Tha	Selection Problem		
Order Statistics			
Order Statistics			
We often want to comput	te a median of a list of val	ues.	
-	picture than the average s		
More generally what eleg	nent has position <i>k</i> in the s	corted list?	
(For example, for percent			
、	,		
Selection Problem			
	ist A of size <i>n</i> , and an inte		
what elemer	nt is at position <i>k</i> in the so	orted list?	
		C : 0010	1 (00
SI 335 (USNA)	Unit 5	Spring 2013	1 / 39
The	Selection Problem		
Sorting-Based Soluti			
-			
 First idea: Sort, then 	ı look-up		
Second idea: Cut-off	selection sort		
			2 / 25
SI 335 (USNA)	Unit 5	Spring 2013	2 / 39
The	Selection Problem		
Heap-Based Solution			
 First idea: Use a size 	e-k max-heap		
Second idea: Use a s	size- <i>n</i> min-heap		
			1

Unit 5

Algorithm Design
What algorithm design paradigms could we use to attack the selection problem?
 Reduction to known problem What we just did!
 Memoization/Dynamic Programming Would need a recursive algorithm first
 Divide and Conquer Like binary search — seems promising. What's the problem?
SI 335 (USNA) Unit 5 Spring 2013 4 / 39
 A better "divide" Consider this array: A = [60, 43, 61, 87, 89, 87, 77, 11, 49, 45] Difficult: Finding the element at a given position. For example, what is the 5th-smallest element in A? Easier: Finding the <i>position</i> of a given element. For example, what is the position of x = 77 in the sorted order? Idea: Pick an element (the pivot), and sort around it.
SI 335 (USNA) Unit 5 Spring 2013 5 / 39
QuickSelect
Partition Algorithm
Input : Array A of size n. Pivot is in A[0]. Output : Index p such that $A[p]$ holds the pivot, and $A[a] \le A[p] < A[b]$ for all $0 \le a .$
def partition(A): n = len(A)

SI 335 (USNA)

Unit 5

Spring 2013 6 / 39

	QuickSelect		
Analysis of partiti	on		
• Loop Invariant: Eve	rything before $A[i]$ is \leq the pivot is greater than the pivot.		
• Running time: Cons	ider the value of $j - i$.		
SI 335 (USNA)	Unit 5	Spring 2013	7 / 39
	QuickSelect		
Choosing a Pivot			
The choice of pivot is real • Want the partitions t • What would be the ve	o be close to the same size.		
Initial "dumb" idea: Just	pick the first element:		
Input : Array <i>A</i> of length Dutput : Index of the pive	n		
<pre>def choosePivot1(A) return 0</pre>	:		
SI 335 (USNA)	Unit 5	Spring 2013	8 / 39
SI 335 (USNA)	Unit 5	Spring 2013	8 / 39

Input: Array A of length n, and integer k**Output**: Element at position k in the sorted array

```
def quickSelect1(A, k):
    n = len(A)
    swap(A, 0, choosePivot1(A))
    p = partition(A)
    if p == k:
        return A[p]
    elif p < k:
        return quickSelect1(A[p+1 : n], k-p-1)
    elif p > k:
        return quickSelect1(A[0 : p], k)
```

	Analysis of QuickSelect		
QuickSelect: Initi	al Analysis		
• Best case:			
Worst case:			
SI 335 (USNA)	Unit 5	Spring 2013	10 / 39

Analysis of QuickSelect

Average-case analysis

Assume all n! permutations are equally likely. Average cost is sum of costs for all permutations, divided by n!.

Define T(n, k) as average cost of quickSelect1(A,k):

$$T(n,k) = n + \frac{1}{n} \left(\sum_{p=0}^{k-1} T(n-p-1,k-p-1) + \sum_{p=k+1}^{n-1} T(p,k) \right)$$

See the book for a precise analysis, or...

SI 335 (USNA)

Unit 5

Spring 2013 11 / 39

12 / 39

Analysis of QuickSelect Average-Case of quickSelect1 First simplification: define $T(n) = \max_k T(n, k)$ The key to the cost is the **position of the pivot**. There are *n* possibilities, but can be grouped into: • **Good pivots**: The position *p* is between *n*/4 and 3*n*/4. Size of recursive call: • **Bad pivots**: Position *p* is less than *n*/4 or greater than 3*n*/4 Size of recursive call: Each possibility occurs $\frac{1}{2}$ of the time. SI 335 (USNA) Unit 5 Duit 5

	Analysis of QuickSelect	
Average-Case of a	-	
-	the probability of each possibilit	ty, we have:
	$1_{3n} = 1_{3n}$	
7	$T(n) \leq n + \frac{1}{2}T\left(\frac{3n}{4}\right) + \frac{1}{2}T(n)$	
(Assumption: every pe	ermutation in each partition is al	so equally likely.)
SI 335 (USNA)	Unit 5	Spring 2013 13 / 39
0.000 (00.00)	one o	op.mg 2010 10 / 00
Ran	ndomized Pivot Choosing	
	ndomized Pivot Choosing erage-Case Analysis	
	° °	
	° °	
Drawbacks of Ave	erage-Case Analysis	
Drawbacks of Ave To get the average-ca	erage-Case Analysis se we had to make some BIG ass	umptions:
Drawbacks of Ave To get the average-ca • Every permutatio	erage-Case Analysis	
Drawbacks of Ave To get the average-ca • Every permutatio • Every permutatio	erage-Case Analysis se we had to make some BIG ass on of the input is equally likely on of each half of the partition is	still equally likely
Drawbacks of Ave To get the average-ca • Every permutatio • Every permutatio	erage-Case Analysis se we had to make some BIG ass on of the input is equally likely	still equally likely
Drawbacks of Ave To get the average-ca • Every permutatio • Every permutatio	erage-Case Analysis se we had to make some BIG ass on of the input is equally likely on of each half of the partition is	still equally likely
Drawbacks of Ave To get the average-ca • Every permutatio • Every permutatio	erage-Case Analysis se we had to make some BIG ass on of the input is equally likely on of each half of the partition is	still equally likely
Drawbacks of Ave To get the average-ca • Every permutatio • Every permutatio	erage-Case Analysis se we had to make some BIG ass on of the input is equally likely on of each half of the partition is	still equally likely

Randomized Pivot Choosing

Randomized algorithms

Randomized algorithms use a source of ${\bf random\ numbers}$ in addition to the given input.

AMAZINGLY, this makes some things faster!

Idea: Shift assumptions on the *input distribution* to assumptions on the *random number distribution*. (Why is this better?)

Specifically, assume the function random(n) returns an integer between 0 and n-1 with uniform probability.

```
Randomized Pivot Choosing
Randomized quickSelect
We could shuffle the whole array into a randomized ordering, or:
 ① Choose the pivot element randomly:
Randomized pivot choice
def choosePivot2(A):
     \ensuremath{\texttt{\#}} This returns a random number from 0 up to n-1
     return randrange(0, len(A))
 ② Incorporate this into the quickSelect algorithm:
Randomized selection
def quickSelect2(A, k):
     swap(A, 0, choosePivot2(A))
     # ... the rest is the same as quickSelect1
      SI 335 (USNA)
                                 Unit 5
                                                        Spring 2013 16 / 39
                   Randomized Pivot Choosing
Analysis of quickSelect2
```

SI 335 (USNA)

Unit 5

The **expected cost** of a randomized algorithm is the probability of each

Two cases: good pivot or bad pivot. Each occurs half of the time...

possibility, times the cost given that possibility.

We will focus on the expected worst-case running time.

The analysis is exactly the same as the average case!

Expected worst-case cost of quickSelect2 is $\Theta(n)$.

Spring 2013 17 / 39

Do we need randomization?

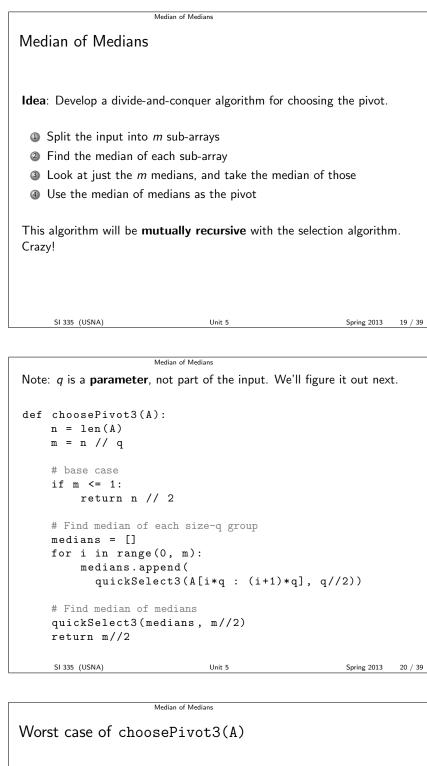
Why is this better than average-case?

Can we do selection in linear time without randomization?

Median of Medians

Blum, Floyd, Pratt, Rivest, and Tarjan figured it out in 1973.

But it's going to get a little complicated. . .



Assume all array elements are distinct.

Question: How unbalanced can the pivoting be?

- ${\ }$ At least $\lceil m/2\rceil$ medians must be \leq the chosen pivot.
- At least $\lceil q/2 \rceil$ elements are \leq each median.
- So the pivot must be greater than or equal to at least

$\left\lceil \frac{m}{2} \right\rceil \cdot \left\lceil \frac{q}{2} \right\rceil$

elements in the array, in the worst case.

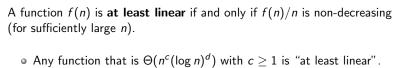
 ${\ensuremath{\,\circ\,}}$ By the same reasoning, as many elements must be ${\ensuremath{\,\geq\,}}$ the chosen pivot.

	Median of Med	lians		
Worst-ca	ise example, $q = 3$	}		
A	A = [13, 25, 18, 76, 39, 5]	1, 53, 41, 96, 5, 19, 72, 20, 6	3, 11]	
SI 335	(USNA)	Unit 5	Spring 2013	22 / 39

Median of Medians

Aside: "At Least Linear"

Definition



- You can pretty much assume that any running time that is Ω(n) is "at least linear".
- Important consequence: If T(n) is at least linear, then $T(m) + T(n) \le T(m+n)$ for any positive-valued variables n and m.

SI 335 (USNA)

Unit 5

Spring 2013 23 / 39

Analysis of quickSelect3

Since quickSelect3 and choosePivot3 are **mutually recursive**, we have to analyze them together.

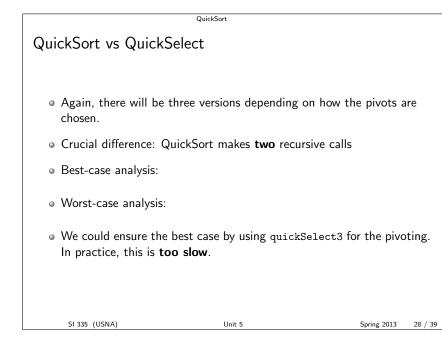
• Let T(n) =worst-case cost of quickSelect3(A,k)

Median of Medians

- Let S(n) = worst-case cost of selectPivot3(A)
- T(n) =
- S(n) =
- Combining these, T(n) =

Unit 5

oosing <i>q</i> What if <i>q</i> is big? Try <i>q</i> = <i>n</i> What if <i>q</i> is small? Try <i>q</i> =		
What if q is big? Try $q = n$		
• What if q is small? Try q =	- 3.	
• What if <i>q</i> is small? Try <i>q</i> =	- 3.	
What if <i>q</i> is small? Try <i>q</i> =	= 3.	
What if <i>q</i> is small? Try <i>q</i> =	- 3.	
What if <i>q</i> is small? Try <i>q</i> =	= 3.	
SI 335 (USNA)	Unit 5	Spring 2013 25 / 3
	one s	
Median of M	edians	
oosing <i>q</i>		
hat about $q = 5$?		
SI 335 (USNA)	Unit 5	Spring 2013 26 / 3
	ckSort	
ickSort		
ickSelect is based on a sorting	g method develope	ed by Hoare in 1960:
	- 1	-
<pre>f quickSort1(A): n = len(A)</pre>		
if n > 1:		
swap(A, O, choos		
p = partition(A)		,1)
A[O : p] = quick A[p+1 : n] = qui		
return A		
SI 335 (USNA)		



QuickSort

Average-case analysis of quickSort1

Of all n! permutations, (n-1)! have pivot A[0] at a given position i.

Average cost over all permutations:

$$T(n) = \frac{1}{n} \sum_{i=0}^{n-1} (T(i) + T(n-i-1)) + \Theta(n), \qquad n \ge 2$$

Do you want to solve this directly?

Instead, consider the **average depth** of the recursion. Since the cost at each level is $\Theta(n)$, this is all we need.

SI 335 (USNA)

Unit 5

QuickSort

Spring 2013 29 / 39

Average depth of recursion for quickSort1

D(n) = average recursion depth for size-*n* inputs.

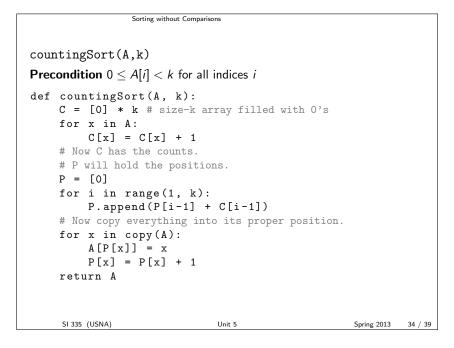
$$H(n) = \begin{cases} 0, & n \le 1 \\ 1 + \frac{1}{n} \sum_{i=0}^{n-1} \max \left(H(i), H(n-i-1) \right), & n \ge 2 \end{cases}$$

• We will get a **good pivot** $(n/4 \le p \le 3n/4)$ with probability $\frac{1}{2}$

• The *larger* recursive call will determine the height (i.e., be the "max") with probability at least $\frac{1}{2}$.

Unit 5

	QuickSort		
Summary of QuickS	ort analysis		
 quickSort1: Choose Worst-case: Θ(n² Average case: Θ(²)		
 quickSort2: Choose Worst-case: Θ(n² Expected case: θ 	²)		
 quickSort3: Use the Worst-case: Θ(n 	e median of medians to cho log n)	oose pivots.	
SI 335 (USNA)	Unit 5	Spring 2013	31 / 39
Sorting wi	ithout Comparisons		
Sorting so far			
We have seen: • Quadratic-time algor	rithms:		
BubbleSort, Selectio	nSort, InsertionSort		
 n log n-time algorithr HeapSort, MergeSort 			
O(n log n) is asymptotic	ally optimal in the compa	arison model.	
So how could we do bette	er?		
SI 335 (USNA)	Unit 5	Spring 2013	32 / 39
BucketSort	ithout Comparisons		
BucketSort is a general a	pproach, not a specific alg	orithm:	
	tputs into k groups or bu		
	y, put each element into its each bucket (perhaps recu		
3 Sort the elements in4 Dump sorted buckets		a sively)	
·			
Notice : No comparisons!	!		



Analysis of CountingSort • Time: • Space:

Sorting without Comparisons

Stable Sorting
Definition

A sorting algorithm is stable if elements with the same key stay in the same order.

Quadratic algorithms and MergeSort are easily made stable

QuickSort will require extra space to do stable partition.
CountingSort is stable.

radixSort(A,d,B) **Input**: Integer array A of length n, and integers d and B such that every A[i] has d digits $A[i] = x_{d-1}x_{d-2}\cdots x_0$, to the base B. **Output**: *A* gets sorted. def radixSort(A, d, B): for i in range(0, d): countingSort(A, B) # based on the i'th digits return A Works because CountingSort is stable! Analysis: SI 335 (USNA) Unit 5 Spring 2013 37 / 39

Sorting without Comparisons

Sorting without Comparisons

Summary of Sorting Algorithms

Every algorithm has its place and purpose!

Algorithm	Analysis	In-place?	Stable?	
SelectionSort	$\Theta(n^2)$ best and worst	yes	yes	
InsertionSort	$\Theta(n)$ best, $\Theta(n^2)$ worst	yes	yes	
HeapSort	$\Theta(n \log n)$ best and worst	yes	no	
MergeSort	$\Theta(n \log n)$ best and worst	no	yes	
QuickSort	$\Theta(n \log n)$ best, $\Theta(n^2)$ worst	yes	no	
CountingSort	$\Theta(n+k)$ best and worst	no	yes	
RadixSort	$\Theta(d(n+k))$ best and worst	yes	yes	
SI 335 (USNA)	Unit 5		Spring 2013	38 / 39

Spring 2013 38 / 39

Sorting without Comparisons Unit 5 Summary • Selection problem Partition quickSelect and quickSort • Average-case analysis • Randomized algorithms and analysis Median of medians • Non-comparison based sorting • BucketSort, CountingSort, RadixSort • Stable sorting Unit 5 SI 335 (USNA) Spring 2013 39 / 39