COMP 251

Algorithms & Data Structures (Winter 2021)

Algorithm Paradigms – Divide and Conquer

School of Computer Science
McGill University

Slides of (Comp321 ,2021), Langer (2014), slides by K. Wayne & Snoeyink and (Kleinberg & Tardos, 2005)

Outline

- Complete Search
- Divide and Conquer.
  - Introduction.
  - Examples.
- Dynamic Programming.
- Greedy.

Algorithmic Paradigms – Divide and Conquer

- It is a problem solving paradigm where we try to make a problem simpler by ‘dividing’ it into smaller parts and ‘conquering’ them.
- Recursive in structure
  - ***Divide*** the problem into sub-problems that are similar to the original but smaller in size
    - Usually by half or nearly half.
  - ***Conquer*** the sub-problems by solving them recursively. If they are small enough, just solve them in a straightforward manner.
  - ***Combine*** the solutions to create a solution to the original problem

Decrease and Conquer

- Sometimes we're not actually dividing the problem into many subproblems, but only into one smaller subproblem
- Usually called decrease and conquer
- The most common example of this is binary search O(log n).

  - Given a sorted array of elements.
    1. Base case: the array is empty, return false
    2. Compare x to the element in the middle of the array
    3. If it's equal, then we found x and we return true
    4. If it's less, then x must be in the left half of the array
       4.1 Binary search the element (recursively) in the left half
    5. If it's greater, then x must be in the right half of the array
       5.1 Binary search the element (recursively) in the right half

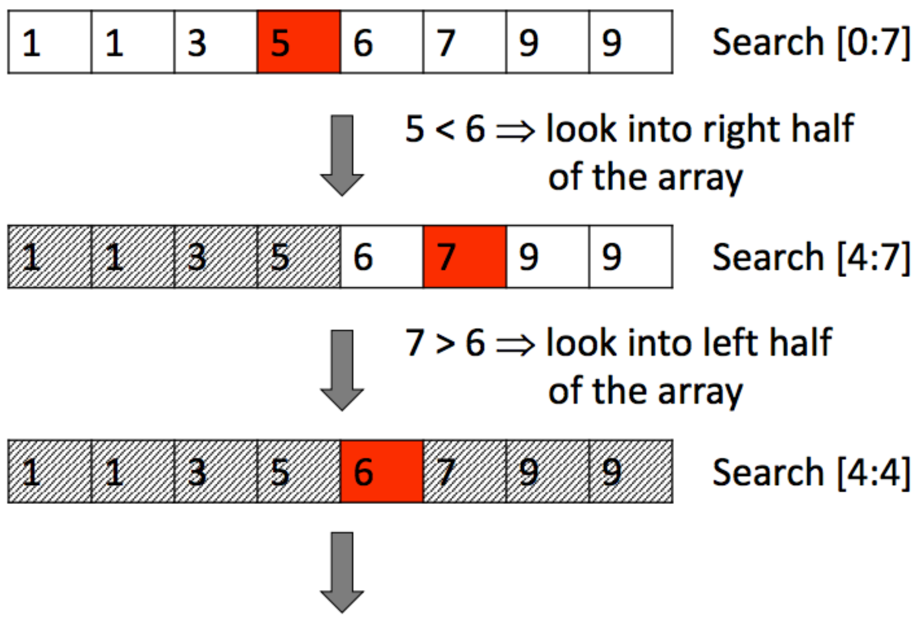
Decrease and Conquer

Example: Does the following **sorted** array A contains the number 6?

A =

1	1	3	5	6	7	9	9

Call: `binarySearch(A, 0, 7, 6)`



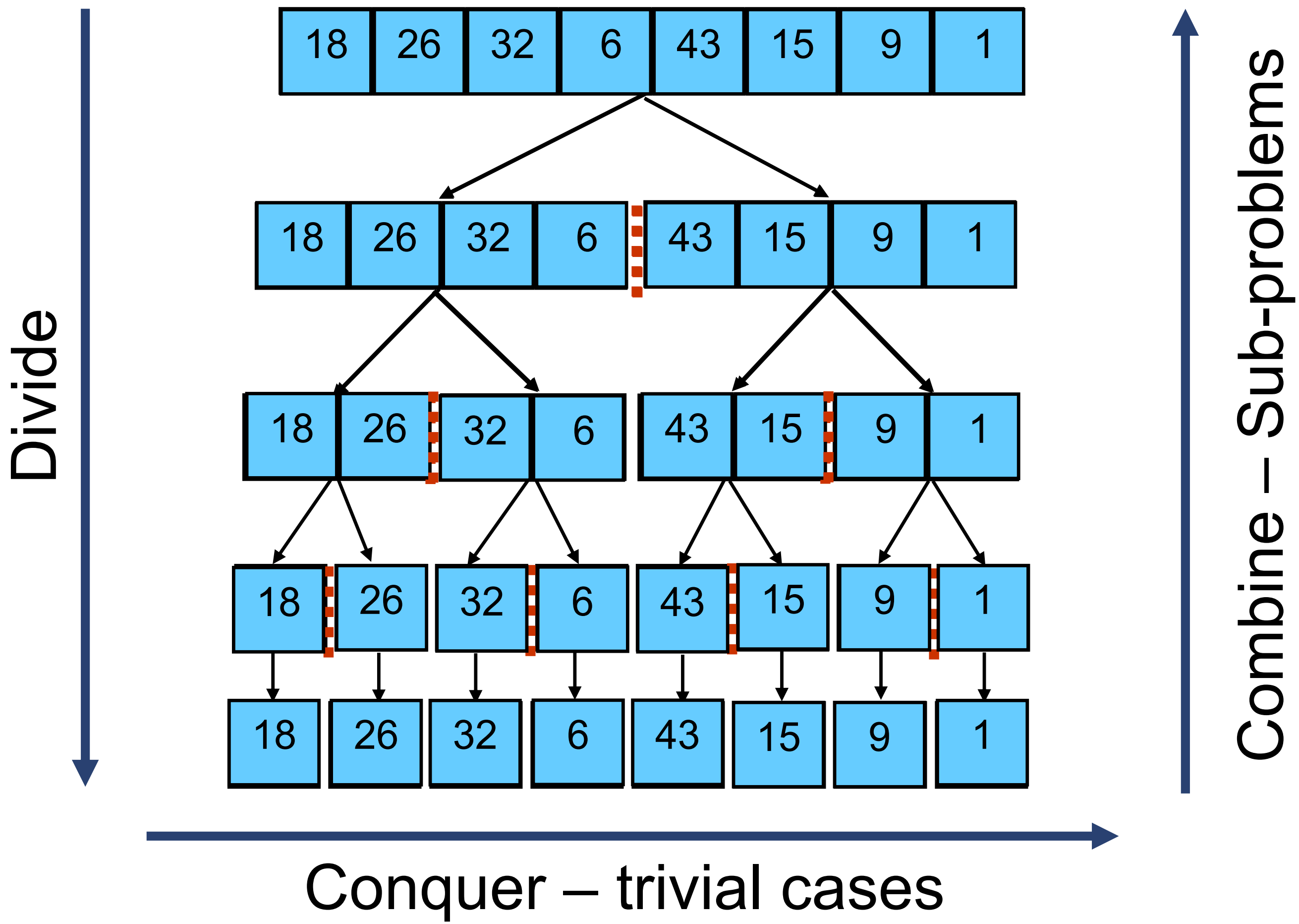
6 is found. Return 4 (index)

Divide and Conquer – Merge Sort

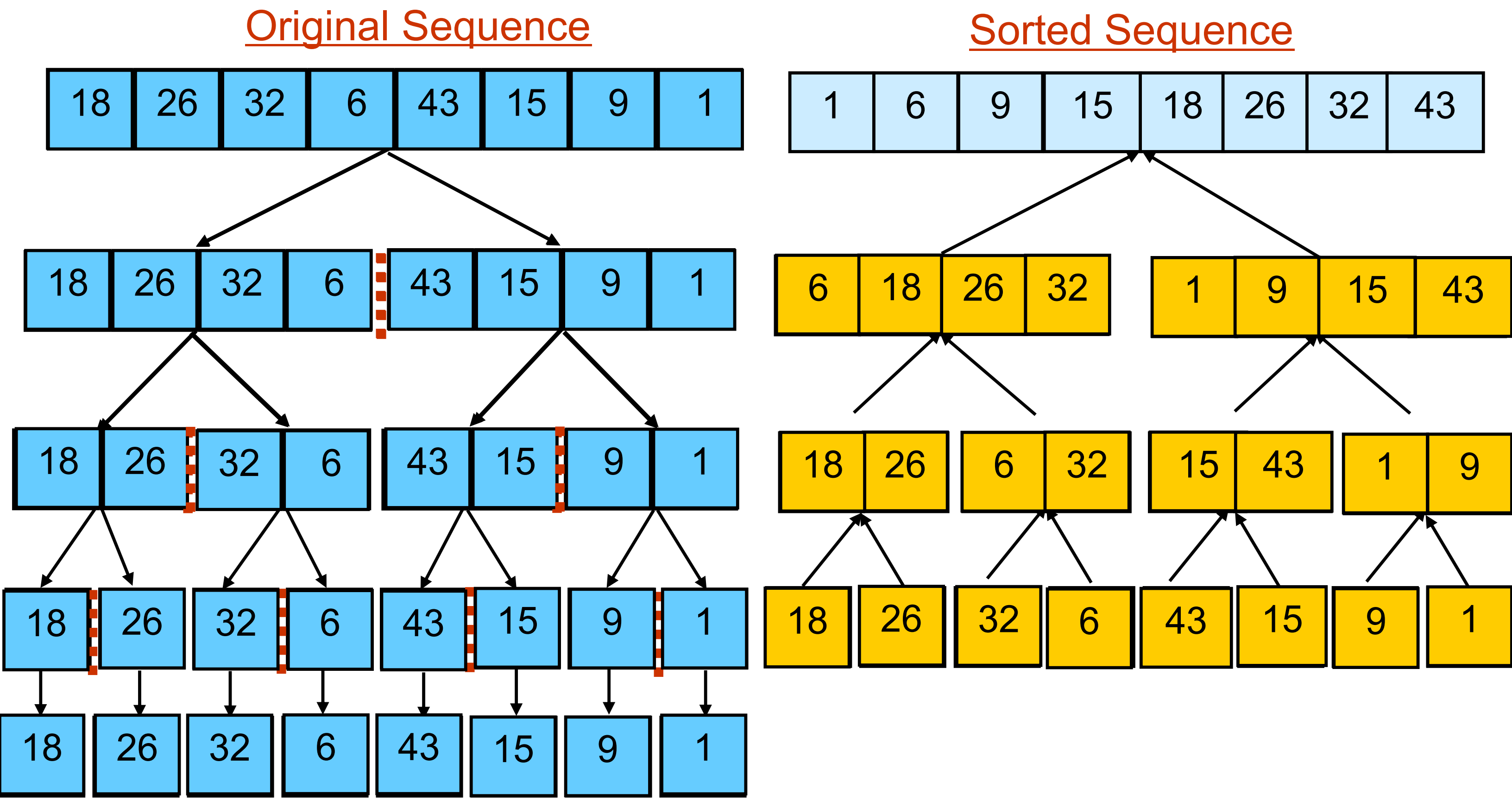
***Sorting Problem*:** Sort a sequence of *n* elements into non-decreasing order.

- ***Divide*:** Divide the *n*-element sequence to be sorted into two subsequences of *n/2* elements each
- ***Conquer:*** Sort the two subsequences recursively using merge sort.
- ***Combine*:** Merge the two sorted subsequences to produce the sorted answer.

Divide and Conquer – Merge Sort



Divide and Conquer – Merge Sort



MergeSort(A, p, r)

INPUT: **a sequence of *n* numbers stored in array A**

OUTPUT: **an ordered sequence of *n* numbers**

left right

***MergeSort* (*A*, *p*, *r*)** // sort $A[p..r]$ by divide & conquer

```
if p < r
    then q ← ⌊(p+r)/2⌋
        MergeSort (A, p, q)
        MergeSort (A, q+1, r)
        Merge (A, p, q, r) // merges A[p..q] with A[q+1..r]
```

Initial Call: MergeSort(A, 1, n)

Merge(A, p, q, r)

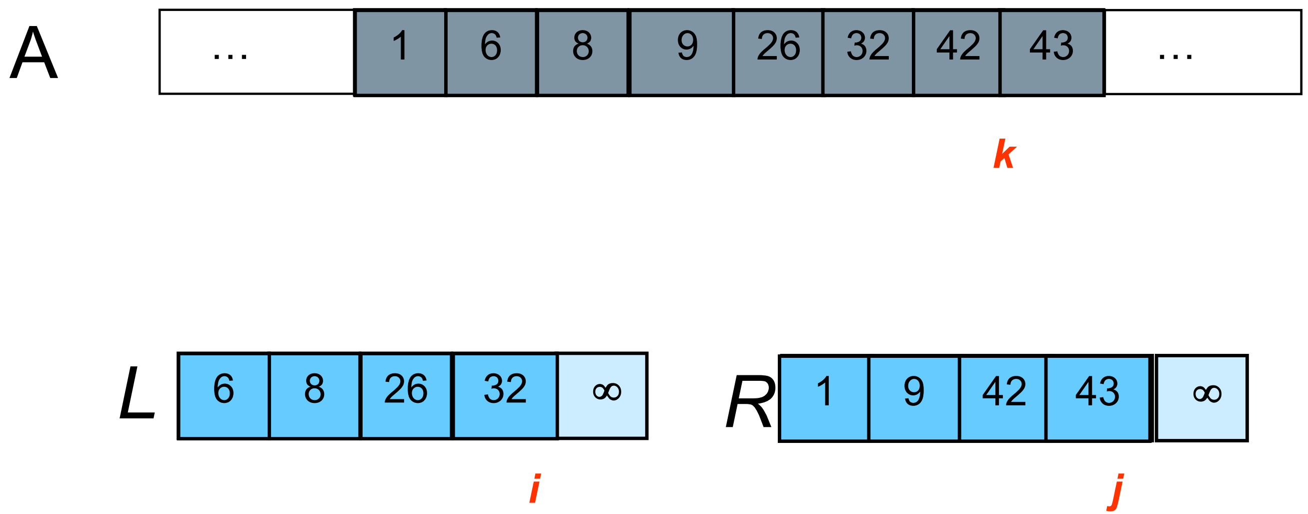
```
Merge(A, p, q, r)
    n1 ← q – p + 1
    n2 ← r – q
    for i ← 1 to n1
        do L[i] ← A[p + i – 1]
    for j ← 1 to n2
        do R[j] ← A[q + j]
    L[n1+1] ← ∞
    R[n2+1] ← ∞
    i ← 1
    j ← 1
    for k ←p to r
        do if L[i] ≤ R[j]
            then A[k] ← L[i]
                 i ← i + 1
            else A[k] ← R[j]
                 j ← j + 1
```

Input: Array containing sorted subarrays $A[p..q]$ and $A[q+1..r]$.

Output: Merged sorted subarray in $A[p..r]$.

Sentinels, to avoid having to check if either subarray is fully copied at each step.

Merge - Example



Merge - Correctness

Merge(*A*, *p*, *q*, *r*)

```
n1 ← q – p + 1
n2 ← r – q
for i ← 1 to n1
    do L[i] ← A[p + i – 1]
for j ← 1 to n2
    do R[j] ← A[q + j]
L[n1+1] ← ∞
R[n2+1] ← ∞
i ← 1
j ← 1
for k ←p to r
    do if L[i] ≤ R[j]
        then A[k] ← L[i]
                 i ← i + 1
        else A[k] ← R[j]
                 j ← j + 1
```

Loop Invariant property (main *for* loop)

- At the start of each iteration of the `for` loop, subarray $A[p..k-1]$ contains the $k-p$ smallest elements of L and R in sorted order.
- $L[i]$ and $R[j]$ are the smallest elements of L and R that have not been copied back into A.

Initialization:

Before the first iteration:

•$A[p..k-1]$ is empty, k = p => k – p = 0.

•$i = j = 1$.

•$L[1]$ and $R[1]$ are the smallest elements of L and R not copied to A.

Merge - Correctness

Merge(*A*, *p*, *q*, *r*)

```
1.  n1 ← q – p + 1
2.  n2 ← r – q
3.  for i ← 1 to n1
4.      do L[i] ← A[p + i – 1]
5.  for j ← 1 to n2
6.      do R[j] ← A[q + j]
7.  L[n1+1] ← ∞
8.  R[n2+1] ← ∞
9.  i ← 1
10. j ← 1
11. for k ←p to r
12.     do if L[i] ≤ R[j]
13.         then A[k] ← L[i]
14.                 i ← i + 1
15.         else A[k] ← R[j]
16.                 j ← j + 1
```

Maintenance:

Case 1: $L[i] \leq R[j]$

- By LI, A contains $k - p$ smallest elements of L and R in sorted order.
- By LI, $L[i]$ and $R[j]$ are the smallest elements of L and R not yet copied into A.
- Line 13 results in A containing $p - k + 1$ smallest elements (again in sorted order). Incrementing i and k reestablishes the LI for the next iteration.

Case 2: Similar arguments with $L[i] > R[j]$

Termination:

- On termination, $k = r + 1$. By LI, *A[p..k-1], which is A[p..r],* contains $k\text{-}p=r-p+1$ smallest elements of L and R in sorted order.
- L and R together *contain* $n_1 + n_2 + 2 = r - p + 3$ elements including the two sentinels. All but the two largest (i.e., sentinels) have been copied in A.

MergeSort - Analysis

- Running time ***T*(*n*)** of Merge Sort:
- Divide: computing the middle takes O(1)
- Conquer: solving 2 subproblems takes $2T(n/2)$
- Combine: merging n elements takes $O(n)$
- Total:

$$T(n) = \begin{cases} O(1) & \text{if } n = 1 \\ 2T(n/2) + O(n) & \text{if } n > 1 \end{cases}$$

$\Rightarrow T(n) = O(n \lg n)$

In general – Analysis - Recurrence

Trivial case of size <= constant

$$T(n) = \begin{cases} \Theta(1) & \text{if } n \le c\,, \\ aT(n/b) + D(n) + C(n) & \text{otherwise}\,. \end{cases}$$

# of subproblems

size of subproblems

divide time

combine time

Solving recurrences

- ***Substitution method***: we guess a bound and then use mathematical induction to prove that our guess is correct.
- ***Recursion-tree method:*** converts the recurrence into a tree whose nodes represent the costs incurred at various levels of the recursion. We use techniques for bounding summations to solve the recurrence.
- ***Master method***: provides bounds for recurrences of the form.

$$T(n) = aT(n/b) + f(n)$$

where $a \geq 1$, $b > 1$, and $f(n)$ is a given function

MergeSort – Substitution method

Proposition. If $T(n)$ satisfies the following recurrence, then $T(n) = n \log_2 n$.

assuming n is a power of 2

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ 2\,T(n/2) + n & \text{otherwise} \end{cases}$$

Pf 2. [by induction on n]

- Base case: when $n = 1$, $T(1) = 0$.
- Inductive hypothesis: assume $T(n) = n \log_2 n$.
- Goal: show that $T(2n) = 2n \log_2 (2n)$.

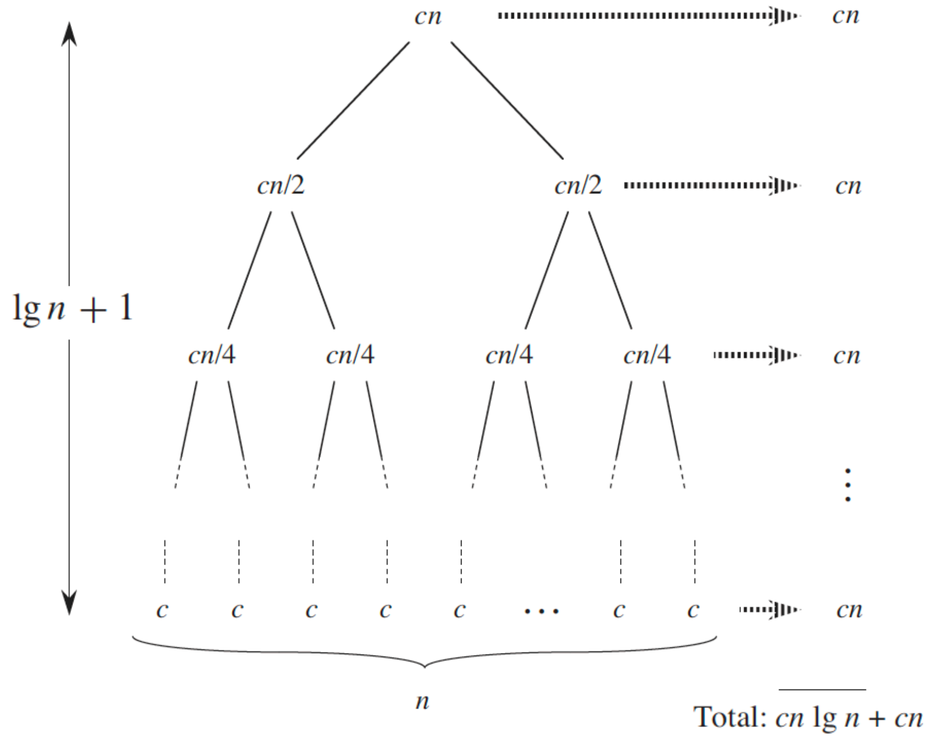
$$\boxed{\begin{gathered}\log_2 2n = \log_2 2 + \log_2 n \\ \log_2 2n - 1 = \log_2 n\end{gathered}}$$

$$\begin{aligned} T(2n) &= 2\,T(n) + 2n \\ &= 2\,n \log_2 n + 2n \\ &= 2\,n\,(\log_2 (2n) - 1) + 2n \\ &= 2\,n \log_2 (2n). \quad \blacksquare \end{aligned}$$

MergeSort – Recursion Tree

$$T(n) = \begin{cases} c & \text{if } n = 1\,, \\ 2T(n/2) + cn & \text{if } n > 1\,, \end{cases}$$

assuming n is a power of 2

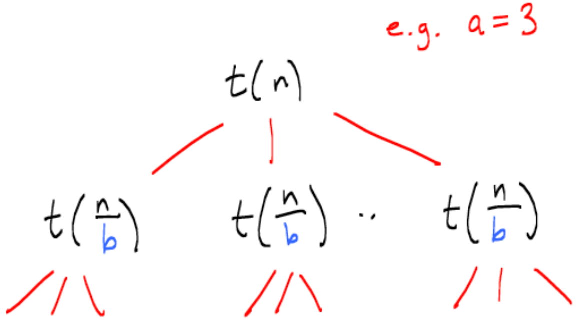


Recursion Tree

Suppose we have a divide and conquer algorithm that gives a recurrence :

$$\boxed{t(n) = a \, t\left(\frac{n}{b}\right) + c\, n^d}$$

Notice that a, b and d are independent of n



- a is the number of subproblems
- $\frac{n}{b}$ is the size of each subproblem
- n^d is the overhead for problem of size n (to partition and combine solutions)

Note that n^d represents the work done outside the recursive call

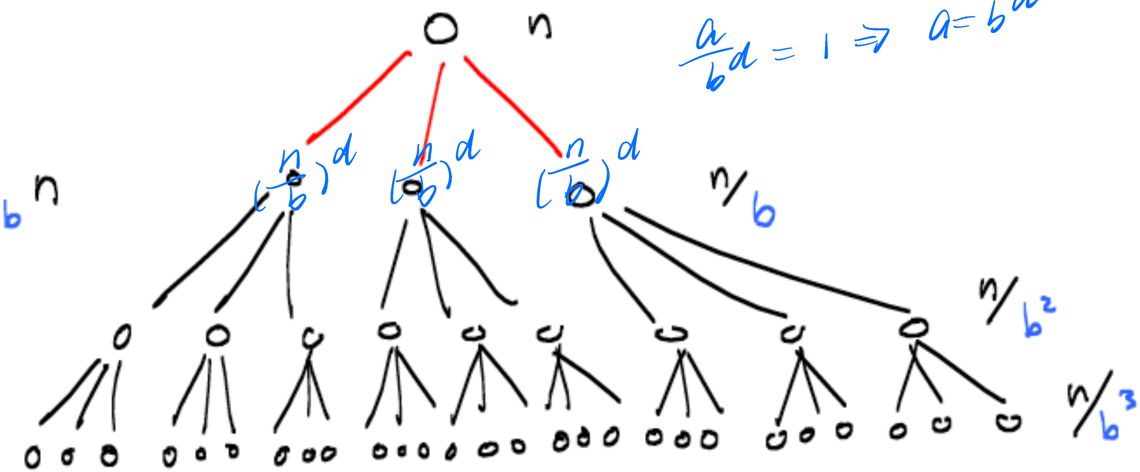
I'll set $c = 1$.

Recursion Tree

$$t(n) = a\, t\left(\frac{n}{b}\right) + c\, n^d$$

$\frac{a}{b^d} = 1 \Rightarrow a = b^d$

height of call tree $= \log_b n$



number of leaves. $= a^{\log_b n}$

Recursion stops at the base case, typically when problem size is a small number

Recursion Tree – Good VS Evil

$$\boxed{t(n) = a \; t\left(\frac{n}{b}\right) + c \, n^d}$$

assume $c=1$

Tim Roughgarden :

" the forces of good and evil "

good - the size of each subproblem shrinks with each recursive call $(b>1)$

evil - the number of subproblems increases at each level of the call tree. $(a>1)$

Let’s the battle begin (supplemental)

- But first lets define (recall) the allowed ‘super powers’.
- Geometric series power (convergence power).
  - Sum of a number of terms that have a constant ratio between successive terms.

$$\frac{1}{1}+\frac{1}{2}+\frac{1}{4}+\frac{1}{8}+\frac{1}{16}+\ \ldots\ldots\ldots$$

  - In general:

$$S_n=\sum_{k=0}^{n} r^k = 1+r+r^2+\cdots+r^n$$

  - Multiplying both sides by r gives.

$$\mathrm{r}S_n=\ r+r^2+r^3+\cdots+r^{n+1}$$

Let’s the battle begin (supplemental)

- Geometric series (convergence power).
  - Substracting the two previous equations.

$$(1-r)S_n = (1+r+r^2+\cdots+r^n) - (r+r^2+r^3+\cdots+r^{n+1})$$

$$(1-r)S_n = 1 - r^{n+1}$$

$$\boxed{S_n = \sum_{k=0}^{n} r^k = \frac{1 - r^{n+1}}{1 - r}}$$

  - For -1 < r < 1, the sum converges as $n \to \infty$, in which case

$$\boxed{S_\infty = \sum_{k=0}^{\infty} r^k = \frac{1}{1 - r}}$$

Let's the battle begin (supplemental)

- Exponents and logs (manipulation power).

$$x^{(yz)} = (x^y)^z \neq x^{(y^z)}$$

e.g. $$x^{2\cdot 3} = (x^2)^3 \neq x^{(2^3)}$$

$$= (x^2)(x^2)(x^2) \qquad = x^8$$

$$= x^6$$

e.g $$(b^d)^{\log_b n} = b^{d \log_b n}$$

$$= (b^{\log_b n})^d$$

$$= n^d$$

Let's the battle begin (supplemental)

- Exponents and logs (manipulation power).

for any $a, b, x > 0$

$$\log_b x = \log_b a \cdot \log_a x$$

Why?

$$x \equiv a^{\log_a x}$$

take $\log_b$ of both sides

$$\log_b x = \log_b \left(a^{\log_a x} \right)$$

$$= \log_a x \cdot \log_b a$$

Let’s the battle begin (supplemental)

- Exponents and logs (manipulation power).

Claim: $a^{\log_b n} = n^{\log_b a}$

Proof:

$$
\begin{aligned}
a^{\log_b n} &= a^{\log_b a \, \log_a n} \\
&= \left(a^{\log_a n}\right)^{\log_b a} \\
&= n^{\log_b a}
\end{aligned}
$$

$\frac{n}{n} = 1$ $\frac{n}{b^n} = 1$

Good VS Bad – battle

Assume $n = b^k$ for simplicity

$$t(n) = a\, t\left(\frac{n}{b}\right) + n^{a}$$

$$= a\left[a\, t\left(\frac{n}{b^2}\right) + \left(\frac{n}{b}\right)^d \right] + n^d$$

level 2

$$= a^2\, t\left(\frac{n}{b^2}\right) + a\left(\frac{n}{b}\right)^d + n^d$$

$$a\, t\left(\frac{n}{b^3}\right) + \left(\frac{n}{b^2}\right)^d$$

level 3

$$= a^3\, t\left(\frac{n}{b^3}\right) + a^2\left(\frac{n}{b^2}\right)^d + a\left(\frac{n}{b}\right)^d + n^d$$

Good VS Bad – battle

level k

$$= a^k \, t\left(\frac{n}{b^k}\right) + \sum_{i=0}^{k-1} a^i \left(\frac{n}{b^i}\right)^d$$

level $\log_b n$

$$= a^{\log_b n} \, t(1) + \sum_{i=0}^{\log_b n - 1} a^i \underbrace{\left(\frac{n}{b^i}\right)^d}$$

number of leaves

work at each leaf

number of internal nodes at level i

work at each internal node at level i

Good VS Bad – battle

$$t(n) = a^{\log_b n}\, t(1) + \sum_{i=0}^{\log_b n - 1} a^i \left(\frac{n}{b^i}\right)^d$$

$$= n^d \sum_{i=0}^{\log_b n} \underbrace{\left(\frac{a}{b^d}\right)}\,^i$$

Note that the battle is this ratio.

Assume $t(1) = 1$, and note $\frac{n}{b^{\log_b n}} = 1$.

Good VS Bad – battle possible results

$$t(n) = n^d \sum_{i=0}^{\log_b n} \left(\frac{a}{b^d}\right)^i$$

- If $a < b^d$ (good wins)
  - The amount of work is decreasing with the recursion level i.
  - Worst case is in the root (i.e., $i = 0$)
    - Might expect $O(n^d)$ => The work n^d of the root dominates
- If $a = b^d$ (there is a tie)
  - The amount of work is the same at every recursion level i.
  - All levels have the same 'worst' case.
    - Might expect $O(n^d \log n)$ => n^d for all the *log* levels
- If $a > b^d$ (evil wins)
  - The amount of work is increasing with the recursion level i.
  - Worst case is in the leaves (i.e., $i = \log_b n$)
    - Might expect => $O(n^{\log(a)})$ => O(#leaves) because leaves dominates

Good VS Bad – battle possible results

$$t(n) = n^d \sum_{i=0}^{\log_b n} \left(\frac{a}{b^d}\right)^i$$

geometric series

$$\sum_{i=0}^{k} r^i$$

three cases

1.) $r = 1$
2.) $r < 1$
3.) $r > 1$

If $a = b^d$ (there is a tie)
If $a < b^d$ (good wins)
If $a > b^d$ (evil wins)

Case 1 (r = 1) : a = b^d

Here we have the same amount of work at each level.

$$1 + r + r^2 + r^3 + \dots r^k$$

$$= 1 + 1 + 1 + 1 + \dots 1$$

$$= k + 1$$

$$= \log_b n + 1$$

$$\Rightarrow \quad t(n) = O(n^d \log_b n)$$

$$t(n) = n^d \underbrace{\sum_{i=0}^{\log_b n} \left(\frac{a}{b^d}\right)^i}_{\sum_{i=0}^{k} r^i}$$

Case 1 (r = 1) : $a = b^d$

e.g. mergesort

$$t(n) = a\, t\left(\frac{n}{b}\right) + c n^d$$

$a = 2$, $b = 2$, $d = 1$

$$t(n) = O(n^d \log_b n) = O(n \log_2 n)$$

The same amount of total work done at each level i, namely $O(n)$.

Case 2 (r < 1) : a < b^d

Here we have a decreasing amount of work at each level.

$$1 + r + r^2 + r^3 + \dots r^k$$

$$= \frac{1 - r^{k+1}}{1 - r}$$

$$< \frac{1}{1-r}, \quad \text{if } r < 1$$

$= \text{constant}$ (independent of n)

$$\Rightarrow \quad t(n) = O(n^d)$$

$$t(n) = n^d \underbrace{\sum_{i=0}^{\log_b n} \left(\frac{a}{b^d}\right)^i}_{\sum_{i=0}^{k} r^i}$$

Case 3 ($r > 1$) : $a > b^d$

Here we have an increasing amount of work to do at each level.

The leaves dominate.

$$1 + r + r^2 + r^3 + \dots r^k$$

$$= \frac{r^{k+1} - 1}{r - 1}$$

$< c\, r^k$, for some c which depends on r

$$t(n) = n^d \underbrace{\sum_{i=0}^{\log_b n} \left(\frac{a}{b^d}\right)^i}_{\sum_{i=0}^{k} r^i}$$

Case 3 ($r > 1$) : $a > b^d$

$$r^k = \left(\frac{a}{b^d}\right)^k$$

let $k = \log_b n$

$$= \left(\frac{a}{b^d}\right)^{\log_b n}$$

$$= a^{\log_b n} / (b^d)^{\log_b n}$$

$$= n^{\log_b a} / n^d$$

See Supplemental Slide

$$t(n) = n^d \underbrace{\sum_{i=0}^{\log_b n} \left(\frac{a}{b^d}\right)^i}_{\sum_{i=0}^{k} r^i}$$

Case 3 (r > 1) : a > b^d

$$t(n) = n^d \sum_{i=0}^{\log_b n} r^i$$

$$< n^d \, c \, r^{\log_b n}$$

$$= n^d \, c \left(\frac{a}{b^d}\right)^{\log_b n}$$

$$< n^d \cdot c \, \frac{n^{\log_b a}}{n^d}$$

$$= c \, n^{\log_b a}$$

$$t(n) = n^d \underbrace{\sum_{i=0}^{\log_b n} \left(\frac{a}{b^d}\right)^i}_{\sum_{i=0}^{k} r^i}$$

Master Method (summary)

$$t(n) = a\, t\left(\frac{n}{b}\right) + n^d, \quad t(1) = 1$$

same work at each level

root dominates

$$t(n) \text{ is } \begin{cases} O(n^d \log_b n), & a = b^d \\ O(n^d), & a < b^d \\ O(n^{\log_b a}), & a > b^d \end{cases}$$

leaves dominate

Master Method (summary)

$$t(n) = a\, t\left(\frac{n}{b}\right) + n^d, \quad t(1) = 1$$

same work at each level

root dominates

$$t(n) \text{ is } \begin{cases} O(n^d \log_b n), & a = b^d \\ O(n^d), & a < b^d \\ O(n^{\log_b a}), & a > b^d \end{cases}$$

$\frac{a}{b^d} = 1$

leaves dominate

- Limitations:
  - Defined for worst case.
  - Sub-problems need to have the same size.
  - Applicable to recursions of divide-and-conquer solutions
  - Etc
  - Etc

Master theorem

- Goal: Recipe for solving common recurrences.

$$T(n) = aT(n/b) + f(n)$$

a = # subproblems

f(n) = work to divide/merge subproblems

b = factor by which the subproblem size decreases

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$. bad wins
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$. tie
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$. ■

good wins

Note that the three cases do not cover all the possibilities for $f(n)$.

Master theorem – Case 1

Master theorem. Suppose that $T(n)$ is a function on the nonnegative integers that satisfies the recurrence

$$T(n) = a\,T\left(\frac{n}{b}\right) + f(n)$$

where n/b means either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Let $k = \log_b a$. Then,

Case 1. If $f(n) = O(n^{k-\varepsilon})$ for some constant $\varepsilon > 0$, then $T(n) = \Theta(n^k)$.

Ex. $T(n) = 3\,T(n/2) + n$.

- $a = 3,\ b = 2,\ f(n) = n,\ k = \log_2 3$.
- $T(n) = \Theta(n^{\lg 3})$.

The formula works with $\varepsilon = \log_2 3 - 1 > 0$

$$f(n) = n = O\left(n^{\log_2 3 - (\log_2 3 - 1)}\right)$$

Master theorem – Case 2

Master theorem. Suppose that $T(n)$ is a function on the nonnegative integers that satisfies the recurrence

$$T(n) = a\,T\left(\frac{n}{b}\right) \;+\; f(n)$$

where n/b means either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Let $k = \log_b a$. Then,

Case 2. If $f(n) = \Theta(n^k \log^p n)$, then $T(n) = \Theta(n^k \log^{p+1} n)$.

Ex. $T(n) = 2\,T(n/2) + \Theta(n \log n)$.

- $a = 2,\; b = 2,\; f(n) = n \log n,\; k = \log_2 2 = 1,\; p = 1.$
- $T(n) = \Theta(n \log^{2} n)$.

$$f(n) = \Theta(n \log n) = \Theta(n^{\log_2 2} \log n)$$

Master theorem – Case 3

Master theorem. Suppose that $T(n)$ is a function on the nonnegative integers that satisfies the recurrence

$$T(n) = a\,T\left(\frac{n}{b}\right) + f(n)$$

where n/b means either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Let $k = \log_b a$. Then,

regularity condition holds
if f(n) = $\Theta(n^{k+\varepsilon})$

Case 3. If $f(n) = \Omega(n^{k+\varepsilon})$ for some constant $\varepsilon > 0$ and if $a\,f(n/b) \le c\,f(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$.

Ex. $T(n) = 3\,T(n/4) + n^5$.

- $a = 3,\ b = 4,\ f(n) = n^5,\ k = \log_4 3$.
- $T(n) = \Theta(n^5)$.

1st property satisfied with $\varepsilon = 4 - \log_4 3$

$$f(n) = n^5 = \Omega(n^{\log_4 3 + (4 - \log_4 3)})$$

2nd property satisfied with $c = \frac{3}{4}$

$$3 \cdot \left(\frac{n}{4}\right)^5 \le c \cdot n^5$$

Master theorem – Applications

$k = \log_2 1 = 0; f(n) = 2^n$
$2^n = \Omega\left(n^{0+\log 2}\right)$
$1 \cdot 2^{\frac{n}{2}} \leq \frac{1}{2} \cdot 2^n$

T(n) = 3 * T(n/2) + n^2
⇒ T(n) = $\Theta(n^2)$ (case 3)

$k = \log_2 3; f(n) = n^2$
$n^2 = \Omega\left(n^{\log_2 3+(2-\log_2 3)}\right)$
$3 \cdot \left(\frac{n}{2}\right)^2 \leq \frac{3}{4} \cdot n^2$

T(n) = T(n/2) + 2^n
⇒ T(n) = $\Theta(2^n)$ (case 3)

T(n) = 16 * T(n/4) + n
⇒ T(n) = $\Theta(n^2)$ (case 1)

T(n) = 2 * T(n/2) + n log n
⇒ T(n) = n $\log^2$ n (case 2)

T(n) = 2^n * T(n/2) + n^n
⇒ Does not apply!!

$k = \log_4 16 = 2; f(n) = n$
$n = O(n^{2-1})$

$k = \log_2 2 = 1; f(n) = n \log n$
$n \log n = \Theta(n^1 log^1 n)$

Master theorem – Other variants – Akra-Bazzi

Desiderata. Generalizes master theorem to divide-and-conquer algorithms where subproblems have substantially different sizes.

Theorem. [Akra-Bazzi] Given constants $a_i > 0$ and $0 < b_i \leq 1$, functions $h_i(n) = O(n / \log^2 n)$ and $g(n) = O(n^c)$, if the function $T(n)$ satisfies the recurrence:

$$T(n) = \sum_{i=1}^{k} \underbrace{a_i}_{} \, T(\underbrace{b_i n}_{a_i \text{ subproblems of size } b_i n} + \underbrace{h_i(n)}_{\text{small perturbation to handle floors and ceilings}}) + g(n)$$

Then $T(n) = \Theta\left(n^p\left(1 + \int_1^n \frac{g(u)}{u^{p+1}}du\right)\right)$ where p satisfies $\sum_{i=1}^{k} a_i b_i^p = 1$.

Ex. $T(n) = 7/4 \; T(\lfloor n/2 \rfloor) + T(\lceil 3/4\, n \rceil) + n^2$.

- $a_1 = 7/4,\; b_1 = 1/2,\; a_2 = 1,\; b_2 = 3/4 \;\Rightarrow\; p = 2$.
- $h_1(n) = \lfloor 1/2\, n \rfloor - 1/2\, n,\; h_2(n) = \lceil 3/4\, n \rceil - 3/4\, n$.
- $g(n) = n^2 \;\Rightarrow\; T(n) = \Theta(n^2 \log n)$.

Outline

- Complete Search
- Divide and Conquer.
  - Introduction.
  - Examples.
- Dynamic Programming.
- Greedy.


time for questions

COMP 251

Algorithms & Data Structures (Winter 2021)

Algorithm Paradigms – Divide and Conquer 2

School of Computer Science
McGill University

Slides of (Comp321 ,2021), Langer (2014), slides by K. Wayne Snoeyink, Kleinberg & Tardos, 2005 & Cormen et al., 2009

Announcements

- Why does my code not work on codepost but works locally?
- https://piazza.com/class/kjtabnv6a021az?cid=236
  - codepost runs in Java 8
  - don't upload a zip. don't declare packages
  - your code has to not crash for any valid input.
    - Some input is not shown to you, so if your public tests fail, it is probably because of an issue with a hidden test.
  - in Q2, we are handling exceptions for you.
    - -2 flag
    - Limit of 30 seconds.

Outline

- Complete Search
- Divide and Conquer.
  - Introduction.
  - Examples.
- Dynamic Programming.
- Greedy.

Divide and Conquer – Arithmetic Operations

- Given 2 (binary) numbers, we want efficient algorithms to:
  - Add 2 numbers
  - **Multiply 2 numbers** (using divide-and-conquer!)

Divide and Conquer – Arithmetic Operations

Integer addition

Addition. Given two n-bit integers a and b, compute $a + b$.
Subtraction. Given two n-bit integers a and b, compute $a - b$.

Grade-school algorithm. $\Theta(n)$ bit operations.

	1	1	1	1	1	1	0	1	
		1	1	0	1	0	1	0	1
+		0	1	1	1	1	1	0	1
	1	0	1	0	1	0	0	1	0

$$\begin{array}{r} 352 \\ +\ 964 \\ \hline 1316 \end{array}$$

$$\begin{array}{c} x[n] \\ y[n] \\ \hline \text{sum}[n+1] \end{array}$$

Remark. Grade-school addition and subtraction algorithms are asymptotically optimal.

Divide and Conquer – Arithmetic Operations

Integer multiplication

Multiplication. Given two n-bit integers a and b, compute $a \times b$.

Grade-school algorithm. $\Theta(n^2)$ bit operations.

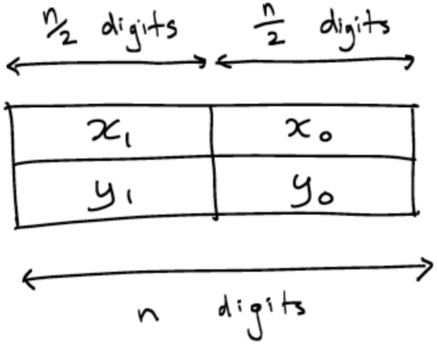
									1	1	0	1	0	1	0	1
								×	0	1	1	1	1	1	0	1
									1	1	0	1	0	1	0	1
								0	0	0	0	0	0	0	0	
							1	1	0	1	0	1	0	1		
						1	1	0	1	0	1	0	1			
					1	1	0	1	0	1	0	1				
				1	1	0	1	0	1	0	1					
			1	1	0	1	0	1	0	1						
		0	0	0	0	0	0	0	0							
	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	1

Conjecture. [Kolmogorov 1952] Grade-school algorithm is optimal.

Theorem. [Karatsuba 1960] Conjecture is wrong.

352 x[n]
× 964 y[n]
1408
2112 tmp[n][2n]
3168
339328 r[2n]

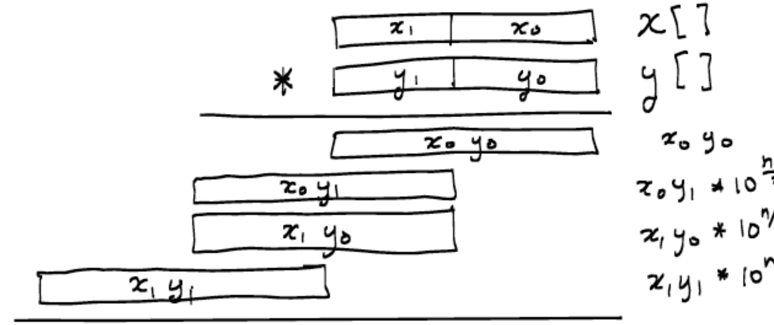
Divide and Conquer – Arithmetic Operations



$$x = x_1 * 10^{n/2} + x_0$$
$$y = y_1 * 10^{n/2} + y_0$$

e.g.

$$3527 = 3500 + 27$$
$$= 35 \times 10^2 + 27$$



$$x * y$$

$$= (x_1 * 10^{\frac{n}{2}} + x_0) * (y_1 * 10^{\frac{n}{2}} + y_0)$$

$$= x_1 y_1 * 10^n + (x_0 y_1 + x_1 y_0) * 10^{n/2} + x_0 y_0$$

$t(\frac{n}{2})$ $t(\frac{n}{2})$ $t(\frac{n}{2})$ $t(\frac{n}{2})$

Note:

$* 10^{n/2}$ shifts left by $\frac{n}{2}$ positions

$* 10^n$ " " " n positions.

Divide and Conquer – Arithmetic Operations

Divide-and-conquer multiplication

To multiply two n-bit integers x and y:

- Divide x and y into low- and high-order bits.
- Multiply four ½n-bit integers, recursively.
- Add and shift to obtain result.

$$m = \lceil n / 2 \rceil$$

$$a = \lfloor x / 2^m \rfloor \quad b = x \bmod 2^m$$

$$c = \lfloor y / 2^m \rfloor \quad d = y \bmod 2^m$$

use bit shifting to compute 4 terms

$$(2^m a + b)(2^m c + d) = \underbrace{2^{2m} ac}_{1} + 2^m (\underbrace{bc}_{2} + \underbrace{ad}_{3}) + \underbrace{bd}_{4}$$

Ex. $x = \underbrace{1\,0\,0\,0}_{a}\,\underbrace{1\,1\,0\,1}_{b} \quad y = \underbrace{1\,1\,1\,0}_{c}\,\underbrace{0\,0\,0\,1}_{d}$

MULTIPLY(x, y, n)

IF ($n = 1$)
  RETURN $x \times y$.
ELSE
  $m \leftarrow \lceil n / 2 \rceil$.
  $a \leftarrow \lfloor x / 2^m \rfloor$; $b \leftarrow x \bmod 2^m$.
  $c \leftarrow \lfloor y / 2^m \rfloor$; $d \leftarrow y \bmod 2^m$.
  $e \leftarrow$ MULTIPLY(a, c, m).
  $f \leftarrow$ MULTIPLY(b, d, m).
  $g \leftarrow$ MULTIPLY(b, c, m).
  $h \leftarrow$ MULTIPLY(a, d, m).
  RETURN $2^{2m} e + 2^m (g + h) + f$.

Divide and Conquer – Arithmetic Operations

Divide-and-conquer multiplication analysis

Proposition. The divide-and-conquer multiplication algorithm requires $\Theta(n^2)$ bit operations to multiply two n-bit integers.

Pf. Apply case 1 of the master theorem to the recurrence:

$$T(n) = \underbrace{4T(n/2)}_{\text{recursive calls}} + \underbrace{\Theta(n)}_{\text{add, shift}} \Rightarrow T(n) = \Theta(n^2)$$

Multiplication. Given two n-bit integers a and b, compute $a \times b$.
Grade-school algorithm. $\Theta(n^2)$ bit operations.

Divide and Conquer – Karatsuba trick

$x =$	x_1	x_0
$y =$	y_1	y_0

$x * y$

$$= x_1 y_1 * 10^n + \underbrace{(x_0 y_1 + x_1 y_0)}_{(x_1 + x_0)(y_1 + y_0) - x_1 y_1 - x_0 y_0} * 10^{n/2} + x_0 y_0$$

Thus, $\boxed{t(n) = 3\, t\left(\frac{n}{2}\right) + c\, n\, .}$

Avengers Assemble In Final Battle Scene - AVENGERS: ENDGAME (2019). Taken from youtube

Divide and Conquer – Karatsuba trick

To compute middle term $bc + ad$, use identity:

$$bc + ad = ac + bd - (a-b)(c-d)$$

$$m = \lceil n/2 \rceil$$

$$a = \lfloor x / 2^m \rfloor \quad b = x \bmod 2^m$$

$$c = \lfloor y / 2^m \rfloor \quad d = y \bmod 2^m$$

$$(2^m a + b)(2^m c + d) = 2^{2m} ac + 2^m \overbrace{(bc + ad)}^{\text{middle term}} + bd$$

$$= \underset{❶}{2^{2m} ac} + 2^m (\underset{❶}{ac} + \underset{❸}{bd} - \underset{❷}{(a-b)(c-d)}) + \underset{❸}{bd}$$

Bottom line. Only three multiplication of $n/2$-bit integers.

Divide and Conquer – Karatsuba trick

KARATSUBA-MULTIPLY(x, y, n)

IF $(n = 1)$
  RETURN $x \times y$.
ELSE
  $m \leftarrow \lceil n / 2 \rceil$.
  $a \leftarrow \lfloor x / 2^m \rfloor$; $b \leftarrow x \bmod 2^m$.
  $c \leftarrow \lfloor y / 2^m \rfloor$; $d \leftarrow y \bmod 2^m$.
  $e \leftarrow$ KARATSUBA-MULTIPLY(a, c, m).
  $f \leftarrow$ KARATSUBA-MULTIPLY(b, d, m).
  $g \leftarrow$ KARATSUBA-MULTIPLY$(a - b, c - d, m)$.
  RETURN $2^{2m}\, e + 2^m\, (e + f - g) + f$.

Proposition. Karatsuba's algorithm requires $O(n^{1.585})$ bit operations to multiply two n-bit integers.

Pf. Apply case 1 of the master theorem to the recurrence:

$$T(n) = 3\,T(n/2) + \Theta(n) \quad \Rightarrow \quad T(n) = \Theta(n^{\lg 3}) = O(n^{1.585}).$$

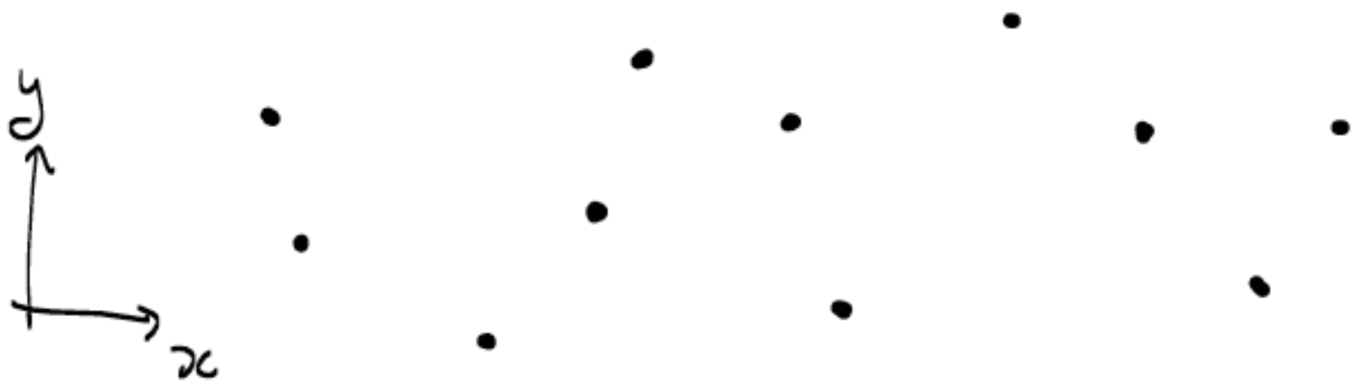
Divide and Conquer – Integer Multiplication

year	algorithm	order of growth
?	brute force	$\Theta(n^2)$
1962	Karatsuba-Ofman	$\Theta(n^{1.585})$
1963	Toom-3, Toom-4	$\Theta(n^{1.465})$, $\Theta(n^{1.404})$
1966	Toom-Cook	$\Theta(n^{1+\varepsilon})$
1971	Schönhage–Strassen	$\Theta(n \log n \log \log n)$
2007	Fürer	$n \log n \, 2^{O(\log^* n)}$
?	?	$\Theta(n)$

number of bit operations to multiply two n-bit integers

Divide and Conquer – Closest points

- Given n points in the plane, find the pair that is closest together.



- Applications in:
  - Computational Geometry.
    - Graphics, computer vision, geographic information systems, molecular modeling.

Divide and Conquer – Closest points

- Given n points in the plane, find the pair that is closest together.

Solution ("brute force"):

```
closest pair = null
δ = ∞
for each i = 1 to n
    for each j = i+1 to n
        if d(i,j) < δ {
            closestpair = (i,j)
            δ = d(i,j)
        }
return closestpair
```

$O(n^2)$ too slow!

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

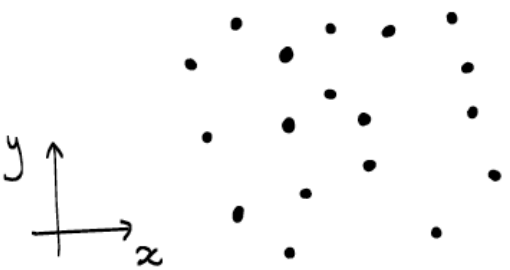
Divide and Conquer – Closest points

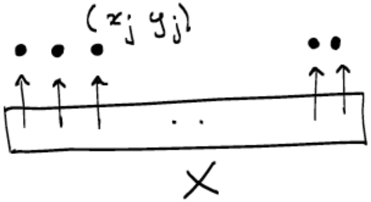
- 1-D Solution.
  - We first sort the points (merge sort) => O(n log n).
  - We'd walk through the sorted list, computing the distance from each point to the one that comes after it => O(n).
    - One of these distances must be the minimum one.
- 2-D Solution.
  - we could try sorting the points by their y-coordinate (or x-coordinate) and hoping that the two closest points were near one another in the order of this sorted list.
    - it is easy to construct examples in which they are very far apart
  - Mimic Merge sort.
    - Find the closest pair among the points in the "left half"
    - Find the closest pair among the points in the "right half"
      - Be careful with the distances that have not been considered.
        - One point is the left and one point in the right half.

Divide and Conquer – Closest points

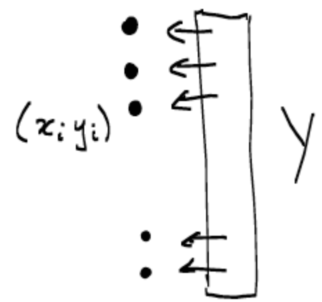
- 2-D Solution.

Solution for 2D (Shamos & Hoey 1970's)



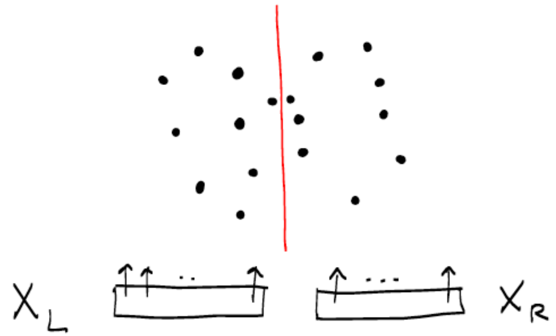


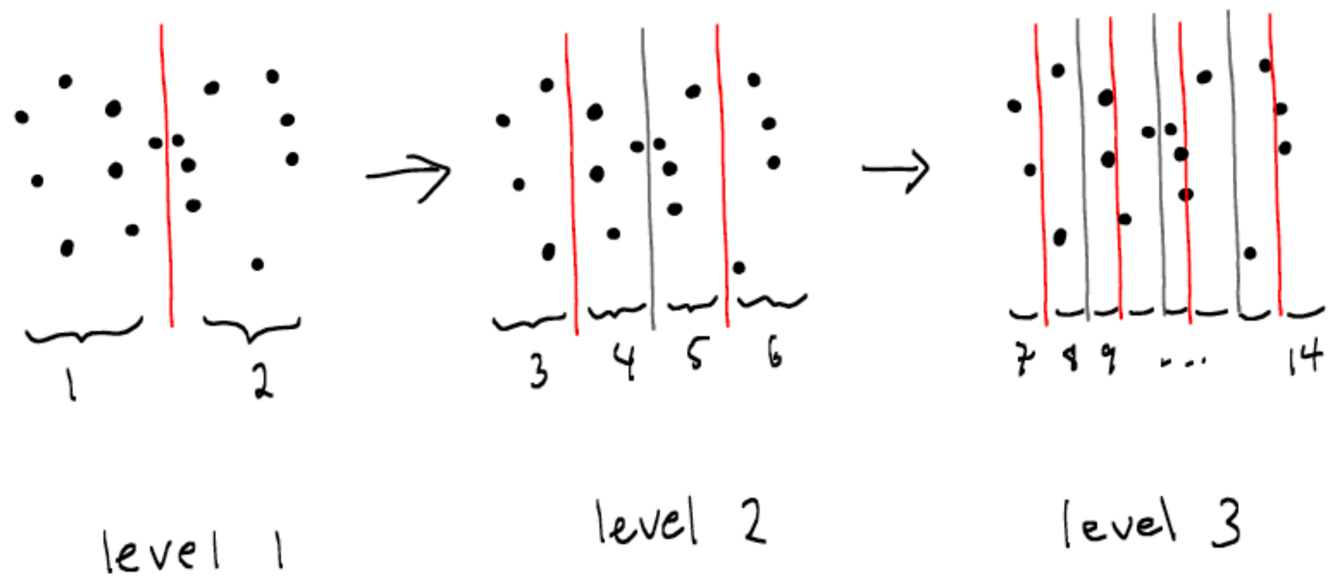
Begin by sorting points by x value, and sorting points by y value, giving two sorted arrays X and Y.



Divide and Conquer – Closest points

- Partition X into two sets:
  - X_L has the n/2 smallest x values ('left')
  - X_R has the n/2 largest x values ('right')





Divide and Conquer – Closest points

Find closest pair (X) { → t(n)

if $|X| \le 3$ then compute closest pair by brute force and return it → c_1

else {

Compute X_L X_R → c_2

Find closest pair (X_L) → t(n/2)

Find closest pair (X_R) → t(n/2)

Find the closest pair such that one point is in X_L and the other point is in X_R. → ?

Return the closest of the three pairs. → c_3

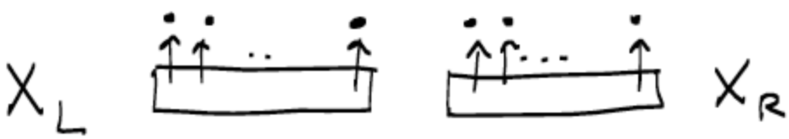
}

}

$$t(n) = 2\,t\left(\frac{n}{2}\right) + ? + c$$

Divide and Conquer – Closest points

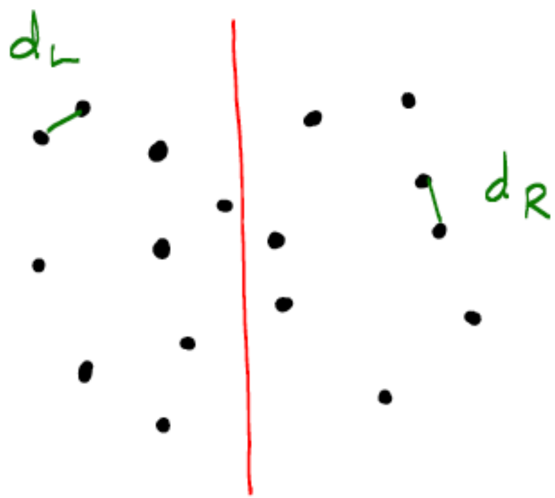
$$t(n) = 2\,t\left(\frac{n}{2}\right) + \,?\, + c$$



- X_L and X_R each have n/2 points. Thus there are n/2 * n/2 pairs of points such that one is in X_L and the other in X_R.
  - Finding the pair with minimum distance using “brute force” would take $O(n^2)$, which is too slow.
  - Can we solve this problem in time $O(n)$, instead on $O(n^2)$?

Divide and Conquer – Closest points

- Let the closest pair in X_L have distance d_L.
- Let the closest pair in X_R have distance d_R.



These are the pairs returned by the two recursive calls

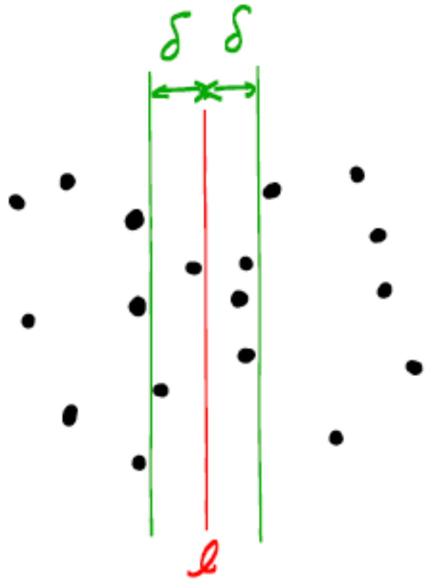
findClosestPair(X_L)

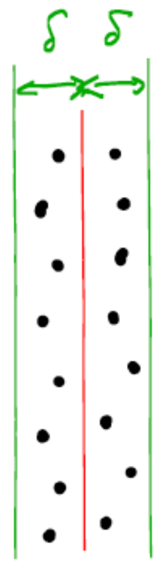
findClosestPair(X_R)

Let $\delta = \min(d_L, d_R)$

Divide and Conquer – Closest points

- Observe that to find the closest pair with one point in X_L and the other point in X_R, we only need to consider points that are a distance δ from the line ℓ that separates L and R.

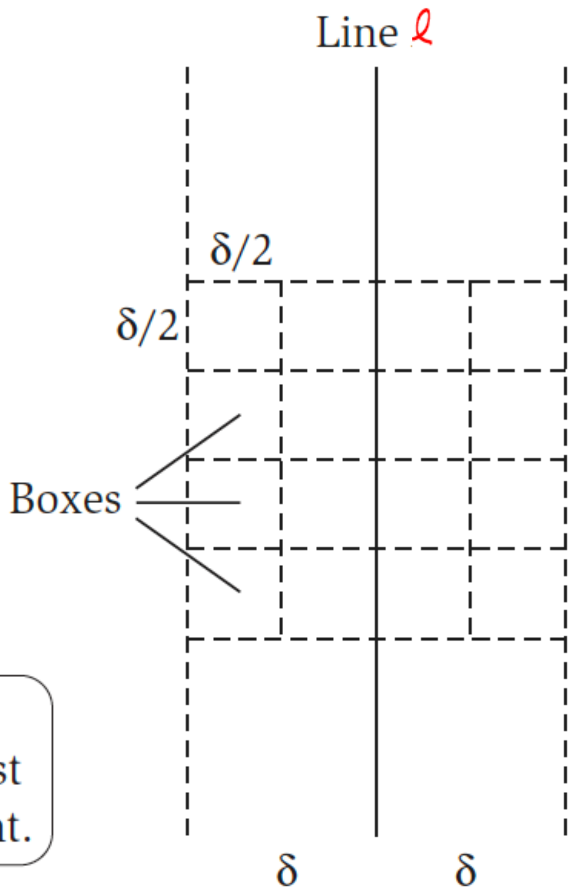




The observation does not necessarily reduce the number of points we Need to consider

Divide and Conquer – Closest points

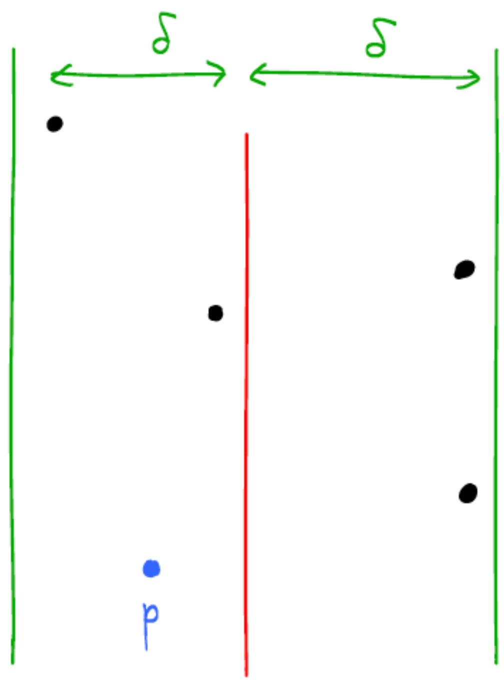
- Consider the subset of the plane consisting of all points within distance δ of ℓ. We can partition this subset into boxes (squares with horizontal and vertical sides of length $\delta/2$). One row of this subset will consist of four boxes whose horizontal sides have the same *y*-coordinates.



Each box can contain at most one input point.

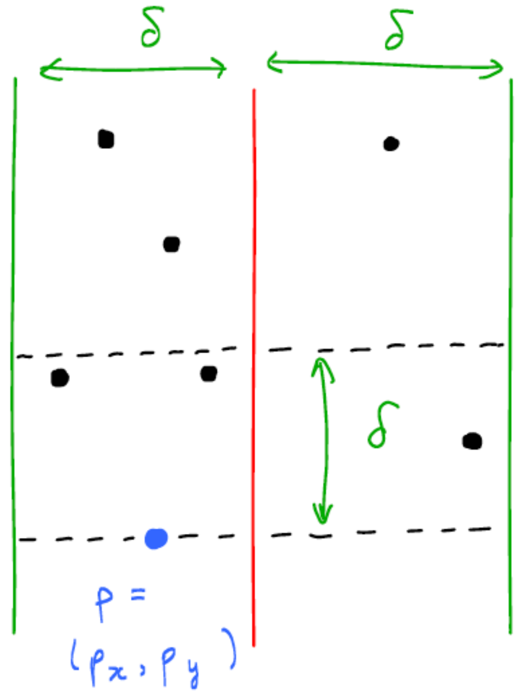
Divide and Conquer – Closest points

- Consider a point p that lies between the two green lines.
  - Is there another point between the green lines that has a *y* value greater than that of p **and** is at a distance less than δ from p?



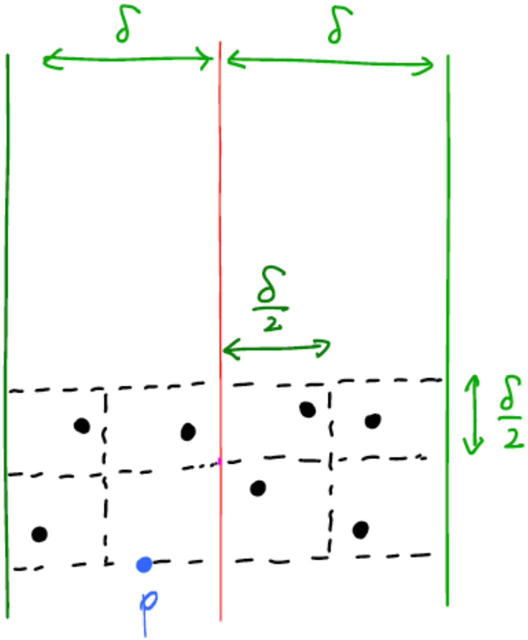
It is sufficient to check those points whose y values are between p_y and $p_y + \delta$

Q: How many points do we need to check in the worst case?



Divide and Conquer – Closest points

- **Q: How many points do we need to check in the worst case?**
- **A: At most 7.**
  - Remember that square cells of width $\frac{\delta}{2}$ can contain at most 1 point.
  - Remember that we also sorted the points by their *y* coordinate



Find the closest pair such that one point is in X_L and the other point is in X_R. {

Find all points that lie between the green lines.

Starting from point with min y value, examine the distance to next 7 points (sorted by y). If we find a pair with distance $< \delta$, make it the new closest pair & update δ.

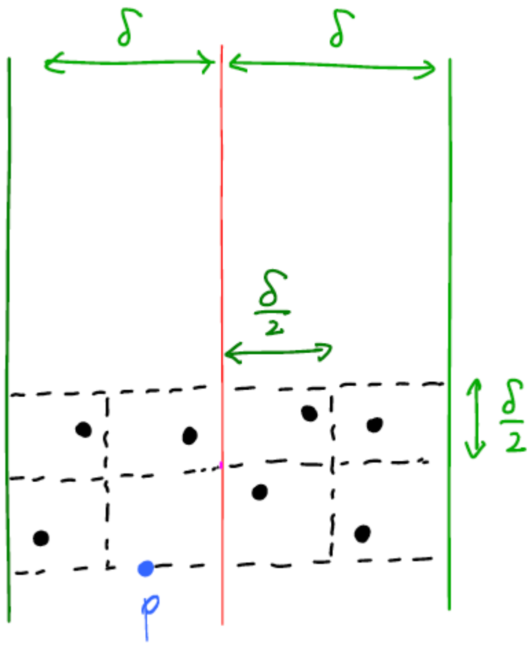
}

Divide and Conquer – Closest points

- **Q: How many points do we need to check in the worst case?**
- **A: At most 7.**

```
middle = empty list
for i = 1 to n                               // find points between
    if Y[i] is between green lines           // green lines: O(n)
        middle.add( Y[i] )

for i = 1 to middle.size {              // O(n)
    for j = 1 to 7 {   // ignore out of bounds error
        tmp = d( middle[i], middle[i+j] )
        if tmp < δ {
            Closest pair = ( middle[i], middle[i+j] )
            δ = tmp
        }
    }
}
```



Divide and Conquer – Closest points

Find closest pair (X) { // → $t(n)$

if $|X| \le 3$ then compute closest pair by brute force and return it → c_1

else {

Compute X_L X_R → c_2

Find closest pair (X_L) → $t(n/2)$

Find closest pair (X_R) → $t(n/2)$

Find the closest pair such that one point is in X_L and the other point is in X_R. → $c_4 n$

Return the closest of the three pairs. → c_3

}

}

$$t(n) = 2\,t\left(\frac{n}{2}\right) + c_4 n + c$$

Divide and Conquer – Closest points

Find closest pair (X) { → $t(n)$

if $|X| \leq 3$ then compute closest pair by brute force and return it → c_1

else {

Compute X_L X_R → c_2

Find closest pair (X_L) → $t(n/2)$

Find closest pair (X_R) → $t(n/2)$

Find the closest pair such that one point is in X_L and the other point is in X_R. → $c_4 n$

Return the closest of the three pairs. → c_3

}

}

$$t(n) = 2t\left(\frac{n}{2}\right) + c_4 n + C$$

→ Same than merge sort O(nlogn)

Matrix multiplication – If time allows

Matrix multiplication. Given two n-by-n matrices A and B, compute $\underbrace{C = AB}_{c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}}$.

Grade-school. $\Theta(n^3)$ arithmetic operations.

$$c_{ij} = \sum_{k=1}^{n} a_{ik}\, b_{kj}$$

$$\begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nn} \end{bmatrix}$$

$$\begin{bmatrix} .59 & .32 & .41 \\ .31 & .36 & .25 \\ .45 & .31 & .42 \end{bmatrix} = \begin{bmatrix} .70 & .20 & .10 \\ .30 & .60 & .10 \\ .50 & .10 & .40 \end{bmatrix} \times \begin{bmatrix} .80 & .30 & .50 \\ .10 & .40 & .10 \\ .10 & .30 & .40 \end{bmatrix}$$

SQUARE-MATRIX-MULTIPLY(A, B)

```
n = A.rows
let C be a new n × n matrix
for i = 1 to n
    for j = 1 to n
        c_ij = 0
        for k = 1 to n
            c_ij = c_ij + a_ik · b_kj
return C
```

Matrix multiplication – divide and conquer

Suppose that we partition each of A, B, and C into four $n/2 \times n/2$ matrices

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \quad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}, \quad C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix},$$

so that we rewrite the equation $C = A \cdot B$ as

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \cdot \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}.$$

Equation (4.10) corresponds to the four equations

$$\begin{aligned} C_{11} &= A_{11} \cdot B_{11} + A_{12} \cdot B_{21}, \\ C_{12} &= A_{11} \cdot B_{12} + A_{12} \cdot B_{22}, \\ C_{21} &= A_{21} \cdot B_{11} + A_{22} \cdot B_{21}, \\ C_{22} &= A_{21} \cdot B_{12} + A_{22} \cdot B_{22}. \end{aligned}$$

C_{11} A_{11} A_{12} B_{11}

$$\begin{bmatrix} 152 & 158 & 164 & 170 \\ 504 & 526 & 548 & 570 \\ 856 & 894 & 932 & 970 \\ 1208 & 1262 & 1316 & 1370 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 2 & 3 \\ 4 & 5 & 6 & 7 \\ 8 & 9 & 10 & 11 \\ 12 & 13 & 14 & 15 \end{bmatrix} \times \begin{bmatrix} 16 & 17 & 18 & 19 \\ 20 & 21 & 22 & 23 \\ 24 & 25 & 26 & 27 \\ 28 & 29 & 30 & 31 \end{bmatrix}$$

B_{11}

$$C_{11} = A_{11} \times B_{11} + A_{12} \times B_{21} = \begin{bmatrix} 0 & 1 \\ 4 & 5 \end{bmatrix} \times \begin{bmatrix} 16 & 17 \\ 20 & 21 \end{bmatrix} + \begin{bmatrix} 2 & 3 \\ 6 & 7 \end{bmatrix} \times \begin{bmatrix} 24 & 25 \\ 28 & 29 \end{bmatrix} = \begin{bmatrix} 152 & 158 \\ 504 & 526 \end{bmatrix}$$

Matrix multiplication – divide and conquer

To multiply two n-by-n matrices A and B:

- Divide: partition A and B into ½n-by-½n blocks.
- Conquer: multiply 8 pairs of ½n-by-½n matrices, recursively.
- Combine: add appropriate products using 4 matrix additions.

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

$$\begin{aligned} C_{11} &= (A_{11} \times B_{11}) + (A_{12} \times B_{21}) \\ C_{12} &= (A_{11} \times B_{12}) + (A_{12} \times B_{22}) \\ C_{21} &= (A_{21} \times B_{11}) + (A_{22} \times B_{21}) \\ C_{22} &= (A_{21} \times B_{12}) + (A_{22} \times B_{22}) \end{aligned}$$

Running time. Apply case 1 of Master Theorem.

$$T(n) = \underbrace{8T(n/2)}_{\text{recursive calls}} + \underbrace{\Theta(n^2)}_{\text{add, form submatrices}} \Rightarrow T(n) = \Theta(n^3)$$

Matrix multiplication – Strassen's trick

Key idea. multiply 2-by-2 blocks with only 7 multiplications.
(plus 11 additions and 7 subtractions)

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

$$\begin{aligned} C_{11} &= P_5 + P_4 - P_2 + P_6 \\ C_{12} &= P_1 + P_2 \\ C_{21} &= P_3 + P_4 \\ C_{22} &= P_1 + P_5 - P_3 - P_7 \end{aligned}$$

$$\begin{aligned} P_1 &\leftarrow A_{11} \times (B_{12} - B_{22}) \\ P_2 &\leftarrow (A_{11} + A_{12}) \times B_{22} \\ P_3 &\leftarrow (A_{21} + A_{22}) \times B_{11} \\ P_4 &\leftarrow A_{22} \times (B_{21} - B_{11}) \\ P_5 &\leftarrow (A_{11} + A_{22}) \times (B_{11} + B_{22}) \\ P_6 &\leftarrow (A_{12} - A_{22}) \times (B_{21} + B_{22}) \\ P_7 &\leftarrow (A_{11} - A_{21}) \times (B_{11} + B_{12}) \end{aligned}$$

Pf. $$\begin{aligned} C_{12} &= P_1 + P_2 \\ &= A_{11} \times (B_{12} - B_{22}) + (A_{11} + A_{12}) \times B_{22} \\ &= A_{11} \times B_{12} + A_{12} \times B_{22}. \ \checkmark \end{aligned}$$

Matrix multiplication – Strassen's trick

STRASSEN(n, A, B)

IF $(n = 1)$ RETURN $A \times B$.

Partition A and B into 2-by-2 block matrices.

$P_1 \leftarrow$ STRASSEN$(n / 2, A_{11}, (B_{12} - B_{22}))$.

$P_2 \leftarrow$ STRASSEN$(n / 2, (A_{11} + A_{12}), B_{22})$.

$P_3 \leftarrow$ STRASSEN$(n / 2, (A_{21} + A_{22}), B_{11})$.

$P_4 \leftarrow$ STRASSEN$(n / 2, A_{22}, (B_{21} - B_{11}))$.

$P_5 \leftarrow$ STRASSEN$(n / 2, (A_{11} + A_{22}) \times (B_{11} + B_{22}))$.

$P_6 \leftarrow$ STRASSEN$(n / 2, (A_{12} - A_{22}) \times (B_{21} + B_{22}))$.

$P_7 \leftarrow$ STRASSEN$(n / 2, (A_{11} - A_{21}) \times (B_{11} + B_{12}))$.

$C_{11} = P_5 + P_4 - P_2 + P_6$.

$C_{12} = P_1 + P_2$.

$C_{21} = P_3 + P_4$.

$C_{22} = P_1 + P_5 - P_3 - P_7$.

RETURN C.

assume n is a power of 2

keep track of indices of submatrices (don't copy matrix entries)

Matrix multiplication – Strassen's trick

Theorem. Strassen's algorithm requires $O(n^{2.81})$ arithmetic operations to multiply two n-by-n matrices.

Pf. Apply case 1 of the master theorem to the recurrence:

$$T(n) = \underbrace{7T\left(n/2\right)}_{\text{recursive calls}} + \underbrace{\Theta(n^2)}_{\text{add, subtract}} \;\Rightarrow\; T(n) = \Theta(n^{\log_2 7}) = O(n^{2.81})$$

Matrix multiplication – Strassen's trick

Implementation issues.

- Sparsity.
- Caching effects.
- Numerical stability.
- Odd matrix dimensions.
- Crossover to classical algorithm when n is "small" .

Common misperception. *"Strassen is only a theoretical curiosity."*

- Apple reports 8x speedup on G4 Velocity Engine when $n \approx 2{,}048$.
- Range of instances where it's useful is a subject of controversy.

Matrix multiplication

year	algorithm	order of growth
?	brute force	$O(n^3)$
1969	Strassen	$O(n^{2.808})$
1978	Pan	$O(n^{2.796})$
1979	Bini	$O(n^{2.780})$
1981	Schönhage	$O(n^{2.522})$
1982	Romani	$O(n^{2.517})$
1982	Coppersmith-Winograd	$O(n^{2.496})$
1986	Strassen	$O(n^{2.479})$
1989	Coppersmith-Winograd	$O(n^{2.376})$
2010	Strother	$O(n^{2.3737})$
2011	Williams	$O(n^{2.3727})$
?	?	$O(n^{2+\varepsilon})$

Outline

- Complete Search
- Divide and Conquer.
  - Introduction.
  - Examples.
- Dynamic Programming.
- Greedy.


time for questions