More Divide & Conquer

CSE 421 Fall 2022
Lecture 10

Today

Another classic Divide & Conquer Algorithm

Linear Time Median

Today's algorithm is ***not*** practical. You will never use it in practice.

But it is beautiful.

It shows the extent of what one can do with D&C

And it is interesting for theoretical reasons.

Caveats for pseudocode/proofs today

We're assuming all elements are distinct (makes code a bit cleaner)

There are about a million spots where you need to worry about ceilings/floors, off-by-ones, what to do if the number of elements isn't exactly a multiple of something, etc.

You can work them all out on your own. I'm skipping them for this lecture…you'll never actually implement this code anyway…

Goal: Median Finding

Input: An unsorted array

Output: the median element of the array.

Baseline:

What's the first algorithm you think of? What's the running time?

Goal: Median Finding

Input: An unsorted array

Output: the median element of the array.

Baseline:

What's the first algorithm you think of? What's the running time?

Sort the array, return element $n/2$; $O(n \log n)$ running time.

Find the Median

Remember the idea behind quicksort.

Pick an element that you ***hope*** is near the median ("the pivot").

Put everything smaller in one array, everything bigger in the other.

In the sorted array, the pivot goes between the "smaller" array and the "bigger" array.

Make recursive calls on each array and stick the arrays together.

We can adapt the idea to ***just*** find the median:

Find The Median

Where is the median?

```
MedianFind(A[0..n-1])
    Let A[p] be the pivot //TODO need to select p.
    Let S and B be two arrays //"small" and "big" elements
    for(i from 0 to n-1 except p)
        if(A[i] <= A[p])
            Copy A[i] into S
        else
            Copy A[i] into B
    if(S.length == n/2) return A[p] //A[p] is median
    else if (S.length > n/2)
        …//TODO what goes here?
    else
        …//TODO what goes here?
```

Examples

6	3	1	8	10	2	5	11	9	7	4

$n = 11$, median is 6

Pivot $A[0] = 6$

3	1	2	5	4	6	8	10	11	9	7

Pivot $A[10] = 4$

3	1	2	4	6	8	10	5	11	9	7

Pivot $A[3] = 8$

6	3	1	2	4	7	4	8	10	11	9

Find The Median

```
MedianFind(A[0..n-1])
    Let A[p] be the pivot //TODO need to select p.
    Let S and B be two arrays //"small" and "big" elements
    for(i from 0 to n-1 except p)
        if(A[i] <= A[p])
            Copy A[i] into S
        else
            Copy A[i] into B
    if(S.length == n/2) return A[p] //A[p] is median
    else if (S.length > n/2)
        //return element n/2 of SORTED version of S
    else
        //return element n/2-S.length of SORTED version of B
```

//Filling In The Comments

We don't want to sort! We could have done that right from the start.

But, wait, does "find the element that would be at index k in sorted order without doing the sorting" sound familiar?

If we set $\mathrm{k} = n/2$ that's another way of saying find the median!

This ***is*** a recursive call! Or at least it could be, if we just rephrase our problem a bit, and make the index a parameter…

Selection Problem

Input: An unsorted array and an index k

Output: The element that would be at index k in the sorted version of A.

Set $\mathrm{k} = n/2$ to get the median.

Make sure you're able to say in English exactly what you're relying on a recursive call to give you:

QuickSelect(A, k) returns the k^{th} smallest element of A. (i.e. the one at index $k - 1$)

Selection

```
QuickSelect(A[0..n-1], k)
     Let A[p] be the pivot //TODO need to select p.
     Let S and B be two arrays //"small" and "big" elements
     for(i from 0 to n-1 except p)
          if(A[i] <= A[p])
               Copy A[i] into S
          else
               Copy A[i] into B
     if(S.length == k - 1) return A[p] //A[p] is index k
     else if (S.length > k -1)
          //return element k of SORTED version of S
     else
          //return element k-S.length of SORTED version of B
```

Selection

```
QuickSelect(A[0..n-1], k)
    Let A[p] be the pivot //TODO need to select p.
    Let S and B be two arrays //"small" and "big" elements
    for(i from 0 to n-1 except p)
        if(A[i] <= A[p])
            Copy A[i] into S
        else
            Copy A[i] into B
    if(S.length == k - 1) return A[p] //A[p] is index k
    else if (S.length > k -1)
        QuickSelect(S, k)
    else
        QuickSelect(B, k-S.length)
```

Pivot Finding

The key to a good running time is now the same as quicksort – find a good pivot quickly!

What's good? Near the middle.

- Even if our desired index is not near the middle. Goal is to guarantee a decrease in problem size.

Pivot Finding

Remember median-of-three?

It's a common heuristic for pivot finding for quicksort.

Take the median of three arbitrary elements (usually first, last, midpoint).

Guaranteed not to be the absolute worst pivot.

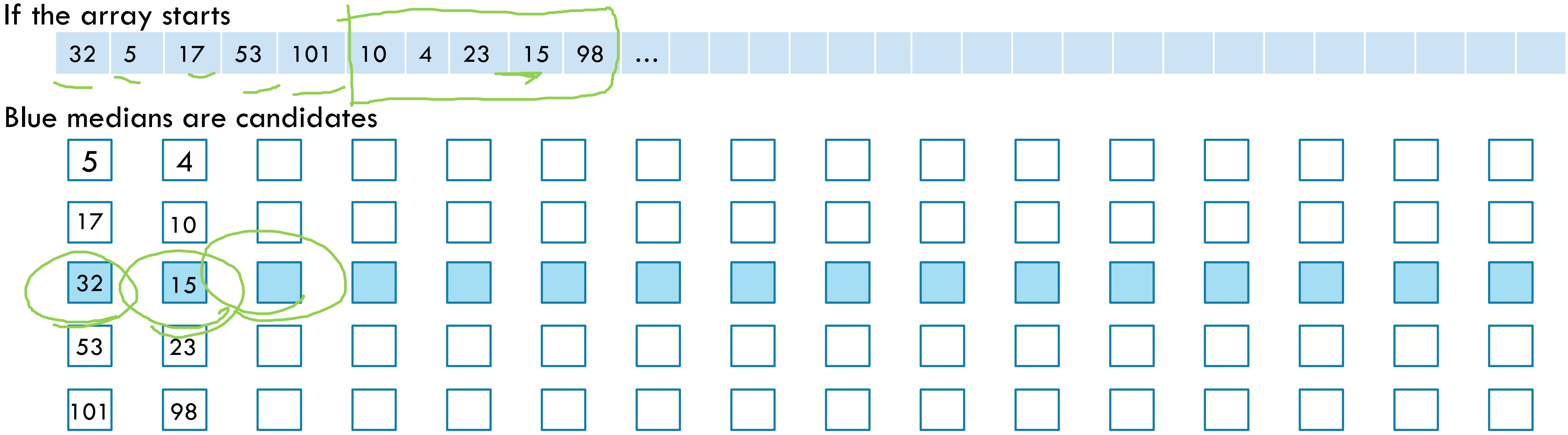
Let's take that idea a lot further…

Pivot Finding

You can still find the median of 5 elements in constant time. (5 is a constant).

Don't just find the median of 5 elements and make that a pivot

…split the array into groups of 5 and get $n/5$ candidate pivots



Pivot Finding

Which of the $n/5$ candidates do you want?

How do we find the median of an array?

QuickSelect! Another recursive call!

Is it the true median of the whole array?

Not necessarily. But it's a good pivot, we'll see how good in a second.

Let's see the pseudocode again…

Pivot Finding

```
int PivotFinder(A[])
        Divide A into n/5 groups of 5.
        Find the median of each group
        Let M contain each of the n/5 medians
        return QuickSelect(M, n/10) //median of M
```

Selection

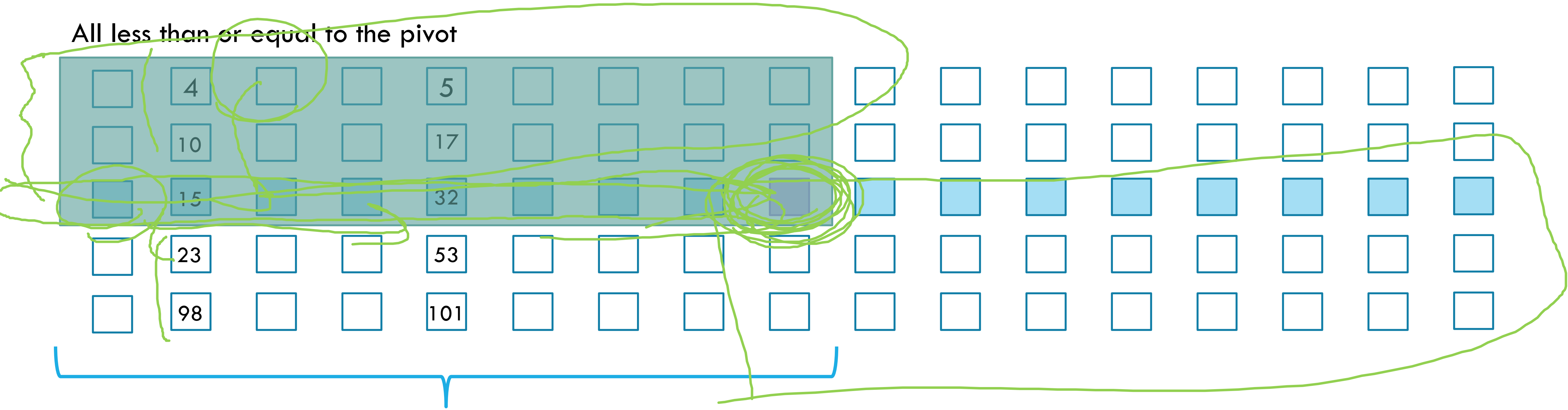
```
QuickSelect(A[0..n-1], k)
    p = PivotFinder(A)
    Let S and B be two arrays //"small" and "big" elements
    for(i from 0 to n-1 except A[p])
        if(A[i] <= A[p])
            Copy A[i] into S
        else
            Copy A[i] into B
    if(S.length == k - 1) return A[p] //A[p] is index k
    else if (S.length > k -1)
        QuickSelect(S, k)
    else
        QuickSelect(B, k-S.length)
```

$O(n)$ and a recursive call

How Good Is Our Pivot?

How many elements smaller than the pivot can you find?

Imagine we lay out the grid with medians increasing left to right



$\frac{n}{10}$ columns (half of the $n/5$)

How Good Is Our Pivot?

How many elements smaller than the pivot can you find?

$\frac{n}{10}$ groups, each with at least 3 elements less than the pivot. $3n/10$

How many elements bigger?

Same analysis: $\frac{3n}{10}$

So how big is that last recursive call?

At most $7n/10$.

And what's the size of the recursive call inside pivot selection? $n/5$.

Running Time Analysis

Let $T(n)$ be the running time of `QuickSelect` on an array of size n.

Non-recursive work?

Selection

```
PivotFinder(A[0..n-1]) //assume n a multiple of 5
    for(i from 0 to n/5-1)
        Find median of A[5i], A[5i+1],…,A[5i+4]
        Add median to C[]
    return QuickSelect(C,n/10)
```

$O(n)$ (for loop)

Selection

```
QuickSelect(A[0..n-1], k)
    p = PivotFinder(A)      } O(n) and a recursive call
    Let S and B be two arrays //"small" and "big" elements
    for(i from 0 to n-1 except A[p])
        if(A[i] <= A[p])
            Copy A[i] into S
        else
            Copy A[i] into B
    if(S.length == k - 1) return A[p] //A[p] is index k
    else if (S.length > k -1)
        QuickSelect(S, k)
    else
        QuickSelect(B, k-S.length)
```

$O(n)$ — Let S and B … Copy A[i] into B

$O(n)$ and a recursive call — p = PivotFinder(A)

Running Time Analysis

Let $T(n)$ be the running time of `QuickSelect` on an array of size n.

Non-recursive work? $O(n)$

$$T(n) = \begin{cases} O(1) & \text{if } n < 100 \\ ???+O(n) & \text{otherwise} \end{cases}$$

Running Time Analysis

Let $T(n)$ be the running time of `QuickSelect` on an array of size n.

Non-recursive work? $O(n)$

$$T(n) = \begin{cases} O(1) & \text{if } n < 100 \\ T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10}\right) + O(n) & \text{otherwise} \end{cases}$$

Running Time Analysis

$$T(n) = \begin{cases} O(1) & \text{if } n < 100 \\ T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10}\right) + O(n) & \text{otherwise} \end{cases}$$

So…what's the closed form? Remember we need better than $O(n \log n)$.

It's $O(n)$. The section solutions have a proof, for today, some intuition…

1. The total combined instance sizes add up to $9n/10$. A constant factor less than n.

2. Two recursive calls of combined size $9n/10$ is no worse than one call of size $9n/10$, as long as the function is concave or linear.

Takeaways

Wow! That was an unexpected algorithm.

You can implement quicksort with guaranteed $O(n \log n)$ running time!

- Use QuickSelect to find the pivot.
- Don't actually do this though. Median-of-3 or a uniformly random pivot are better in practice.

Generalizing a problem can make it easier to solve

- Instead of just the median, finding a general index is recursive.

More Detailed Analysis

You’re not responsible for this.

A simpler analysis

Let's solve this other recurrence for intuition:

$$R(n) = \begin{cases} O(1) & \text{if } n \le 100 \\ R\left(\frac{9n}{10}\right) + c \cdot n & \text{otherwise} \end{cases}$$

$$R(n) = R\left(\frac{9n}{10}\right) + cn$$

$$= R\left(\frac{9^2 n}{10^2}\right) + \frac{9}{10}cn + cn$$

$$= R\left(\frac{9^3}{10^3}n\right) + \frac{9^2}{10^2}cn + \frac{9}{10}cn + cn$$

…

$$= R\left(\frac{9^i}{10^i}n\right) + \sum_{j=0}^{i-1} \frac{9^j}{10^j}cn \quad \text{Set } i = \log_{10/9} n$$

$$= O(1) + \sum_{j=0}^{\log_{10/9} n - 1} \frac{9^j}{10^j}cn \le O(1) + \sum_{j=0}^{\infty} \frac{9^j}{10^j}cn = O(1) + cn \cdot \frac{1}{1-\frac{9}{10}} = O(n)$$

Why groups of 5?

$$\frac{n}{5} + \frac{7n}{10} < n$$

$$\frac{n}{3} + \frac{2n}{3} = n$$

We want an odd number, so there's a "real" median.

$n = 3$ is too small.

The pivot-selection recursive call becomes size $n/3$

The main recursive call becomes size $\frac{2n}{3}$

So the "combined recursion size" is still n. That's too big! We've "rearranged" work, not shrunk it.

Bigger than 5 is worse than 5:

Intuitively, the median-finding is a "quadratic" brute force, while the recursive part is linear. Want recursion to do as much work as possible.

Why is the pivot aimed at the median?

Why not "aim for" the spot you're really interested in?

So if you're looking for spot k in an array of size n, have the pivot finder be searching for k/n instead of $n/2$?

Spot k/n of a group of ***medians*** is not necessarily extremely close to spot k/n. Would have to change the brute force calculation as well.

Our pivot will always be an approximation. Need to make sure if we miss to the "small side" we're still removing a substantial portion of the array.

That might be possible, but harder to write and analyze. And it can only help in constant factors, for an algorithm you aren't going to implement!

Can’t do better than $O(n)$.

The section handout has a proof that you can’t find the median faster than $O(n)$.

Intuition: In less than $O(n)$ time, you can’t even look at every element. And if you don’t look at all the elements, you might not have seen the median itself!