Selection and Quicksort

Class 17

Median

- the median of a list of numbers is the middle element
- half the elements are larger, half smaller
- e.g., the median of (45, 1, 10, 30, 25) is 25

Median

- the median of a list of numbers is the middle element
- half the elements are larger, half smaller
- e.g., the median of (45, 1, 10, 30, 25) is 25
- computing the median is easy:

Median

- the median of a list of numbers is the middle element
- half the elements are larger, half smaller
- e.g., the median of (45, 1, 10, 30, 25) is 25
- computing the median is easy: sort the numbers and pick the n/2 element
- unfortunately, the drawback is that sorting is in $O(n \lg n)$
- we should be able to do better
- we just need the n/2-th largest, not everything in order

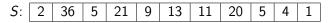
 the median problem is actually a specific example of the selection problem

Selection

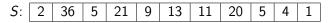
Given a list of numbers S and a specific value k, find the k-th smallest number in S.

- for example, if k = 1, we want the smallest element in S
- if k = n/2, then we are asking for the median element of S

- we can solve selection using divide and conquer
- given a list of numbers S, and an arbitrary value v, we can divide S into three categories
 - 1. elements smaller than v
 - 2. elements equal to v (duplicates are allowed)
 - 3. elements greater than v
- how many steps to do this?



- we can solve selection using divide and conquer
- given a list of numbers S, and an arbitrary value v, we can divide S into three categories
 - 1. elements smaller than v
 - 2. elements equal to v (duplicates are allowed)
 - 3. elements greater than v
- how many steps to do this? $2n \in \Theta(n)$



S_L :	2	4	1	S_v :	5	5	S_R :	36	21	9	13	11	20

- we wish to find the k-smallest value
- it must be in one of the three sublists
- if k = 8, it must be in S_R : len $(S_L) = 3$ and len $(S_v) = 2$
- furthermore, it must be the 3rd-smallest value in S_R

S_L :	2	4	1	S_v :	5	5	S_R :	36	21	9	13	11	20

- we wish to find the k-smallest value
- it must be in one of the three sublists
- if k = 8, it must be in S_R : len $(S_L) = 3$ and len $(S_v) = 2$
- furthermore, it must be the 3rd-smallest value in S_R
- thus we can recurse: selection(S, 8) = selection(S_R , 3)

more generally

$$\mathsf{selection}(S,k) = \begin{cases} \mathsf{selection}(S_L,k) & \text{if } k \leq \mathsf{len}(S_L) \\ v & \text{if } \mathsf{len}(S_L) < k \leq \mathsf{len}(S_L) + \mathsf{len}(S_v) \\ \mathsf{selection}(S_R,k-\mathsf{len}(S_L)-\mathsf{len}(S_v)) & \text{otherwise} \end{cases}$$

 in one round of recursion, we shrink the size of input from len(S) to at most the larger of len(S_L) and len(S_R)

- but what is v?
- we need it to be easy to pick
- and ideally it needs to be a value so that both $len(S_L)$ and $len(S_R)$ are approximately equal in size
- what happens if it is a "bad" value?

Picking the Pivot

- v is called the pivot value
- three main schemes are used to pick the pivot
 - 1. use the first element of the current S
 - pro: it's fastest
 - pro: after the first round, it's "unlikely" to be very close to smallest or largest
 - con: you could be unlucky and get very large or small element
 - 2. use the middle of the 0, n/2, n-1 elements
 - pro: it's extremely fast
 - pro: it's much more likely to be medium than 1 or 3
 - 3. literally pick a random element
 - con: slowest
 - con: you could be unlucky and get large or small element
- questions about selection?

Tony Hoare

- Sir Charles Antony Richard Hoare
- born 1934 in Sri Lanka
- read classics and philosophy at Merton College, Oxford
- master's in statistics at Oxford
- studied in Moscow under Kolmogorov (you should look him up)
- implemented ALGOL 60 which led to C. Pascal, and Ada
- invented the null reference, his "billion-dollar mistake"



invented quicksort algorithm

Quicksort

• identical first part to selection

if the current range is > 2:

- 1. pick one element to be the PIVOT
- rearrange all smaller elements to the left and all bigger elements to the right (pivot is now in its final location in the array)
- 3. recurse on the left and right ranges

```
void quicksort(array, left, right)
{
  if (left < right)
  {
    pivot_index = partition(array, left, right);
    quicksort(array, left, pivot_index - 1);
    quicksort(array, pivot_index + 1, right);
  }
}</pre>
```

```
void quicksort(array, left, right)
{
  if (left < right)
  {
    pivot_index = partition(array, left, right);
    quicksort(array, left, pivot_index - 1);
    quicksort(array, pivot_index + 1, right);
  }
}</pre>
```

$$T(n) = T(pivot_index - left) + T(right - pivot_index) + n$$

ugh, we're stuck

$$T(n) = T(pivot_index - left) + T(right - pivot_index) + n$$

best case:

$$T(n) = T(pivot_index - left) + T(right - pivot_index) + n$$

• best case: pivot is always exactly in the middle

$$T(n) \geq 2T\left(\frac{n}{2}\right) + n$$

worst case:

$$T(n) = T(pivot_index - left) + T(right - pivot_index) + n$$

• best case: pivot is always exactly in the middle

$$T(n) \geq 2T\left(\frac{n}{2}\right) + n$$

worst case: pivot is always exactly at one end

$$T(n) \leq T(n-1) + n$$

solving each independently, overall we have:

$$T(n) = T(pivot_index - left) + T(right - pivot_index) + n$$

• best case: pivot is always exactly in the middle

$$T(n) \ge 2T\left(\frac{n}{2}\right) + n$$

worst case: pivot is always exactly at one end

$$T(n) \leq T(n-1) + n$$

solving each independently, overall we have:

$$T(n) \in O(n^2)$$

 $\in \Omega(n \lg n)$

Quicksort Pros and Cons

- pros
 - good pivot choice makes worst case extremely unlikely
 - good partition algorithm reduces overhead to almost zero
 - pivot is never re-visited, further reducing overhead
 - sorting is strictly in-place, requiring zero extra space
- cons
 - the clever tweaks can be quite complex
 - easy to code incorrectly

Sorting Analysis

algorithm	lower	upper
insertion	n^2	n^2
merge	n lg n	n lg n
heap	n lg n	n lg n
quick	n lg n	n ²