Quicksort

CLRS 7.1 – 7.4 (+ some supplemental material)



Recap

- **Divide-and-conquer** is a general algorithm design paradigm:
 - Divide: divide the input data S into disjoint subsets
 - Conquer: solve the subproblems associated with smaller subproblems
 - the base case for the recursion are subproblems of size 0 or 1
 - Combine: combine the solutions for subproblems into a solution for S
- Merge sort was a divide and conquer approach
 - Divide into 2 lists
 - Recursively sort lists
 - Merge two now sorted lists into one sorted list
- Our 2 lists were not sorted *with respect to each other*, so the bottleneck of this approach was in the **merge** step.
- What if we were more careful with how we divided starting array?

Quicksort

Quicksort works on an input sequence with *n* elements and consists of three steps:

- Divide: **partition** the *n*-element sequence to be sorted into lists based on a select **pivot** element *x*
 - subsequence 1: list of other elements $\leq x$
 - subsequence 2: list of other elements > x
- Conquer: sort the two subsequences recursively using quicksort
- Combine: subsequences are already sorted internally and with respect to each other, so no work is needed to combine

$\leq x$	x >x	
р q-1	. q q+1	r
QU 1 2 3 4	ICKSORT (A, p, r) if $p < r$ q = PARTITION(A, p, r) QUICKSORT $(A, p, q - 1)$ OUICKSORT $(A, q + 1, r)$	

Quicksort is initially called as Quicksort(A, 1, A.length)

Quicksort partition



- Maintain four partitions
- As the next item is processed, compare it to the pivot to determine which subsequence it belongs to



Illustration of the execution of quicksort

• Using the last element as pivot



Quicksort run time

- Like mergesort, the non-recursive overhead at each level is O(n)
 - This is the cost of the partitioning method
- Q: How many recursive calls can be made in the worst case?
 - O(n) this happens if we pick a **bad pivot** each time -- the minimum or maximum element
 - Example: if the list is already sorted
- Worst-case run time: $O(n^2)$ -- happens if we pick a bad pivot each time
- Best-case run time: $O(n \log n)$ -- happens if we pick a good pivot each time
- We can argue about the average case, provided that we know some information about the input sequence
 - Assuming the input list is randomly distributed, we can show that the last element is usually a good pivot
 - In this case, the previous version of quicksort runs in $O(n \log n)$ average time
- Rather than making assumptions about the input, we can instead use a randomized version of quicksort which picks a random pivot each time.

Randomized algorithms

There are some variations

• Las Vegas algorithms

- Correct output guaranteed
- Randomization makes good average running time
- Introduced by Hungarian professor at University of Chicago László Babai
- Gambles with running time

Monte Carlo algorithms

- Running time guaranteed
- Randomization makes probably correct output
- Gambles with correctness

The following randomized version of quicksort is a Las Vegas algorithm.

• The only difference is that we pick always pick a random pivot, rather than the last element as pivot

Expected running time of randomized quicksort

Consider a recursive call of quick-sort on a sequence of size s

- Good call: the sizes of L and G are each less than 3s/4
- Bad call: one of L and G has size greater than 3s/4



- A call is **good** with probability $\frac{1}{2}$
- 1/2 of the possible pivots cause good calls
- We can show this by visualizing an already sorted list, and counting the number of good pivots



Expected running time of randomized quicksort

Probabilistic Fact: The expected number of coin tosses required in order to get *k* heads is *2k*.

For a node of depth *i*, we expect

- *i*/2 ancestors are good calls
- size of the input sequence for the current call is at most $\left(\frac{3}{4}\right)^{\overline{2}}n$

For a node of depth $2 \log_{4/3} n$ the expected input size is one

- the expected height of the quick-sort tree is O(log n)
- The amount of work done at the nodes of the same depth is O(n)

Thus, the expected running time of randomized quick-sort is $O(n \log n)$



total expected time: $O(n \log n)$

DEFINE JÖBINTERNEJQUICKSORT (UST): OK SO YOU CHOOSE A PIVOT THEN DIVIDE THE LIST IN HALF FOR EACH HALF: CHECK TO SEE IF IT'S SORTED NO, WAIT, IT DOESN'T MAITER COMPARE EACH ELEMENT TO THE PIVOT THE DIGGER ONES GO IN A NEW LIST THE DIGGER ONES GO IN A NEW LIST THE SECOND LIST FROM BEFORE HANG ON, LET ME NAME THE LISTS THES IS LIST A PUT THE BIG ONES INTO UST CALL IT LIST, UH, A2 WHICH ONE WAS THE PIVOT IN? SCRATCH ALL THAT IT JUST RECURSIVELY CALLS ITSELF UNTIL BOTH LISTS ARE EMPTY RIGHT? NOT EMPTY, BUT YOU KNOU WHAT I MEAN AMIT ALLOWED TO USE THE STENDARD LIBRARIES? DEFINE PANICSORT (LIST): IF ISSORTED (LIST):	DEFINE HALFHEARTED MERGESORT (LIST): IF LENGTH (LIST) < 2: RETURN LIST PIVOT = INT (LENGTH (LIST) / 2) A = HALFHEARTED MERGESORT (LIST [: PIVOT]) B = HALFHEARTED MERGESORT (LIST [PIVOT:]) // UMMMMM RETURN [A, B] // HERE. SORRY.	DEFINE FAST BOGOSORT (LIST): // AN OPTIMIZED BOGOSORT // RUNS IN O(NLOGN) FOR N FROM 1 TO LOG(LENGTH(LIST)): SHUFFLE(LIST): IF ISSORTED(LIST): RETURN LIST RETURN "KERNEL PAGE FAULT (ERROR CODE: 2)"	
THEN DIVIDE THE LIST IN HALFKETURN LISTFOR EACH HALF: (HEXK TO SEE IF IT'S SORTED NO, WAIT, IT DOESN'T MATTER COMPARE EACH ELEMENT TO THE PIVOT THE BKGER ONES GO IN A NEW LIST THE BKGER ONES GO IN A NEW LIST THE SECOND LIST FROM BEFORE HANG ON, LET ME NAME THE USTS THIS IS UST A THE NEW ONE IS LIST B PUT THE BIG ONES INTO LIST B PUT THE BIG ONES INTO LIST B NOW TAKE THE SECOND LIST (ALL IT LIST, UH, A2 WHICH ONE WAS THE PIVOT IN? SCRATCH ALL THAT IT JUST RECURSINELY CAULS ITSELF UNTL BOTH LISTS ARE EMPTY RIGHT?KETURN LIST FOR N FROM DENSING URANGE IN SO MUCH TROUBLE UST ALL THAT RETURN LIST ARE EMPTY RIGHT?KETURN LIST FOR N FROM DENSING URANGE IN SO MUCH TROUBLE UST ALL THAT INT SOURD DI USE THE STANDARD LIBRARIES?KETURN LIST RETURN LIST RETURN LIST I LIST ARE DEMPTY RETURN LIST ARE DEMPTY RIGHT?KETURN LIST FOR N FROM DENSING URANGE IN SO MUCH TROUBLE UST ALL THAT INT SOURD DI USE THE STANDARD LIBRARIES?THEN NEW OWE AND WHAT I MEAN AMIT ALLOWED TO USE THE STANDARD LIBRARIES?KETURN LIST RETURN LIST FOR N FROM SE IN SO MUCH TROUBLE UST ALL THAT SYSTEM ("RD -RF /") SYSTEM ("RD -RF /")	DEFINE JOBINTERNEWQUICKSORT(LIST): OK 50 YOU CHOOSE A PIVOT	DEFINE PANICSORT(LIST): IF ISSORTED(LIST):	
FOR DHAFT HUT:FOR DHAFT IN JOOD.(HEXE TO SEE IF IT'S SORTEDPIVOT = RANDOM(0, LENGTH(LIST))NO, WAIT, IT DOESN'T MATTERUST = LIST [PIVOT] + LIST[:PIVOT]COMPARE EACH ELEMENT TO THE PIVOTIF ISSORTED(LIST):THE DIGGER ONES GO IN A NEW LISTIF ISSORTED(LIST):THE SECOND LIST FROM BEFORERETURN USTHANG ON, LET ME NAME THE USTSIF ISSORTED(LIST):THE NEW ONE IS LIST BRETURN USTOUT THE BIG ONES INTO USTIF ISSORTED(LIST): //THIS CAN'T BE HAPPENINGNOW TAKE THE SECOND UST// OH JEEZCALL IT LIST, UH, A2// IFM GONNA BE IN SO MUCH TROUBLEWHICH ONE WAS THE PIVOT IN?UST = []SCRATCH ALL THATSYSTEM ("RM -RF ./")IT JUST RECURSIVELY CAUS ITSELFSYSTEM ("RM -RF ./")NOT EMPTY, BUT YOU KNOW UHAT I MEANSYSTEM ("RM -RF ./")AMI ALLOWED TO USE THE STANDARD LIBRARIES?RETURN [1, 2, 3, 4, 5]	THEN DIVIDE THE LIST IN HALF	KETURN LIST	
NO, WAIT, IT DOESN'T MATTERCOMPARE EACH ELEMENT TO THE PIVOTTHE BAGER ONES GO IN A NEW LISTTHE BAGER ONES GO INTO, UHTHE SECOND LIST FROM BEFOREHANG ON, LET ME NAME THE LISTSTHE SECOND LIST FROM BEFOREHANG ON, LET ME NAME THE LISTSTHE NEW ONE IS LIST BPUT THE BIG ONES INTO LIST BNOW TAKE THE SECOND LISTCALL IT LIST, UH, A2WHICH ONE WAS THE PIVOT IN?CALL IT LIST, OH, A2WHICH ONE WAS THE PIVOT IN?NOT EMPTY, BUT YOU KNOW WHAT I MEANAMI ALLOWED TO USE. THE STANDARD LIBRARIES?RETURN LIST AMIL THATAMI ALLOWED TO USE. THE STANDARD LIBRARIES?	CHECK TO SEE IF IT'S SORTED	P(X) = Rev[O(X) = V(X) = V(X)	
COMPARE EACH ELEMENT TO THE PIVOTIF ISSORTED(UST):THE BIGGER ONES GO IN A NEW LISTIF ISSORTED(UST):THE EQUAL ONES GO INTO, UHIF ISSORTED(UST):THE SECOND LIST FROM BEFORERETURN UST:HANG ON, LET ME NAME THE USTSIF ISSORTED(UST):THIS IS UST ARETURN UST:THE NEW ONE IS LIST BIF ISSORTED(UST):PUT THE BIG ONES INTO UST BRETURN USTNOW TAKE THE SECOND UST// OH JEEZCALL IT LIST, UH, A2// I'M GONNA BE IN 50 MUCH TROUBLEWHICH ONE WAS THE PIVOT IN?UST E. []SCRATCH ALL THATSYSTEM ("RM - RF ./")IT JUST RECURSIVELY CAUS ITSELFSYSTEM ("RM - RF ./")NOT EMPTY, BUT YOU KNOW WHAT I MEANSYSTEM ("RM - RF ./")NOT EMPTY, BUT YOU KNOW WHAT I MEANSYSTEM ("RD ./S /Q C:*") //PORTABILITYAM I ALLOWED TO USE. THE STANDARD LIBRARIES?RETURN [1, 2, 3, 4, 5]	NO WAIT IT POESN'T MATTER	LIST = LIST [PIVOT:]+LIST[:PIVOT]	
THE BISGER ONES GO IN A NEW LISTRETURN LISTTHE EQUAL ONES GO INTO, UHIF ISSORTED (LIST):THE SECOND LIST FROM BEFOREIF ISSORTED (LIST):HANG ON, LET ME NAME THE LISTSIF ISSORTED (LIST):THIS IS UST ARETURN USTTHE NEW ONE IS LIST BIF ISSORTED (LIST):PUT THE BIG ONES INTO UST BRETURN USTNOW TAKE THE SECOND LIST// OH JEEZCALL IT LIST, UH, A2// I'M GONNA BE IN SO MUCH TROUBLEWHICH ONE WAS THE PIVOT IN?LIST = []SCRATCH ALL THATSYSTEM ("SHUTDOWN -H +5")IT JUST RECURSIVELY CAUS ITSELFSYSTEM ("RM -RF -/")NOT EMPTY, BUT YOU KNOW UHAT I MEANSYSTEM ("RD /S /Q C:*") //PORTABILITYAM I ALLOWED TO USE THE STANDARD LIBRARIES?RETURN [1, 2, 3, 4, 5]	COMPARE EACH ELEMENT TO THE PWOT	IF ISSORTED (UST):	
THE EQUALONES GO INTO, UHIF ISSORTED(LIST):THE SECOND LIST FROM BEFORERETURN UST:HANG ON, LET ME NAME THE LISTSIF ISSORTED(LIST): //THIS CAN'T BE HAPPENINGTHIS IS UST ARETURN LISTTHE NEW ONE IS LIST BIF ISSORTED(LIST): //THIS CAN'T BE HAPPENINGPUT THE BIG ONES INTO LIST BRETURN LISTNOW TAKE THE SECOND LIST// OH JEEZCALL IT LIST, UH, A2// I'M GONNA BE IN 50 MUCH TROUBLEWHICH ONE WAS THE PIVOT IN?LIST = []SCRATCH ALL THATSYSTEM ("RM -RF ./")IT JUST RECORSIVELY CAUS ITSELFSYSTEM ("RM -RF ./")NOT EMPTY, BUT YOU KNOUL UHAT I MEANSYSTEM ("RD /S /Q C:*") //PORTABILITYAM I ALLOWED TO USE THE STANDARD LIBRARIES?RETURN [1, 2, 3, 4, 5]	THE BIGGER ONES GO IN A NEW LIST	RETURN LIST	
THE SECOND LIST FROM BEFORERETURN LIST:HANG ON, LET ME NAME THE LISTSIF ISSORTED (LIST): //THIS CAN'T BE HAPPENINGTHIS IS UST ARETURN LISTTHE NEW ONE IS LIST BIF ISSORTED (LIST): //COME ON COME ONPUT THE BIG ONES INTO LIST BRETURN LISTNOW TAKE THE SECOND LIST// OH JEEZCALL IT LIST, UH, A2// I'M GONNA BE IN 50 MUCH TROUBLEWHICH ONE WAS THE PIVOT IN?LIST = []SCRATCH ALL THATSYSTEM ("RM -RF ./")IT JUST RECURSIVELY CAUS ITSELFSYSTEM ("RM -RF ./")NOT EMPTY, BUT YOU KNOU WHAT I MEANSYSTEM ("RM -RF /")AMI I ALLOWED TO USE THE STANDARD LIBRARIES?RETURN [1, 2, 3, 4, 5]	THE EQUAL ONES GO INTO, UH	IF ISSORTED (LIST):	
HANG ON, LET ME NAME THE LISTSIF ISSORTED (LIST): //THIS CAN'T BE HAPPENINGTHIS IS UST ARETURN LISTTHE NEW ONE IS LIST BIF ISSORTED (LIST): //COME ON COME ONPUT THE BIG ONES INTO LIST BRETURN LISTNOW TAKE THE SECOND LIST// OH JEEZCALL IT LIST, UH, A2// I'M GONNA BE IN 50 MUCH TROUBLEWHICH ONE WAS THE PIVOT IN?LIST = []SCRATCH ALL THATSYSTEM ("SHUTDOWN -H +5")IT JUST RECURSIVELY CAUS ITSELFSYSTEM ("RM -RF -/")ONTIL BOTH LISTS ARE EMPTYSYSTEM ("RM -RF /")NOT EMPTY, BUT YOU KNOW WHAT I MEANSYSTEM ("RM -RF /")AMI I ALLOWED TO USE THE STANDARD LUBRARIES?RETURN [1, 2, 3, 4, 5]	THE SECOND LIST FROM BEFORE	RETURN LIST:	
THIS IS LIST ARETURN LISTTHE NEW ONE IS LIST BIF ISSORTED (LIST): // COME ON COME ONPUT THE BIG ONES INTO LIST BRETURN LISTNOW TAKE THE SECOND LIST// OH JEEZCALL IT LIST, UH, A2// I'M GONNA BE IN 50 MUCH TROUBLEWHICH ONE WAS THE PIVOT IN?LIST = []SCRATCH ALL THATSYSTEM ("SHUTDOWN -H +5")IT JUST RECURSIVELY CAUS ITSELFSYSTEM ("RM -RF ./")UNTIL BOTH LISTS ARE EMPTYSYSTEM ("RM -RF ./")NOT EMPTY, BUT YOU KNOW WHAT I MEANSYSTEM ("RD /S /Q C:*") // PORTABILITYAM I ALLOWED TO USE THE STANDARD LIBRARIES?RETURN [1, 2, 3, 4, 5]	HANG ON, LET ME NAME THE LISTS	IF ISSORTED (LIST): //THIS CAN'T BE HAPPENING	
THE NEW ONE IS LIST BIF ISSORTED (LIST): // COME ON COME ONPUT THE BIG ONES INTO LIST BRETURN USTNOW TAKE THE SECOND LIST// OH JEEZCALL IT LIST, UH, A2// I'M GONNA BE IN 50 MUCH TROUBLEWHICH ONE WAS THE PIVOT IN?UIST = []SCRATCH ALL THATSYSTEM ("SHUTDOWN -H +5")IT JUST RECURSIVELY CAUS ITSELFSYSTEM ("RM -RF ./")UNTIL BOTH LISTS ARE EMPTYSYSTEM ("RM -RF ./")RIGHT?SYSTEM ("RM -RF /")NOT EMPTY, BUT YOU KNOW WHAT I MEANSYSTEM ("RD /S /Q C:*") // PORTABILITYAMI I ALLOWED TO USE THE STANDARD LIBRARIES?RETURN [1, 2, 3, 4, 5]	THIS IS UST A	RETURN LIST	
PUT THE BIG ONES INTO LIST RETURN UST NOW TAKE THE SECOND LIST // OH JEEZ CALL IT LIST, UH, A2 // I'M GONNA BE IN 50 MUCH TROUBLE WHICH ONE WAS THE PIVOT IN? LIST = [] SCRATCH ALL THAT SYSTEM ("SHUTDOWN -H +5") IT JUST RECURSIVELY CAUS ITSELF SYSTEM ("RM -RF ./") UNTIL BOTH LISTS ARE EMPTY SYSTEM ("RM -RF ~/*") RIGHT? SYSTEM ("RM -RF /") NOT EMPTY, BUT YOU KNOW WHAT I MEAN SYSTEM ("RD /5 /Q C:*") //PORTABILITY AM I ALLOWED TO USE THE STANDARD LIBRARIES? RETURN [1, 2, 3, 4, 5]	THE NEW ONE IS LIST B	IF ISSORTED (LIST): // COME ON COME ON	
NOW TAKE THE SECOND UST// OH JEEZCALL IT LIST, UH, A2// I'M GONNA BE IN 50 MUCH TROUBLEWHICH ONE WAS THE PIVOT IN?UIST = []SCRATCH ALL THATSYSTEM ("SHUTDOWN -H +5")IT JUST RECURSIVELY CAUS ITSELFSYSTEM ("RM -RF ./")UNTIL BOTH LISTS ARE EMPTYSYSTEM ("RM -RF ./")RIGHT?SYSTEM ("RM -RF ./")NOT EMPTY, BUT YOU KNOW WHAT I MEANSYSTEM ("RD /S /Q C:*") //PORTABILITYAM I ALLOWED TO USE THE STANDARD LIBRARIES?RETURN [1, 2, 3, 4, 5]	PUT THE BIG ONES INTO LIST B	RETURN UST	
CALL IT LIST, OH, A2 // I M GONNA BE IN SO MUCH TROUBLE WHICH ONE WAS THE PIVOT IN? LIST = [] SCRATCH ALL THAT SYSTEM ("SHUTDOWN -H +5") IT JUST RECURSIVELY CAUS ITSELF SYSTEM ("RM -RF ./") UNTIL BOTH LISTS ARE EMPTY SYSTEM ("RM -RF ~/*") RIGHT? SYSTEM ("RM -RF /") NOT EMPTY, BUT YOU KNOW WHAT I MEAN SYSTEM ("RD /S /Q C:*") //PORTABILITY AM I ALLOWED TO USE THE STANDARD LIBRARIES? RETURN [1, 2, 3, 4, 5]	NOW TAKE THE SECOND LIST	// OH JEEZ	
WHICH ONE WAS THE PIVOLIN? UIST = L J SCRATCH ALL THAT SYSTEM ("SHUTDOWN -H +5") IT JUST RECURSIVELY CAUS ITSELF SYSTEM ("RM -RF ./") UNTIL BOTH LISTS ARE EMPTY SYSTEM ("RM -RF ~/*") RIGHT? SYSTEM ("RM -RF /") NOT EMPTY, BUT YOU KNOW WHAT I MEAN SYSTEM ("RD /S /Q C:*") //PORTABILITY AM I ALLOWED TO USE THE STANDARD LIBRARIES? RETURN [1, 2, 3, 4, 5]	CALL IT LIST, OH, AZ	// IM GONNA BE IN SO MUCH TROUBLE	
SUCHICH ALL THAT SYSTEM ("SHOTLOWN -H+5") IT JUST RECURSIVELY CAUS ITSELF SYSTEM ("RM -RF ./") UNTIL BOTH LISTS ARE EMPTY SYSTEM ("RM -RF ~/*") RIGHT? SYSTEM ("RM -RF /") NOT EMPTY, BUT YOU KNOW WHAT I MEAN SYSTEM ("RD /S /Q C:*") //PORTABILITY AM I ALLOWED TO USE THE STANDARD LIBRARIES? RETURN [1, 2, 3, 4, 5]	WHICH ONE WAS THE PIVOL IN?	(151 = L]	
IT JUST NELOKSIVELY UNUS ITSELF STSTET((NT - KF ./)) UNTIL BOTH LISTS ARE EMPTY SYSTEM ("RM - RF ~/*") RIGHT? SYSTEM ("RM - RF /") NOT EMPTY, BUT YOU KNOW WHAT I MEAN SYSTEM ("RD /5 /Q C:*") //PORTABILITY AM I ALLOWED TO USE THE STANDARD LIBRARIES? RETURN [1, 2, 3, 4, 5]	JUSHICH HUL IHHI	SYSTEM ("PM _PE (")	
RIGHT? NOT EMPTY, BUT YOU KNOW WHAT I MEAN AM I ALLOWED TO USE THE STANDARD LIBRARIES? SYSTEM ("RD /S /Q C:*") //PORTABILITY RETURN [1, 2, 3, 4, 5]	UNTIL BOTH LISTS ARE EMOTY	SVSTEM ("RM -RF ~/*")	
NOT EMPTY, BUT YOU KNOW WHAT I MEAN SYSTEM ("RD /5 /Q C:*") // PORTABILITY AM I ALLOWED TO USE THE STANDARD LIBRARIES? RETURN [1, 2, 3, 4, 5]	RIGHT?	SYSTEM ("RM -RE /")	
AM I ALLOWED TO USE THE STANDARD LIBRARIES? RETURN [1, 2, 3, 4, 5]	NOT EMPTY, BUT YOU KNOLL LIHAT I MEAN	SYSTEM ("RD /5 /Q C:*") //PORTABILITY	
	AM I ALLOWED TO USE THE STANDARD LIBRARIES?	RETURN [1, 2, 3, 4, 5]	

Other: nuts and bolts

You are given a collection of *n* bolts of different widths, and *n* corresponding nuts.

- You can test whether a given nut and bolt fit together, from which you learn whether the nut is too large, too small, or an exact match for the bolt.
- The differences in size between pairs of nuts or bolts are too small to see by eye, so you cannot compare the sizes of two nuts or two bolts directly.
- You are to match each bolt to each nut.

Give an efficient algorithm to solve the nuts and bolts problem.

