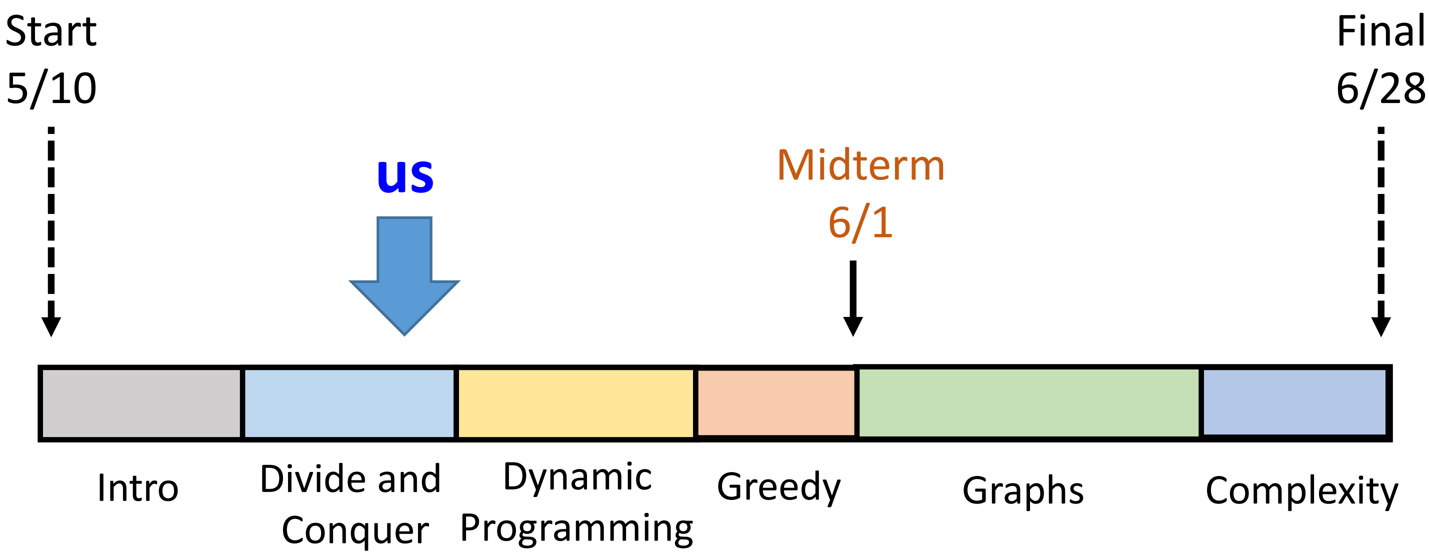
CS3000: Algorithms & Data
Drew van der Poel

Lecture 7
- Binary Search
- Selection

May 19, 2021

Outline



Last class: divide and conquer: Karatsuba's, Master theorem

Next class: dynamic programming: Weighted Interval Scheduling

(Reduce)

Divide-and-Conquer: Binary Search

Binary Search

target

Is t in this list? If so, where?

Sorted:

2	3	8	11	15	17	28	42

A

Q: If t were in the list, which half would it be in?

$t \geq 15$

look in right half

15	17	28	42

$t < 15$

look in left half

2	3	8	11

Binary Search

Is 28 in this list? If so, where?

$n = 8$

Sorted:

2	3	8	$\boxed{11}$	15	17	28	42

A

$l = 1$ $\quad$ m $\quad$ $r = n$

$$m = l + \left\lfloor \frac{r-l}{2} \right\rfloor$$

$$1 + \left\lfloor \frac{8-1}{2} \right\rfloor = 4$$

t vs. $A[m]$

(1) $t > A[m] \rightarrow$ search in $A[m+1, \ldots, r]$

(2) $t < A[m] \rightarrow$ search in $A[1, \ldots, m]$

(3) $t = A[m] \rightarrow$ return m

t vs. $A[m]$

$28 > 11$

look in

15	17	28	42

$l = m+1, \quad r = r$

$28 > 17$

28	42

$l = 7 \qquad r = 8$

$28 = 28$ ✓

return $m = 7$

- Problems: counting students, stable matching, sorting, n-digit mulitiplication, **array searching**

- Alg. techniques: divide & conquer

- Analysis: asymptotic analysis, recursion trees, Master Thm.

- Proof techniques: (strong) induction, contradiction

Binary Search

```
Search(A,t):
  // A[1:n] sorted in ascending order
  Return BS(A,1,n,t)
```

ℓ r

```
BS(A,ℓ,r,t):
  If(ℓ > r): return FALSE
```

$$\mathbf{m} \leftarrow \ell + \left\lfloor \frac{r-\ell}{2} \right\rfloor$$

```
(1) If(A[m] = t): return m
(2) ElseIf(A[m] > t): return BS(A,ℓ,m-1,t)
(3) Else: return BS(A,m+1,r,t)
```

Only make 1 of these recursive calls

w/c runtime of BS on list of size n

T(n): $1\,T(n/2) + O(1)$

T(1): $O(1)$

Running Time Analysis

$$T(n) = T(n/2) + C$$
$$T(1) = C$$

Master Thm!

$a = 1$

$b = 2$

$d = 0$

$$\frac{1}{2^0} = 1$$

Case 2

$T(n) = O(\log n)$

or $\Theta(\log n)$ in w/c

Binary Search Wrapup

- Search a sorted array in time $O(\log n)$
- Divide-and-conquer approach
  - Find the middle of the list, recursively search half the list
  - **Key Fact:** eliminate half the list each time
- Prove correctness via induction
- Analyze running time via Master Thm.
  - $T(n) = T(n/2) + C$

Selection (Median)

Selection

unsorted

- Given an array of numbers $A[1, \dots, n]$, how quickly can I find the:
  - Smallest number? $\rightarrow O(n)$
  - Second smallest? $\rightarrow O(n)$ $\quad \underbrace{n}_{1^{st}\text{ pass}} + \underbrace{n-1}_{2^{nd}\text{ pass}}$
  - SELECTION k-th smallest? $\rightarrow O(kn)$
  - MEDIAN median? $\rightarrow O(n^2)$ let $k = \lceil \frac{n}{2} \rceil$

median: $\lceil \frac{n}{2} \rceil$

11	3	42	28	17	8	2	15	A

Selection

- **Fact:** can select the k-th smallest in $O(n \log n)$ time
  - Sort the list and look up $A[k]$ $O(n \log n)$

11	3	42	28	17	8	2	15	$\boldsymbol{A}$

sort

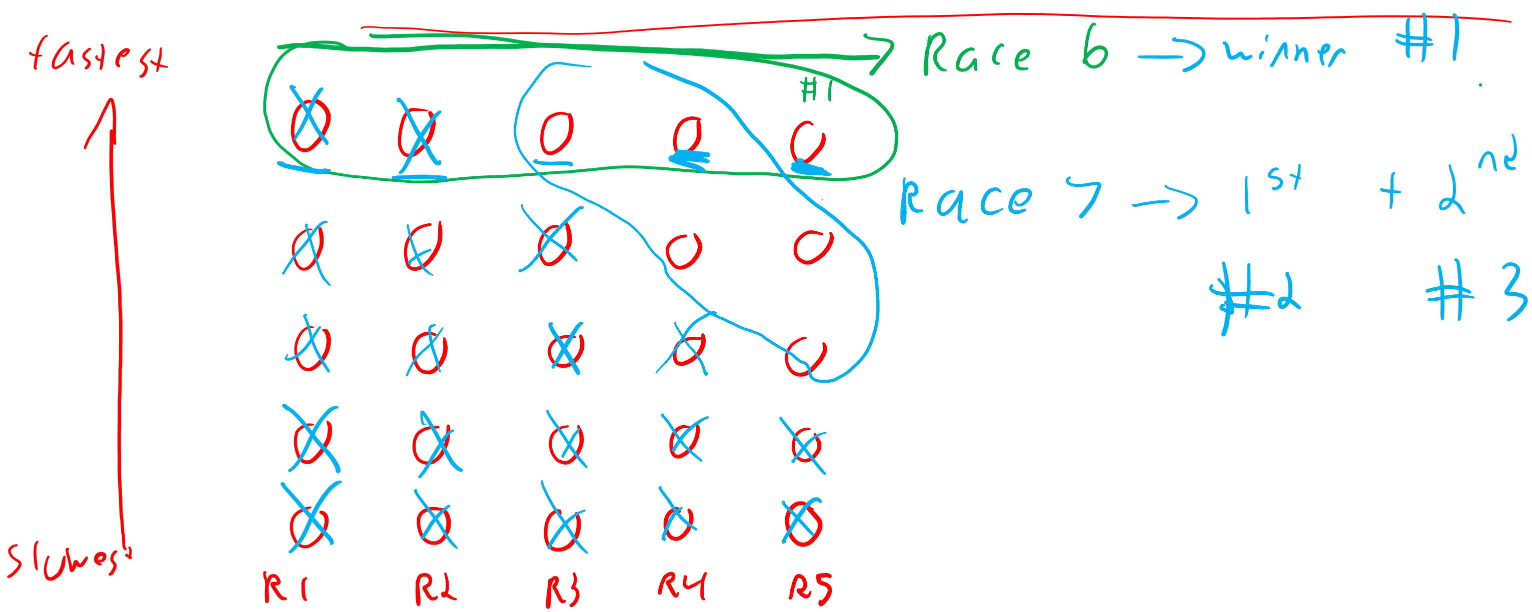
2	3	8	11	15	17	28	42

- **Today:** select the k-th smallest in $O(n)$ time

- Problems: counting students, stable matching, sorting, n-digit mulitiplication, array searching, **selection**
- Alg. techniques: divide & conquer
- Analysis: asymptotic analysis, Master Thm.
- Proof techniques: (strong) induction, contradiction

Warmup

- You have 25 horses and want to find the 3 fastest
- You have a racetrack where you can race 5 at a time
  - In: $\{1, 5, 6, 18, 22\}$ Out: $(6 \succ 5 \succ 18 \succ 22 \succ 1)$
  - You don't have a stopwatch
  - Each horse always has the same finish time
- **Problem:** find the 3 fastest with only seven races



Median Algorithm: Take I

p=17

17	3	42	11	28	8	2	15	13	***A***

r=7

11	3	15	13	2	8	17	28	42

```
Select(A[1:n],k):
  If(n = 1): return A[1]

  Choose a pivot p = A[1]
  Partition around the pivot, let r = indexOf(A,p)

  If (k = r): return A[r]
  ElseIf(k < r): return Select(A[1:r-1],k)
  ElseIf(k > r): return Select(A[r+1:n],k-r)
```

Median Algorithm: Take I

p = 1

1	2	3	4	5	6	7	8	9	A

p = 2

2	3	4	5	6	7	8	9

3	4	5	6	7	8	9

$\vdots$

Partitioning around pivot: $O(n)$

require $n/2$ iterations $\rightarrow O(n^2)$

Median Algorithm: Take II

- **Problem:** we need to find a good pivot element

Best pivot element: Median!

* "close" to the median

is good enough

Median of Medians

```
MOM(A[1:n]):
  Let m ← ⌈n/5⌉
  For i = 1,…,m:
    M[i] ← median{A[5i-4],…,A[5i]}
  p ← Select(M[1:m],⌈m/2⌉)
```

splitting into groups of size 5 + finding median

MoM

list of medians

median

Finding median in list of length 5: $O(1)$

Number of times we find median: $\frac{n}{5} = O(n)$

# of ops. excluding recursive call: $\frac{n}{5} \cdot c = O(n)$

MoM running time: cn + time to run Select (find median) on list of length $m = n/5$

Median of Medians

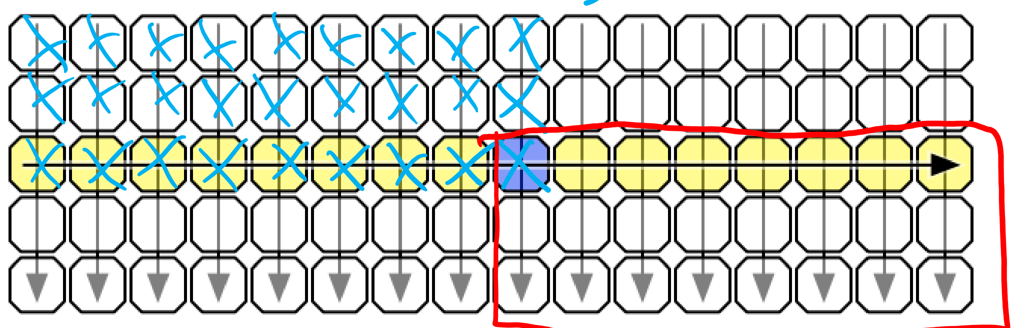
- **Claim:** For every $\boldsymbol{A}$ there are at least $\frac{3n}{10}$ items that are ~~smaller~~ larger than $\mathbf{MOM}(\boldsymbol{A})$

at least

$n/5$ groups, in $1/2$ groups MoM is ≥ 3 elements

$\therefore$ in $n/10$ groups MoM is ≥ 3 elements

$\longrightarrow$ MoM is $\geq \frac{3n}{10}$ elems.



Visualizing the median of medians

Also MoM is $\leq \frac{3n}{10}$ elems.

Selection Algorithm: Take II

T(n): $T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10}\right) + cn$

T(1): $O(1)$

```
MOMSelect(A[1:n],k):
  If(n ≤ 25): sort A and return A[k]

  Let p = MOM(A)
  Partition around the pivot, let r = IndexOf(A, p)

  If(k = r): return A[r]
  ElseIf(k < r): return MOMSelect(A[1:r-1],k)
  ElseIf(k > r): return MOMSelect(A[r+1:n],k-r)
```

$O(1)$

$T\left(\frac{7n}{10}\right)$

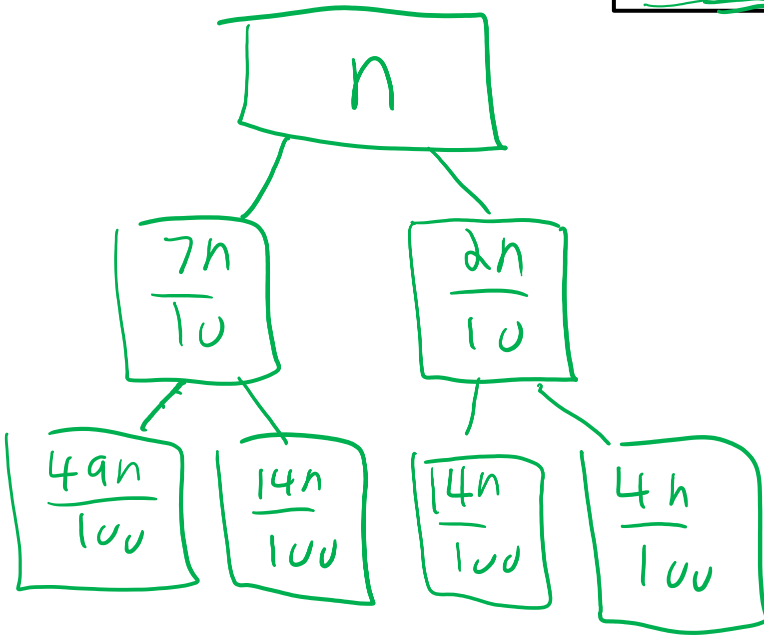
Time of MOM: $\Theta(n) + T(n/5)$

Recursion Tree

$$T(n) = T\left(\frac{7n}{10}\right) + T\left(\frac{2n}{10}\right) + Cn$$

$$T(1) = C$$

level	largest piece	work @ level
0	n	cn
1	$\frac{7n}{10}$	$c\frac{7n}{10} + c\frac{2n}{10} = c\frac{9n}{10}$
2	$\frac{49n}{100}$	$c\frac{81n}{100}$
i	$\left(\frac{7}{10}\right)^i n$	$c\left(\frac{9}{10}\right)^i n$



$$\left(\frac{7}{10}\right)^i n = 1$$

$\log_{\frac{10}{7}}(n)$

$$i = \log_{\frac{10}{7}}(n)$$

$$cn\sum_{i=0}^{\log_{\frac{10}{7}} n}\left(\frac{9}{10}\right)^i = c\,n\,\underline{\underline{c'}} = O(n)$$

Ask the Audience

- If we change MOM so that it uses $\frac{n}{3}$ blocks of size 3, how many items can we eliminate?

- What is the new running time of the algorithm?

Selection Wrapup

- Find the k-th largest element in $O(n)$ time
  - Selection is strictly easier than sorting!
- Divide-and-conquer approach
  - Find a pivot element that splits the list roughly in half
  - **Key Fact:** median-of-medians-of-five is a good pivot
- Can sort in $O(n \log n)$ time using same technique
  - Algorithm is called `Quicksort`
- Analyze running time via recurrence
  - Master Theorem does not apply
- **Fun Fact:** a random pivot is also a good pivot in expectation!

Dynamic Programming

Dynamic Programming

- Don't think too hard about the name
  - *I thought dynamic programming was a good name. It was something not even a congressman could object to. So I used it as an umbrella for my activities.* –Richard Bellman
- Dynamic programming is careful & smarter recursion
  - Break the problem up into small pieces & recursively solve (like Divide & Conquer)
  - Reuse solutions as necessary when subproblems repeat
  - Often the only poly. time algorithm (D&C doesn't work)

- Problems: counting students, stable matching, sorting, n-digit mulitiplication, array searching, selection

- Alg. techniques: divide & conquer, **dynamic programming**

- Analysis: asymptotic analysis, recursion trees, Master Thm.

- Proof techniques: (strong) induction, contradiction

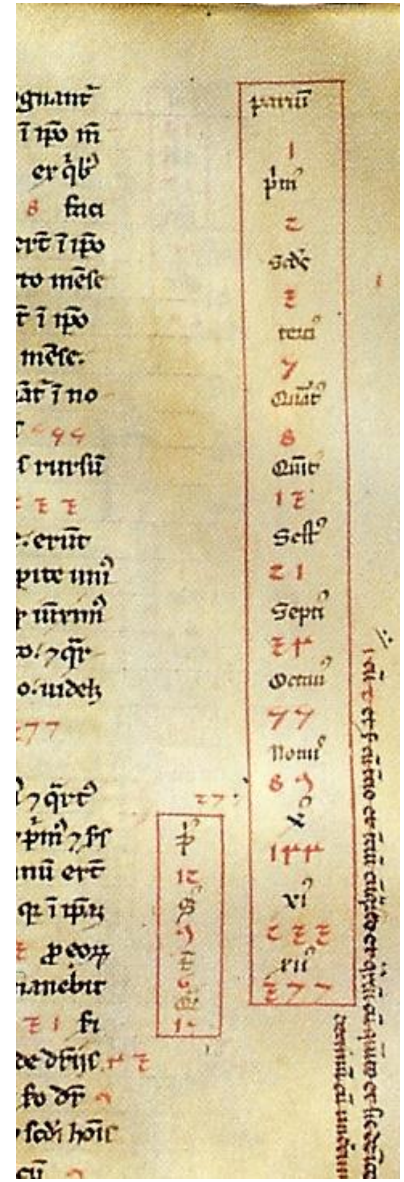
Intro: Fibonacci Numbers

Fibonacci Numbers

- 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...

- $F(1) = 0, F(2) = 1,$
  $F(n) = F(n-1) + F(n-2)$

- $F(n) \to \phi^n \approx 1.62^n$
- $\phi = \left(\frac{1+\sqrt{5}}{2}\right)$ is the golden ratio



Fibonacci's *Liber Abaci* (1202)

Fibonacci Numbers: Take I

```
FibI(n):
  If (n = 1): return 0
  ElseIf (n = 2): return 1
  Else: return FibI(n-1) + FibI(n-2)
```

- How many calls does **`FibI(n)`** make?
  - $T(n) = \#\ of\ calls\ by\ FibI(n)$

Fibonacci Numbers: Take II ("Top down")

```
M ← empty array, M[0]←0, M[1]←1
FibII(n):
  If (M[n] is not empty): return M[n]
  ElseIf (M[n] is empty):
    M[n] ← FibII(n-1) + FibII(n-2)
    return M[n]
```

- How many recursive calls does **`FibII(n)`** make?

Fibonacci Numbers: Take III ("Bottom up")

```
FibIII(n):
  M[1] ← 0, M[2] ← 1
  For i = 3,…,n:
    M[i]← M[i-1] + M[i-2]
  return M[n]
```

- What is the # of loops of **`FibIII(n)`**?

Fibonacci Numbers

- $0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, \ldots$
- $F(n) = F(n-1) + F(n-2)$

- Solving the recurrence recursively takes $\Omega(1.62^n)$ time
  - Problem: Recompute the same values $F(i)$ many times
- Two ways to improve the running time
  - Remember values you've already computed ("top down")
  - Iterate over all values $F(i)$ ("bottom up")

- **Fact:** Fastest algorithms solve in logarithmic time