CSCI 3104: Algorithms

Lecture 8: Median of Medians, Quicksort Continued

Rachel Cox

Department of Computer Science

What will we learn today?

- ❑ Median of Medians
- ❑ Wrap-up of Quicksort

Intro to Algorithms, CLRS:
Sections 7.1, 7.2, 7.3, 7.4

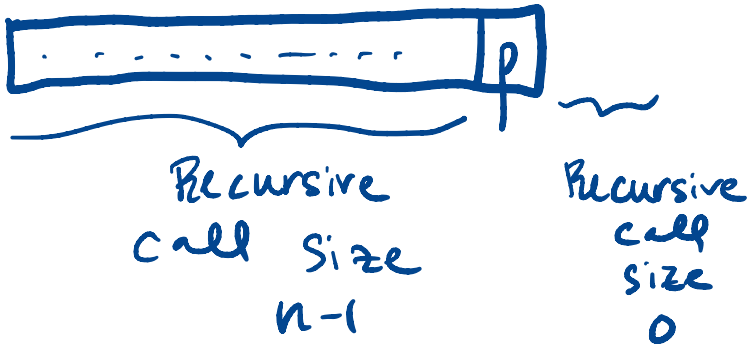
Quicksort

Worst-Case Partitioning

- Occurs when the partitioning routine produces one subproblem with $n - 1$ elements and one with 0 elements.

The recursion tree for this scenario looks like:

worst case - pivot is chosen to be the largest element in the array

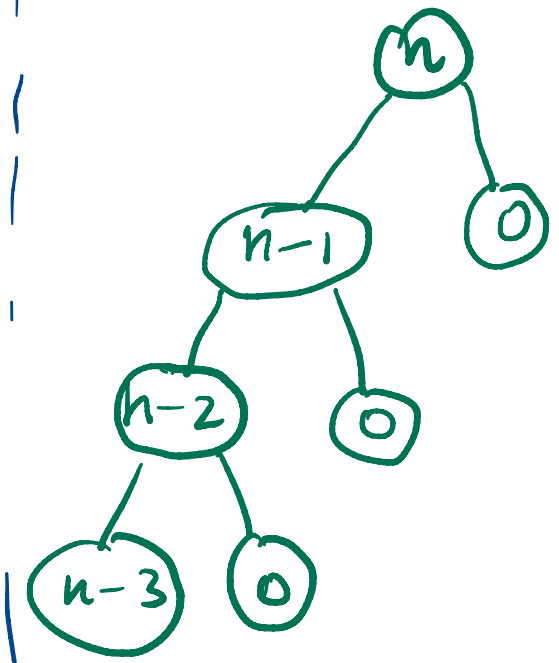


eg. A = [1, 3, 5, 7, 9]

1 3 5 7 (9)

[1 3 5 (7)]

Recursion tree



Depth of that tree?

$\Theta(n)$

Recursion

$$T(n) = T(n-1) + \Theta(n)$$

(1)

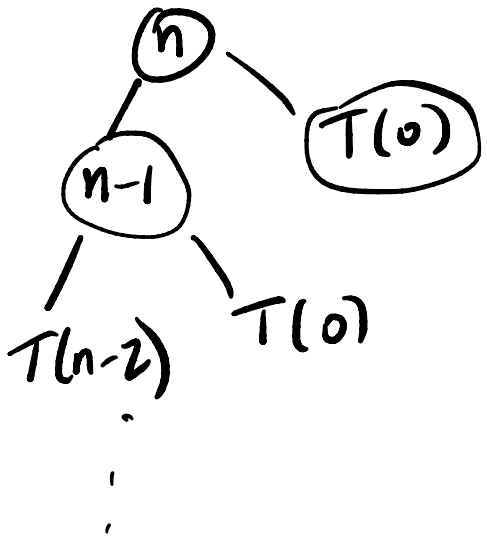
Quicksort

Worst-Case Partitioning

Example: Find the solution to the recurrence $T(n) = T(n-1) + \Theta(n)$

$T(0) = \Theta(1)$ work done on an empty list

Recursion Tree



} work done at top level cn

} work done here c(n-1) + k

} c(n-2) + k

n levels

hand-wavey answer

$\Theta(n^2)$

Quicksort

Worst-Case Partitioning

"unrolling

Example: Find the solution to the recurrence $T(n) = T(n-1) + \Theta(n)$

$T(n) = T(n-1) + \Theta(n)$ Base Case: $T(0) = K$

$= T(n-1) + cn$

$= T(n-2) + c(n-1) + cn$

$= T(n-3) + c(n-2) + c(n-1) + cn$

$= T(n-4) + c(n-3) + c(n-2) + c(n-1) + cn$

$\vdots$

$= T(n-n) + c(n-(n-1)) + c(n-(n-2)) + \cdots + c(n-3) + c(n-2) + c(n-1) + cn$

$= T(0) + c \cdot 1 + c \cdot 2 + c \cdot 3 + \cdots + c \cdot (n-3) + c(n-2) + c \cdot (n-1) + cn$

$= T(0) + c \sum_{i=1}^{n} i$

$= K + c \sum_{i=1}^{n} i$

Quicksort

Worst-Case Partitioning

Example: Find the solution to the recurrence $T(n) = T(n-1) + \Theta(n)$

$$T(n) = k + c\sum_{i=1}^{n} i$$

$$= k + c\left(\frac{n(n+1)}{2}\right)$$

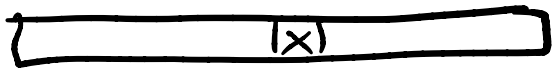
$$= k + \frac{cn^2}{2} + \frac{cn}{2}$$

$\Theta(n^2)$

Quicksort

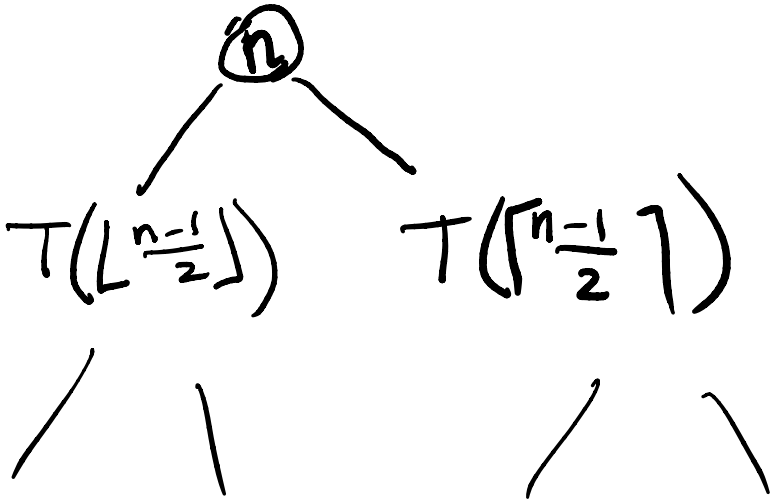
Best-Case Partitioning

- Most even possible split; two subproblems, each of size no more than $n/2$



perfectly balanced partitioning

The recursion tree for this scenario looks like:



recurrence relation:

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

Quicksort

Best-Case Partitioning

Example: Find the solution to the recurrence $T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n)$

$f(n)$, a, b

Master Method.

$f(n) = \Theta(n)$

$a = 2$

$b = 2$

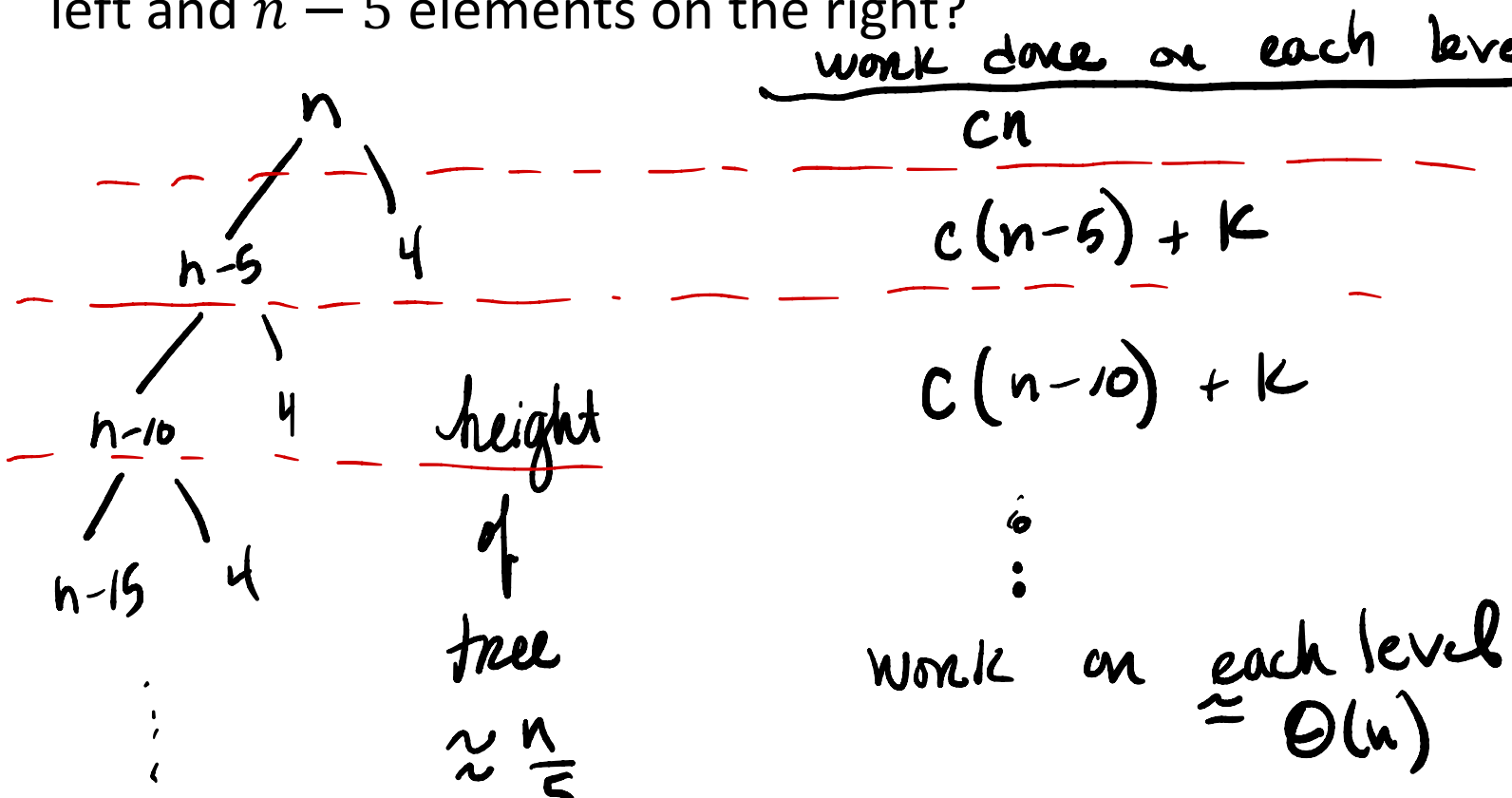
$n^{\log_b a} = n^{\log_2 2} = n^1$

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

CASE 2 of the Master Theorem $\Rightarrow$ $T(n) = \Theta(n \lg(n))$

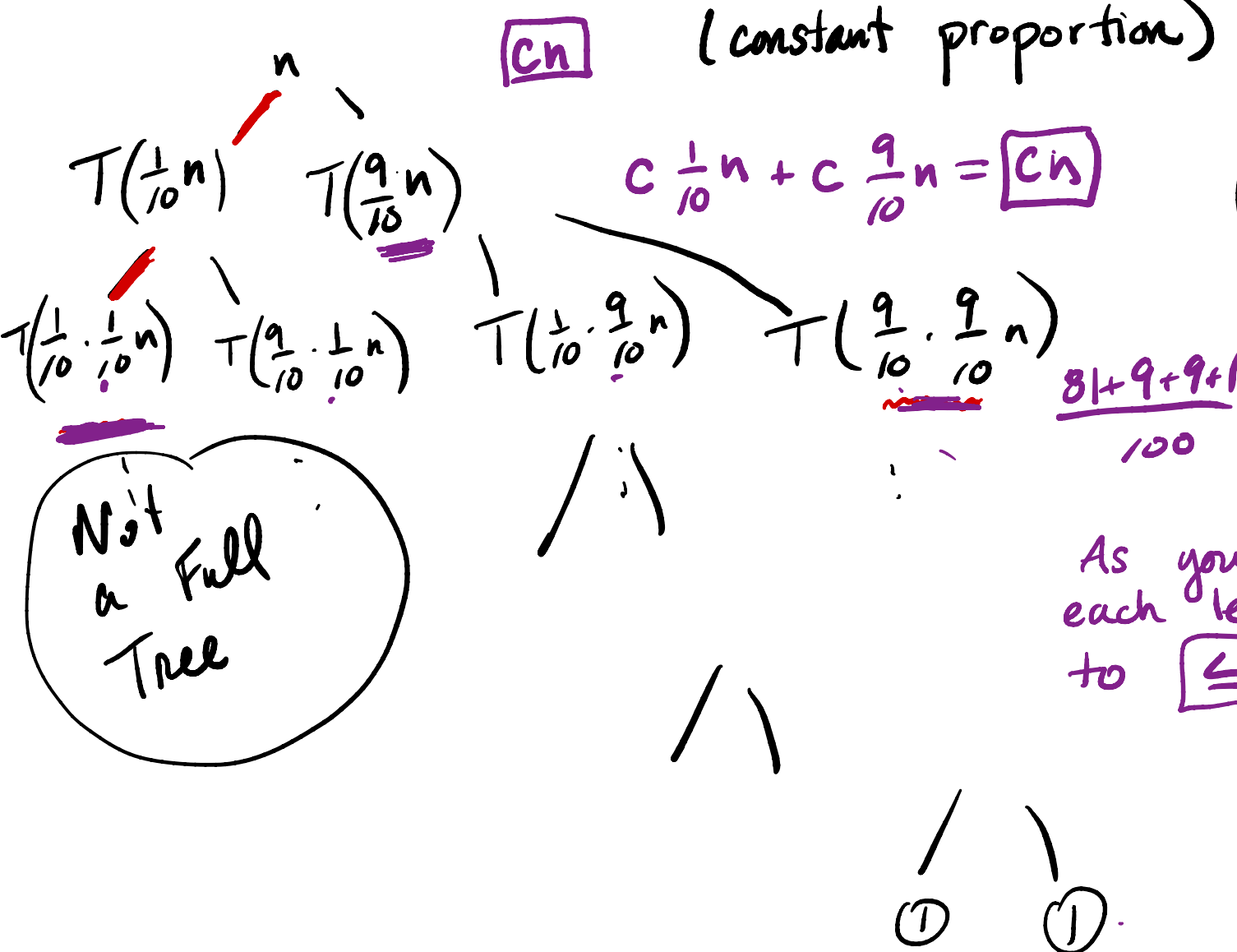
Quicksort

Question: What happens when you always have a partition with 4 elements on the left and $n - 5$ elements on the right?



Quicksort

Question: What happens if we have a strategy that gives us a 1:9 split every time?



Recurrence relation:

$$T(n) = T(\tfrac{n}{10}) + T(\tfrac{9n}{10}) + \Theta(n)$$

Work done at each level?

As you go down each levels sum to $\leq cn$

★ There is at most cn work done at each level.

Quicksort

Question: What happens if we have a strategy that gives us a 1:9 split every time?

How many levels?

longest path is when $\left\lceil \frac{n}{(10/9)^k} \right\rceil \leq 1$

K - number of levels.

$$n \leq \left(\frac{10}{9}\right)^k$$

$$\log_{10/9} n \leq k$$

$$\log_{10/9} n + \varepsilon = k \quad \text{where } 0 \leq \varepsilon < 1$$

number of levels $\approx \log_{10/9} n$

$\Rightarrow$ Total work done $\leq cn \log_{10/9} n \Rightarrow \boxed{O(n \lg n)}$

Quicksort

"Good splits" - splits of the type $\left(\frac{n-1}{b}, (n-1)\left(1-\frac{1}{b}\right)\right)$

- Any split of constant proportionality yields a recursion tree with depth $\Theta(\log n)$

"Bad splits" - splits of the type $(n-k, k-1)$ for some constant k.

"Deterministic" Quicksort

→ choosing last item of subarray as pivot

→ performance - highly dependent on the kind of split produced,

Fair assumption: A is randomly shuffled, then the good and bad splits happen randomly

Hope! A random pivot is "pretty good" "often enough"

Quicksort

Question: What if we have a mixture of good and bad splits?

- It turns out that getting a good split a constant fraction of the time is enough to obtain $\Theta\big(nlg(n)\big)$ running time.

Quicksort Modifications

Finding a good pivot is key to getting quicksort to work efficiently.

“Good splits” - splits of the type $\left(\frac{n-1}{b}, (n-1)\left(1-\frac{1}{b}\right)\right)$

“Bad splits” - splits of the type $(n-k, k-1)$ for some constant k.

Quicksort Modifications

How to avoid a “bad split”?

Possible Variants:

- Always pick the first element as pivot
- Always pick the last element as pivot .
- Randomized Quicksort - Pick a random element as pivot
- Hoare’s Partition Scheme
- Pick the median as pivot
  - Median of 3
  - Median of Medians
- Hybrid of Quicksort and Insertion Sort
- Others!

we discuss this here!

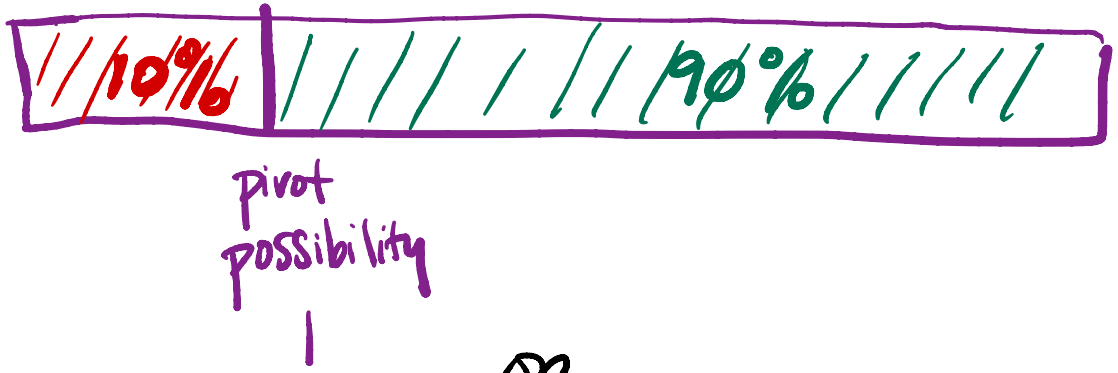
HW

Random Pivots

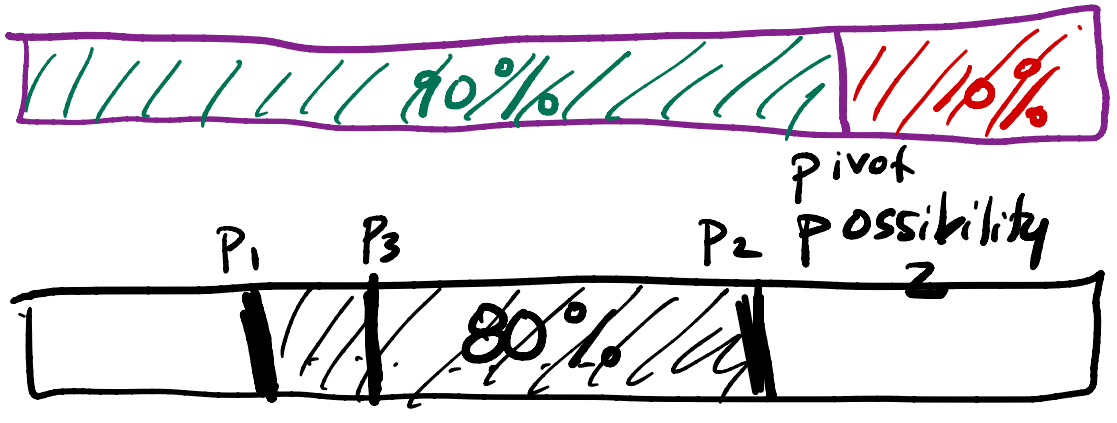
Hope: A random pivot is “pretty good” “often enough”.

Suppose we are happy with 10-90 splits. What is the probability that a random pivot achieves that or better?

Imagine a sorted list A



or



If a pivot is picked from this range, it produces a 10-90 split or better.

probability is $\frac{80}{100} = 0.8$

Randomized Quicksort

Two ways to implement **Randomized Quicksort**:

1. RANDOMIZED-PARTITION(A, p, r)
   i = RANDOM(p, r)
   exchange A[r] with A[i]
   return PARTITION(A, p, r)

Quicksort ~ calls Randomized-partition

2. PARTITION(A, p, r)
   exchange $A[r]$ with $A[random(p, r)]$
   $x = A[r]$
   $i = p - 1$
   for $j = p$ **to** $r - 1$
      if $A[j] \leq x$
         $i = i + 1$
         exchange $A[i]$ with $A[j]$
   exchange $A[i + 1]$ with $A[r]$
   return $i + 1$

Medians and Quicksort

Finding a good pivot is key to getting quicksort to work efficiently.

❖ **Having the pivot be the median of your elements is best.**

Median of 3: Randomly select 3 items from array. Select the median. Use that element as the pivot in the original array.

Median of Medians: Approximation of the median of array found in $\Theta(n)$ time. Use the approximated median as the pivot.

Median of 3

Median of 3: Randomly select 3 items from array. Select the median. Use that element as the pivot in the original array.

- one way to choose is to look at the first, middle, and last elements of the array.
- choose the median of those 3 as our pivot.

A: [8 1 2 4 3 7 9]

median of these 3: 4, 8, 9

pivot

[1 2 4 3 7 8 9]

Median of Medians

Recall: the median is the middle element.

Median of Medians: Approximation of the median of array found in $\Theta(n)$ time. Use the approximation median as the pivot.

1. Given array A, divide A into groups of 5 elements.

$$A = [12, 13, 14, 6, 1, 5, 9, 4, 3, 2, 17, 18, 7, 23, 24]$$

2. Sort each group. Each group is constant size. Sorting is $\Theta(n)$

[1, 6, 12, 13, 14 | 2, 3, 4, 5, 9 | 7, 17, 18, 23, 24]

3. Take the median of each group.

12, 4, 18

4. Take the median of medians and use that as the pivot.

order the medians. 4, 12, 18 ⇒ median of medians = 12

Choose this as our pivot.

Median of Medians

Median of Medians: Approximation of the median of array found in $\Theta(n)$ time. Use the approximation of the median as the pivot.

- $A = [12, 13, 14, 6, 1, 5, 9, 4, 3, 2, 17, 18, 7, 23, 24]$

Using 12 as a pivot, $A = [6, 1, 5, 9, 4, 3, 2, 7, \mathbf{12}, 13, 14, 17, 18, 23, 24]$

median of medians

- 12 is not the median, but its a pretty good approximation.

Median of Medians

Question: What can we say about using the median of medians as a pivot?

What is the worst case split that I get?

Consider

[1 2 3 4 5 | 6 7 8 9 10 | 11 12 13 14 15 | 16 17 18 19 20 | 21 22 23 24 25]

medians: [3, 8, 13, 18, 23] ← size: $\frac{1}{5}n$

median of medians (m): 13

what do we know. well looking at the medians list: $13 > 8$, $13 > 3$ } $m \geq \frac{1}{10}n$

And because 3 and 8 are both medians of their own sublist,

$\Rightarrow m > \frac{n}{10}$ elements - due to relationship with 3

$> \frac{n}{10}$ elements - due to relationship with 8

Median of Medians

Question: What can we say about using the median of medians as a pivot?

What is the worst case split that I get?

Thus by virtue of being the median of medians, m ≥ than at least $\frac{3}{10}$ of elements in original list.

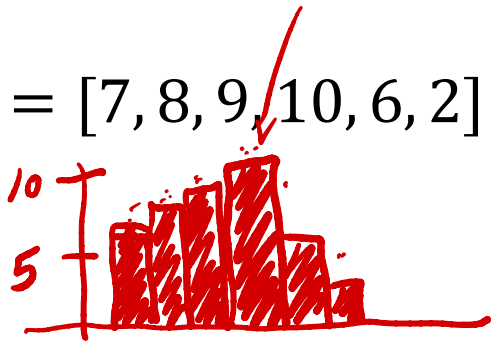
$\Rightarrow m \geq \underbrace{\frac{3}{10} n}_{\text{number of elements } m \text{ is larger than.}}$ ← input array size

So by choosing m, we have at worst a 30:70 split.

Peaked Arrays

A **peaked array** A is defined as an array where the subarray $A[0, \ldots, i]$ has the property that $A[j] < A[j+1]$ for $0 \leq j < i$ and the subarray $A[i \ \ldots n-1]$ has the property that $A[j] > A[j+1]$ for $i \leq j < n-1$. The max value in the array is $A[i]$.

e.g. $A = [7, 8, 9, 10, 6, 2]$ is a peaked array.



Example: Write a recursive $\Theta(\lg n)$ algorithm to find the peak for the given array A.

- $\Theta(\log n)$ hints that this might be some sort of divide and conquer Routine.
- we don't need to sort
- Think about Binary Search. modify.
- What is the Base Case? $n = 3$
  - when we get down to 3 elements, the peak is the middle.

peak

[.... 10 20 10 ....]

Peaked Arrays

Example (continued): Write a recursive $\Theta(\lg n)$ algorithm to find the peak for the given array A.

begin index

end index

```
findPeak(A, p, r)
    mid = ⌊(p+r)/2⌋
    #Check for Base case
    if r-p+1 == 3
        return A[mid]
    elseif A[mid-1] < A[mid]
        return A[mid]
    elseif: A[mid] < A[mid+1]
        findPeak(mid, r)
    else:
        findPeak(A, p, mid)
```

if we get to the base case, A[mid] must be the peak.

AND A[mid] > A[mid+1]

Other Ways to Choose Pivot

Example: Consider the following strategy for choosing a pivot elements for the Partition subroutine of Quicksort, applied to an array A.

- Let n be the number of elements of the array A.
- If $n \leq 15$, perform an Insertion Sort of A and return.
- Otherwise:
  - Choose $2\lfloor\sqrt{n}\rfloor$ elements at random from A; let S be the new list with the chosen elements.
  - Sort the list S using Insertion Sort and store the median of S as m.
  - Partition the sub-array of A using m as a pivot.
  - Carry out QuickSort recursively on the two parts.

Using the following array A with n=20, show one iteration of this partitioning strategy on the array

$$A = [34, 45, 32, 1, 23, 90, 12, 13, 43, 54, 65, 76, 67, 56, 45, 34, 44, 55, 23, 2]$$

Other Ways to Choose Pivot

Example (continued):

- $A = [34, 45, 32, 1, 23, 90, 12, 13, 43, 54, 65, 76, 67, 56, 45, 34, 44, 55, 23, 2]$

How many elements? $2\lfloor\sqrt{n}\rfloor = 2\lfloor\sqrt{20}\rfloor = 2 \cdot 4 = 8$

$S = [34, 45, 32, 1, 23, 90, 12, 13]$

Sorted S: $[1, 12, 13, 23, 32, 34, 45, 90]$

median $= \frac{23+32}{2} = 27.5$ ← choose this as the partition

pivot 27.5

$[1\ 23\ 12\ 13\ 23\ 2]$ $[34\ 45\ 32\ 90\ 43\ 54\ 65\ 76\ 67\ 56\ 45\ 34\ 44\ 55]$ } use this on A.

6 elements less than pivot

14 elements greater than the pivot.

Other Ways to Choose Pivot

Example (continued): If the element m obtained as the median of S is used as the pivot, what can we say about the sizes of the two partitions of the array A? Hint: Think about the best and worst possible selections for the values in S.

we always choose $2\lfloor\sqrt{n}\rfloor$ elements.

(w.l.o.g. smallest)

worst case: we choose $2\lfloor\sqrt{n}\rfloor$ largest elements in A

pick median from those

$\Rightarrow$ By nature of a median:

this partition of A will have at least $\lfloor\sqrt{n}\rfloor$ elements larger than the pivot

and $n - \lfloor\sqrt{n}\rfloor$ elements less than the pivot.

Other Ways to Choose Pivot

Example (continued): How much time does it take to sort S and find its median? Give a Θ bound.

- With Insertion Sort, the set S gets sorted in worst-case $\Theta(n)$ steps.
- Finding the median take $\Theta(1)$ steps.

Other Ways to Choose Pivot

Example (continued): Write a recurrence relation for the worst case running time of QuickSort with this pivoting strategy.

$$T(n) = \begin{cases} T(\sqrt{n}) + T(n-\sqrt{n}) + c_1 n & n > 15 \\ c_0 & n \leq 15 \end{cases}$$

Next Time

- Hash Tables