3** multiplications

Toom-Cook and FFT

Took-Cook:

- a family of algorithms based on similar expressions as Karatsuba
- for $k \geq 2$, $2k-1$ recursive calls in size n/k
- so $\boldsymbol{T(n) = \Theta(n^{\log_k(2k-1)})}$
- gets as close to exponent 1 as we want (but very slowly)

FFT:

- if we use complex coefficients, FFT can be used to multiply polynomials
- FFT follows the same recurrence as merge sort, $T(n) = 2T(n/2) + \Theta(n)$
- so we can multiply polynomials in $\boldsymbol{\Theta(n \log(n))}$ ops over $\mathbb{C}$

Multiplying matrices

Goal: given $\boldsymbol{A = [a_{i,j}]_{1 \le i,j \le n}}$ and $\boldsymbol{B = [b_{j,k}]_{1 \le j,k \le n}}$ compute $C = AB$

Remark: input and output size $\Theta(n^2)$, easy algorithm in $\Theta(n^3)$

```
1.  for i = 1, ..., n do
2.      for j = 1, ..., n do
3.          for k = 1, ..., n do
4.              c_{i,k} = c_{i,k} + a_{i,j} b_{j,k}
```

Divide-and-conquer

Setup: write

$$A = \begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix} \quad B = \begin{pmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{pmatrix}$$

with all $A_{i,k}, B_{i,j}$ of size $n/2 \times n/2$. Then

$$C = \begin{pmatrix} A_{1,1}B_{1,1} + A_{1,2}B_{2,1} & A_{1,1}B_{1,2} + A_{1,2}B_{2,2} \\ A_{2,1}B_{1,1} + A_{2,2}B_{2,1} & A_{2,1}B_{1,2} + A_{2,2}B_{2,2} \end{pmatrix}$$

Naively: 8 recursive calls in size $n/2 + \Theta(n^2)$ additions $\implies T(n) = \Theta(n^3)$

Goal: find a better formula for 2×2 matrices

Strassen's algorithm

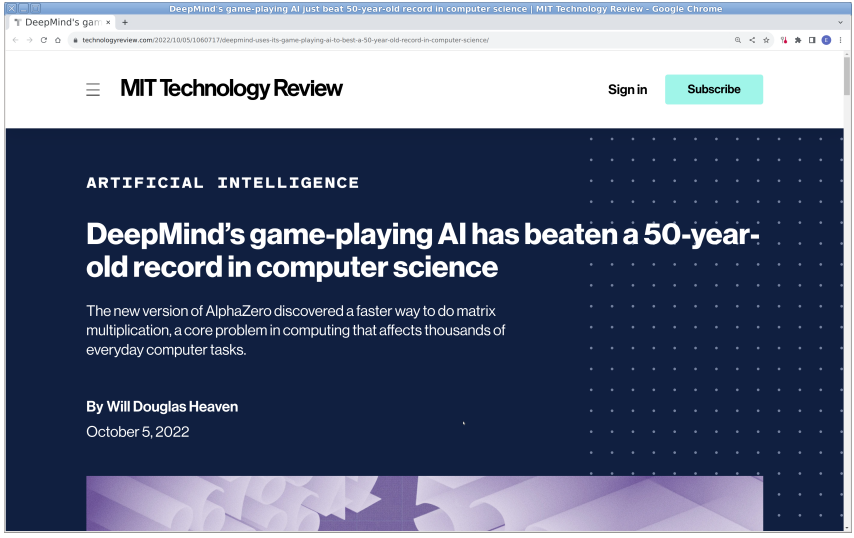
Compute

$$
\left|\begin{array}{lcl}
Q_1 &=& (A_{1,1} - A_{1,2})B_{2,2} \\
Q_2 &=& (A_{2,1} - A_{2,2})B_{1,1} \\
Q_3 &=& A_{2,2}(B_{1,1} + B_{2,1}) \\
Q_4 &=& A_{1,1}(B_{1,2} + B_{2,2}) \\
Q_5 &=& (A_{1,1} + A_{2,2})(B_{2,2} - B_{1,1}) \\
Q_6 &=& (A_{1,1} + A_{2,1})(B_{1,1} + B_{1,2}) \\
Q_7 &=& (A_{1,2} + A_{2,2})(B_{2,1} + B_{2,2})
\end{array}\right.
\quad \text{and} \quad
\left|\begin{array}{lcl}
C_{1,1} &=& Q_1 - Q_3 - Q_5 + Q_7 \\
C_{1,2} &=& Q_4 - Q_1 \\
C_{2,1} &=& Q_2 + Q_3 \\
C_{2,2} &=& -Q_2 - Q_4 + Q_5 + Q_6
\end{array}\right.
$$

Analysis: 7 recursive calls in size $n/2 + \Theta(n^2)$ additions $\implies$ $\boldsymbol{T(n) = \Theta(n^{\log_2(7)})}$

$\log_2(7) = 2.80\ldots$

Faster algorithms: AI to the rescue



What this means

Direct generalization

- an algorithm that does k multiplications for matrices of size ℓ gives $\boldsymbol{T(n) \in \Theta(n^{\log_\ell(k)})}$

What this means

Direct generalization

- an algorithm that does k multiplications for matrices of size ℓ gives $\boldsymbol{T(n) \in \Theta(n^{\log_\ell(k)})}$

One challenge: find best k for small values of ℓ

- SAT solving, gradient descent, ...
- AlphaTensor found some better values, but none beats Strassen (except for matrices over $\{0, 1\}$, with operations modulo 2)

What this means

Direct generalization

- an algorithm that does k multiplications for matrices of size ℓ gives $\boldsymbol{T(n) \in \Theta(n^{\log_\ell(k)})}$

One challenge: find best k for small values of ℓ

- SAT solving, gradient descent, ...
- AlphaTensor found some better values, but none beats Strassen (except for matrices over $\{0, 1\}$, with operations modulo 2)

Best exponent to date (using more than just divide-and-conquer)

- $O(n^{2.37188})$, improves from previous record $O(n^{2.37286})$
- galactic algorithms

Counting inversions

Goal: given an unsorted array $A[1..n]$, find the number of inversions in it.

Def: (i, j) is an inversion if $i < j$ and $A[i] > A[j]$

Example: with $A = [1, 5, 2, 6, 3, 8, 7, 4]$, we get

$$(2, 3), (2, 5), (2, 8), (4, 5), (4, 8), (6, 7), (6, 8), (7, 8)$$

Counting inversions

Goal: given an unsorted array $A[1..n]$, find the number of inversions in it.

Def: (i, j) is an inversion if $i < j$ and $A[i] > A[j]$

Example: with $A = [1, 5, 2, 6, 3, 8, 7, 4]$, we get

$$(2, 3), (2, 5), (2, 8), (4, 5), (4, 8), (6, 7), (6, 8), (7, 8)$$

Remark: we show the indices where inversions occur

Remark: easy algorithm (two nested loops) in $\Theta(n^2)$

Remark: to do better than n^2, we cannot list all inversions

Toward a divide-and-conquer algorithm

Idea (for n a power of two)

- c_ℓ = number of inversions in $A[1..n/2]$
- c_r = number of inversions in $A[n/2+1..n]$
- c_t = number of transverse inversions with $i \leq n/2$ and $j > n/2$
- return $c_\ell + c_r + c_t$

Example: with $A = [1, 5, 2, 6, 3, 8, 7, 4]$

- $c_\ell = 1$ $(2, 3)$
- $c_r = 3$ $(6, 7), (6, 8), (7, 8)$
- $c_t = 4$ $(2, 5), (2, 8), (4, 5), (4, 8)$

c_ℓ and c_r done recursively. What about c_t?

Transverse inversions

Goal: how many pairs (i, j) with $i \leq n/2$, $j > n/2$, $A[i] > A[j]$?

Example: with $A = [\mathbf{1}, \mathbf{5}, \mathbf{2}, \mathbf{6}, \mathbf{3}, \mathbf{8}, \mathbf{7}, \mathbf{4}]$

$c_t = \#i$'s greater than 3 + $\#i$'s greater than 8 + $\#i$'s greater than 7 + $\#i$'s greater than 4

or

$c_t = \#j$'s less than 1 + $\#j$'s less than 5 + $\#j$'s less than 2 + $\#j$'s less than 6

Transverse inversions

Goal: how many pairs (i, j) with $i \leq n/2$, $j > n/2$, $A[i] > A[j]$?

Example: with $A = [\mathbf{1}, \mathbf{5}, \mathbf{2}, \mathbf{6}, \mathbf{3}, \mathbf{8}, \mathbf{7}, \mathbf{4}]$

$c_t = \#i$'s greater than 3 + $\#i$'s greater than 8 + $\#i$'s greater than 7 + $\#i$'s greater than 4

or

$c_t = \#j$'s less than 1 + $\#j$'s less than 5 + $\#j$'s less than 2 + $\#j$'s less than 6

Observation: this number does not change if both sides are **sorted**

So assume that left and right are sorted after the recursive calls.

Example: starting from $[1, 5, 2, 6, 3, 8, 7, 4]$, we get

$$[\mathbf{1}, \mathbf{2}, \mathbf{5}, \mathbf{6}, \mathbf{3}, \mathbf{4}, \mathbf{7}, \mathbf{8}]$$

Enhancing mergesort

Idea: find c_t during merge.

Merge$(A[1..n])$ (both halves of A assumed sorted)

```
1.    copy A into a new array S
2.    i = 1; j = n/2 + 1;
3.    for (k ← 1; k ≤ n; k++) do
4.        if (i > n/2) A[k] ← S[j++]
5.        else if (j > n) A[k] ← S[i++]
6.        else if (S[i] < S[j]) A[k] ← S[i++]
7.        else A[k] ← S[j++]
```

Enhancing mergesort

Idea: find c_t during merge.

Merge$(A[1..n])$ (both halves of A assumed sorted)

```
1.    copy A into a new array S;  c = 0
2.    i = 1; j = n/2 + 1;
3.    for (k ← 1; k ≤ n; k++) do
4.        if (i > n/2) A[k] ← S[j++]
5.        else if (j > n) A[k] ← S[i++]; c = c + n/2
6.        else if (S[i] < S[j]) A[k] ← S[i++]; c = c + j − (n/2 + 1)
7.        else A[k] ← S[j++]
```

Example: with $[1, 2, 5, 6, 3, 4, 7, 8]$

- when we insert 1 back into A, $j = 5$ so $c = c + 0$
- when we insert 2 back into A, $j = 5$ so $c = c + 0$
- when we insert 5 back into A, $j = 7$ so $c = c + 2$
- when we insert 6 back into A, $j = 7$ so $c = c + 2$

Analysis

Enhanced merge is still $\Theta(n)$ so total remains $\Theta(n \log(n))$

Remark: $\Omega(n \log(n))$ lower bound in the comparison model (decision tree)

Closest pairs

Goal: given n points (x_i, y_i) in the plane, find a pair (i, j) that minimizes the distance

$$d_{i,j} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

Equivalent to minimize

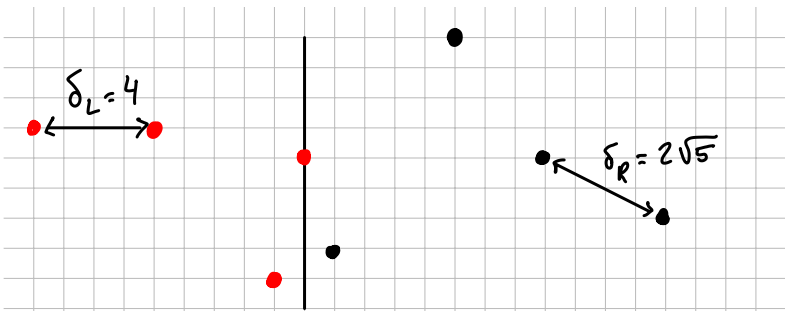
$$d_{i,j}^2 = (x_i - x_j)^2 + (y_i - y_j)^2$$

Assumption: all x_i's are pairwise distinct

Divide-and-conquer

Idea: separate the points into two halves L, R at the median x-value

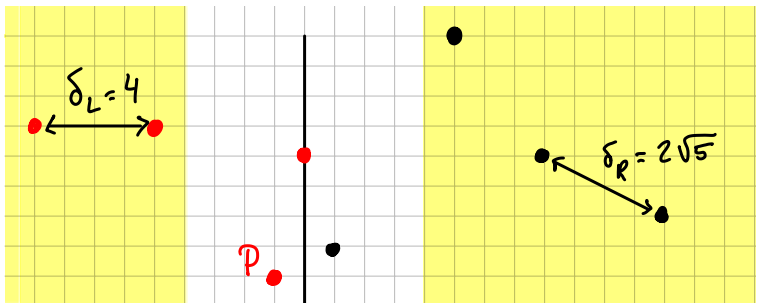
- $L =$ all $n/2$ points with $x \leq x_{\text{median}}$
- $R =$ all $n/2$ points with $x > x_{\text{median}}$
- find the closest pairs in both L and R recursively
- the closest pair is either between points in L (done), or between points in R (done), or transverse (one in L, one in R)



Finding the shortest transverse distance

Set $\delta = \min(\delta_R, \delta_R)$

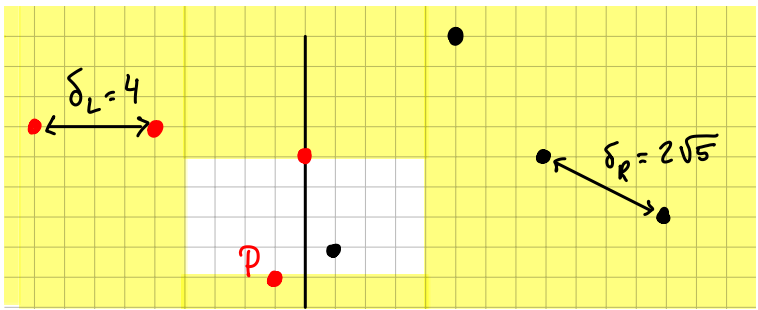
- We only need to consider transverse pairs (P, Q) with $\mathbf{dist}(\boldsymbol{P}, \boldsymbol{R}) \leq \boldsymbol{\delta}$ and $\mathbf{dist}(\boldsymbol{Q}, \boldsymbol{L}) \leq \boldsymbol{\delta}$.



Finding the shortest transverse distance

Set $\delta = \min(\delta_R, \delta_R)$

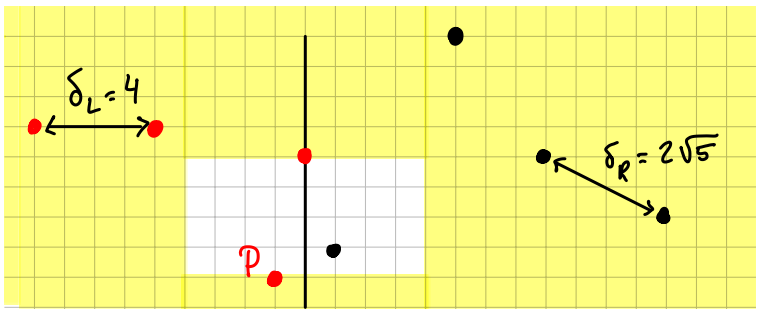
- For any $P = (x_P, y_P)$, enough to look at points with $y_P \leq y < y_P + \delta$



Finding the shortest transverse distance

Set $\delta = \min(\delta_R, \delta_R)$

- For any $P = (x_P, y_P)$, enough to look at points with $y_P \leq y < y_P + \delta$



So it is enough to check distances $d(P, Q)$ for Q in the rectangle.

How many points in the rectangle?

Claim

There are at most **8** points from our initial set (including P) in the rectangle.

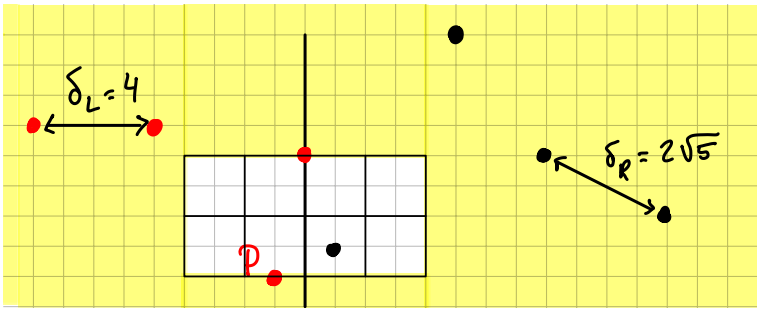
How many points in the rectangle?

Claim

There are at most **8** points from our initial set (including P) in the rectangle.

Proof. Cover the rectangle with **8** squares of side length $\delta/2$

- open on the left, closed on the right
- they overlap along lines, but it's OK



How many points in the rectangle?

Claim

There are at most **8** points from our initial set (including P) in the rectangle.

Proof. Cover the rectangle with **8** squares of side length $\delta/2$

- open on the left, closed on the right
- they overlap along lines, but it's OK

Squares on the left only contain points from L, squares on the right only contain points from R.

Consequence: in each square, only one point (either from L or R).

Data structures

Initialization: sort the points **twice**, with respect to x and to y.
One-time cost $O(n \log(n))$, before recursive calls cf kd-trees

Then: recursion

- finding the x-median is easy $O(1)$
- for the next recursive calls, split the sorted lists $O(n)$
- remove the points at distance $\geq \delta$ from the x-splitting line $O(n)$
- inspect all remaining points in increasing y-order. For each of them, compute the distance to the next 8 points and keep the min. $O(n)$

Runtime: $T(n) = 2T(n/2) + \Theta(n)$ so $T(n) \in \Theta(n \log(n))$

Beyond the master theorem: median of medians

Median: given $A[0..n-1]$, find the entry that would be at index $\lfloor n/2 \rfloor$ if A was sorted

Selection: given $A[0..n-1]$ and k in $\{0, \ldots, n-1\}$, find the entry that would be at index k if A was sorted

Known results: sorting A in $\boldsymbol{O(n \log(n))}$, or a simple randomized algorithm in expected time $\boldsymbol{O(n)}$

The selection algorithm

quick-select(A, k)
A: array of size n, k: integer s.t. $0 \leq k < n$

```
1.    p ← choose-pivot(A)
2.    i ← partition(A, p)                    i is the correct index of p
3.    if i = k then
4.        return A[i]
5.    else if i > k then
6.        return quick-select(A[0, 1, ..., i − 1], k)
7.    else if i < k then
8.        return quick-select(A[i + 1, i + 2, ..., n − 1], k − i − 1)
```

Question: how to find a pivot such that both i and $n - i - 1$ are not too large?

Median of medians

Sketch of the algorithm:

- divide A into $n/5$ groups $G_1, \ldots, G_{n/5}$ of size 5
- find the medians $m_1, \ldots, m_{n/5}$ of each group $\boldsymbol{O(n)}$
- pivot p is the median of $[m_1, \ldots, m_{n/5}]$ $\boldsymbol{T(n/5)}$

Claim

With this choice of p, the indices $\boldsymbol{i}$ and $\boldsymbol{n-i-1}$ are at most $\boldsymbol{7n/10}$

Median of medians

Sketch of the algorithm:

- divide A into $n/5$ groups $G_1, \ldots, G_{n/5}$ of size 5
- find the medians $m_1, \ldots, m_{n/5}$ of each group $\boldsymbol{O(n)}$
- pivot p is the median of $[m_1, \ldots, m_{n/5}]$ $\boldsymbol{T(n/5)}$

Claim

With this choice of p, the indices $\boldsymbol{i}$ and $\boldsymbol{n-i-1}$ are at most $\boldsymbol{7n/10}$

Proof

- **half** of the m_i's are greater than p $\boldsymbol{n/10}$
- for each m_i, there are **3** elements in G_i greater than or equal to m_i
- so **at least** $\boldsymbol{3n/10}$ elements greater than p
- so **at most** $\boldsymbol{7n/10}$ elements less than p
- so i is **at most** $\boldsymbol{7n/10}$. Same thing for $n-i-1$

Median of medians

Sketch of the algorithm:

- divide A into $n/5$ groups $G_1, \ldots, G_{n/5}$ of size 5
- find the medians $m_1, \ldots, m_{n/5}$ of each group $\boldsymbol{O(n)}$
- pivot p is the median of $[m_1, \ldots, m_{n/5}]$ $\boldsymbol{T(n/5)}$

Claim

With this choice of p, the indices $\boldsymbol{i}$ and $\boldsymbol{n-i-1}$ are at most $\boldsymbol{7n/10}$

Consequence: the runtime $T(n)$ satisfies

$$T(n) \leq T(n/5) + T(7n/10) + O(n)$$

Claim

This gives $\boldsymbol{T(n) \in O(n)}$

Establishing $T(n)$ by guess-and-prove

Suppose more precisely $\boldsymbol{T(n) \leq T(n/5) + T(7n/10) + \alpha n}$, α constant.

Establishing $T(n)$ by guess-and-prove

Suppose more precisely $\boldsymbol{T(n) \leq T(n/5) + T(7n/10) + \alpha n}$, α constant.

Guess: $\boldsymbol{T(n) \leq n}$. If true for $n/5$ and $7n/10$, we get

$$T(n) \leq n/5 + 7n/10 + \alpha n = (\alpha + 9/10)n$$

Establishing $T(n)$ by guess-and-prove

Suppose more precisely $\boldsymbol{T(n) \leq T(n/5) + T(7n/10) + \alpha n}$, α constant.

Guess: $\boldsymbol{T(n) \leq n}$. If true for $n/5$ and $7n/10$, we get

$$T(n) \leq n/5 + 7n/10 + \alpha n = (\alpha + 9/10)n$$

Guess: $\boldsymbol{T(n) \leq kn}$, k TBD. If true for $n/5$ and $7n/10$, we get

$$T(n) \leq kn/5 + 7kn/10 + \alpha n = (\alpha + 9k/10)n.$$

Want k such that $\boldsymbol{\alpha + 9k/10 = k}$: take $k = 10\alpha$.

Establishing $T(n)$ by guess-and-prove

Suppose more precisely $\boldsymbol{T(n) \leq T(n/5) + T(7n/10) + \alpha n}$, α constant.

Guess: $\boldsymbol{T(n) \leq n}$. If true for $n/5$ and $7n/10$, we get

$$T(n) \leq n/5 + 7n/10 + \alpha n = (\alpha + 9/10)n$$

Guess: $\boldsymbol{T(n) \leq kn}$, k TBD. If true for $n/5$ and $7n/10$, we get

$$T(n) \leq kn/5 + 7kn/10 + \alpha n = (\alpha + 9k/10)n.$$

Want k such that $\boldsymbol{\alpha + 9k/10 = k}$: take $k = 10\alpha$.

So $T(n)$ is in $O(n)$, but with a relatively large constant.

Final remark

Why not median of three?

- we do $n/3$ groups of 3 and find their medians $m_1, \ldots, m_{n/3}$ — $O(n)$
- p is the median of $[m_1, \ldots, m_{n/3}]$ — $T(n/3)$
- half of the m_i's are greater than p — $n/6$
- in each group, 2 elements greater than or equal to m_i
- so overall at least $n/3$ elements greater than p
- so $i \leq 2n/3$, and $n - 1 - i \leq 2n/3$

Final remark

Why not median of three?

- we do $n/3$ groups of 3 and find their medians $m_1, \ldots, m_{n/3}$ — $O(n)$
- p is the median of $[m_1, \ldots, m_{n/3}]$ — $T(n/3)$
- half of the m_i's are greater than p — $n/6$
- in each group, 2 elements greater than or equal to m_i
- so overall at least $n/3$ elements greater than p
- so $i \leq 2n/3$, and $n - 1 - i \leq 2n/3$

Recurrence: $T(n) \leq T(n/3) + T(2n/3) + O(n)$

What does this give for $T(n)$?