Order Statistics

We often want to compute a **median** of a list of values. (It gives a more accurate picture than the average sometimes.)

More generally, what element has position k in the sorted list? (For example, for percentiles or trimmed means.)

Selection Problem

Given a list A of size n, and an integer k, what element is at position k in the sorted list?

Unit 5

CS 355 (USNA)

Spring 2012 1 / 39

Sorting-Based	l Solutions		
First idea: S	ort, then look-up		
● Second idea	: Cut-off selection sort		
CS 355 (USNA)	Unit 5	Spring 2012	2 / 39

Heap-Based Solutions

- First idea: Use a size-k max-heap
- Second idea: Use a size-*n* min-heap

Unit 5

Algorithm Design		
What algorithm design p a problem?	aradigms could we use	to attack the selection
 Reduction to known p What we just did! 	roblem	
 Memoization/Dynamic Would need a recursiv 		
 Divide and Conquer Like binary search — s 	seems promising. What	's the problem?
CS 355 (USNA)	Unit 5	Spring 2012 4 / 39
A better "divide"		
 Finding the element a 	t a given position is tou	ıgh.

Idea: Pick an element (the **pivot**), and sort around it.

CS 355 (USNA)

Unit 5

Spring 2012 5 / 39

6 / 39

```
partition(A)
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 .
  1 i := 1
  2 j := n −1
    while i <= j do
  3
        if A[i] <= A [0] then</pre>
  4
          i := i + 1
  5
        else if A[j] > A[0] then
  6
          j := j - 1
  7
  8
        else
          swap (A[i], A[j])
  9
    end while
  10
    swap (A[O], A[j])
 11
     return j
  12
     CS 355 (USNA)
                             Unit 5
                                                    Spring 2012
```

Analysis of partition			
 Loop Invariant: Everything before everything after A[j] is greater the third of the second sec			
• Running time : Consider the val	lue of <i>j — i</i> .		
CS 355 (USNA) Un	nit 5	Spring 2012	7 / 39
Choosing a Pivot			
The choice of pivot is really importan • Want the partitions to be close t • What would be the very best cho	to the same size.		
Initial (dumb) idea: Just pick the firs	at element:		
choosePivot1(A) Input: Array A of length n Output: Index of the pivot element w	we want		
1 return 0			
CS 355 (USNA) Un	iit 5	Spring 2012	8 / 39

The Algorithm quickSelect1(A,k) **Input**: Array *A* of length *n*, and integer *k* **Output**: Element at position *k* in the sorted array 1 swap (A[0], A[choosePivot1(A)]) 2 p := partition(A) 3 if p = k then 4 return A[p] 5 else if p < k then return quickSelect1(A[p+1..n-1], k-p-1) 6 7 else if p > k then return quickSelect1(A[0..p-1], k) 8 CS 355 (USNA) Unit 5 Spring 2012 9 / 39

QuickSelect:	Initial Analysis	
Best case:		
Worst case:		
CS 355 (USNA)	Unit 5 Spring 2012	10 / 39

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...

CS 355 (USNA)

Unit 5

Spring 2012 11 / 39

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.

Average-Case of quickSelect1

Based on the cost and the probability of each possibility, we have:

$$T(n) \leq n + \frac{1}{2}T\left(\frac{3n}{4}\right) + \frac{1}{2}T(n)$$

(Assumption: every permutation in each partition is also equally likely.)

CS 355 (USNA)

Unit 5

Spring 2012 13 / 39

Drawbacks of Average-Case Analysis

To get the average-case we had to make some BIG assumptions:

- Every permutation of the input is equally likely
- Every permutation of each half of the partition is still equally likely

The first assumption is actually false in most applications!

CS 355 (USNA)

Unit 5

Spring 2012 14 / 39

15 / 39

Randomized algorithms	
Randomized algorithms use a source of random numbers in addition to the given input.	
AMAZINGLY, this makes some things faster!	
Idea : Shift assumptions on the <i>input distribution</i> to assumptions on the <i>random number distribution</i> . (Why is this better?)	
Specifically, assume the function $random(n)$ returns an integer between 0 and n-1 with uniform probability.	ı
CS 355 (USNA) Unit 5 Spring 2012	1

Randomized quickSelect

We could shuffle the whole array into a randomized ordering, or:

Choose the pivot element randomly:

```
choosePivot2(A)
```

1 return random(n)

② Incorporate this into the quickSelect algorithm:

```
quickSelect2(A)
```

```
1 swap (A[0], A[choosePivot2(A)])
2 ...
```

CS 355 (USNA)

Unit 5

Spring 2012 16 / 39

Analysis of quickSelect2
The expected cost of a randomized algorithm is the probability of each possibility, times the cost given that possibility.
We will focus on the expected worst-case running time.
Two cases: good pivot or bad pivot. Each occurs half of the time... The analysis is exactly the same as the average case!
Expected worst-case cost of quickSelect2 is Θ(n). Why is this better than average-case?

CS 355 (USNA)

Unit 5

Spring 2012 17 / 39

Do we need randomization?

Can we do selection in linear time without randomization?

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

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

Median of Medians

Idea: Develop a divide-and-conquer algorithm for choosing the pivot.

- Split the input into *m* sub-arrays
- 2 Find the median of each sub-array
- 3 Look at just the *m* medians, and take the median of those
- ④ Use the median of medians as the pivot

This algorithm will be $\ensuremath{\textbf{mutually recursive}}$ with the selection algorithm. Crazy!

CS 355 (USNA)

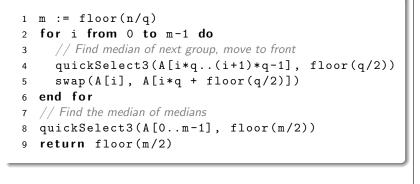
Unit 5

Spring 2012 19 / 39

Note:

- q is a parameter, not part of the input. We'll figure it out next.
- quickSelect3(A,k) finds the element at position k in the sorted array and re-arranges A so that A[k] is that element.

choosePivot3(A)



CS 355 (USNA)

Unit 5

Spring 2012 20 / 39

Worst case of choosePivot3(A)

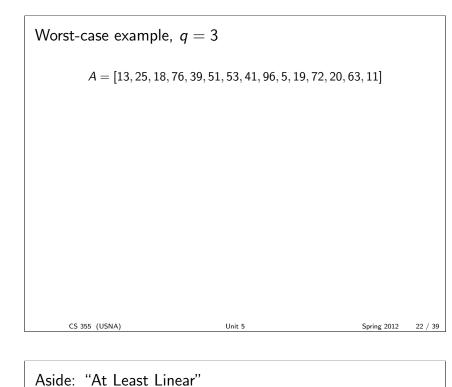
Assume all array elements are distinct.

Question: How unbalanced can the pivoting be?

- Chosen pivot *must* be greater than $\lfloor m/2 \rfloor$ medians.
- Each median must be greater than $\lfloor q/2 \rfloor$ elements.
- Since $m = \lfloor n/q \rfloor$, the pivot must be greater than (and less than) approximately



elements in the worst case.



Definition

A function f(n) is **at least linear** if and only if f(n)/n is non-decreasing (for sufficiently large n).

- Any function that is $\Theta(n^c(\log n)^d)$ with $c \ge 1$ is "at least linear".
- You can pretty much assume that any running time that is $\Omega(n)$ 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.

CS 355 (USNA)

Unit 5

Spring 2012 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)
- Let S(n) =worst-case cost of selectPivot3(A)
- T(n) =
- S(n) =
- Combining these, T(n) =

Unit 5

Choosing <i>q</i>			
• What if <i>q</i> is big? Try	q=n/3.		
What if q is small? Tr	rv q = 3.		
	, , , , , , , , , , , , , , , , , , , ,		
CS 355 (USNA)	Unit 5	Spring 2012	25 / 39
Choosing <i>q</i>			
What about $q = 3$:			
What about $q = 5$?			

QuickSort

QuickSelect is based on a sorting method developed by Hoare in 1960:

Unit 5

```
quickSort1(A)
```

CS 355 (USNA)

Input: Array *A* of size *n* **Output**: The array is sorted in-place.

```
if n > 1 then
swap (A[0], A[choosePivot1(A)])
p := partition(A)
quickSort1(A[0..p-1])
quickSort1(A[p+1..n-1])
end if
```

Spring 2012 26 / 39

QuickSort vs QuickSelect

- Again, there will be three versions depending on how the pivots are chosen.
- Crucial difference: QuickSort makes two recursive calls
- Best-case analysis:
- Worst-case analysis:
- We could ensure the best case by using quickSelect3 for the pivoting.
 In practice, this is too slow.

CS 355 (USNA)

Unit 5

Spring 2012 28 / 39

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.

CS 355 (USNA)

Unit 5

Spring 2012 29 / 39

Average depth of recursion for quickSort1

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

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

- 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 $\frac{1}{2}$.

Summary of QuickSor	t analysis		
 quickSort1: Choose A Worst-case: Θ(n²) Average case: Θ(n quickSort2: Choose t 	og <i>n</i>)		
 ▶ Worst-case: Θ(n²) ▶ Expected case: Θ($n \log n$		
	- /		
 quickSort3: Use the ▶ Worst-case: Θ(n log 		choose pivots.	
	, '')		
CS 355 (USNA)	Unit 5	Spring 2012	31 / 39
Sorting so far			
We have seen:			
Quadratic-time algorith			
BubbleSort, SelectionS n log n-time algorithms 			
HeapSort, MergeSort,			
$O(n \log n)$ is asymptoticall	v ontimal in the com	uparison model	
So how could we do better?)		
CS 355 (USNA)	Unit 5	Spring 2012	32 / 39
			. ,

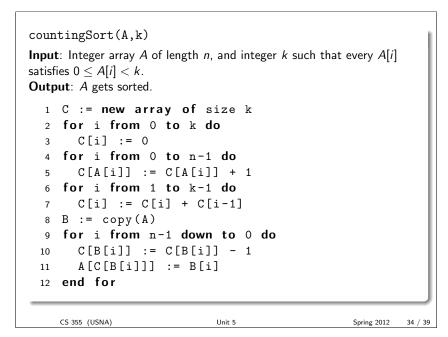
BucketSort BucketSort is a general approach, not a specific algorithm: Split the range of outputs into k groups or buckets Go through the array, put each element into its bucket Sort the elements in each bucket (perhaps recursively) Dump sorted buckets out, in order Notice: No comparisons!

Unit 5

Spring 2012

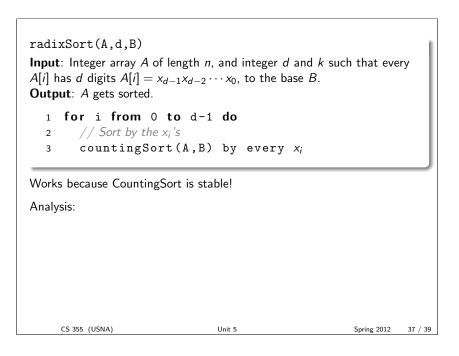
33 / 39

CS 355 (USNA)



Analysis of (• Time:			
• Space:			
CS 355 (USNA	Unit 5	Spring 2012	35 / 39

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.



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	
CS 355 (USNA)	Unit 5		Spring 2012	38 / 39

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