CSE 2320 Notes 6: Sorting

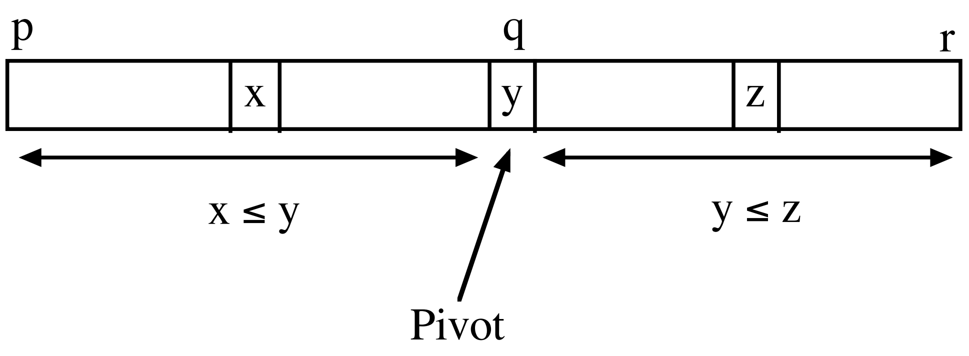
(Last updated 9/16/06 3:22 PM)

CLRS, 7.1-7.2, 8.1-8.3

QUICKSORT

Concepts

Idea: Take an unsorted (sub)array and *partition* into two subarrays such that



Customarily, the last subarray element (subscript r) is used as the pivot value.

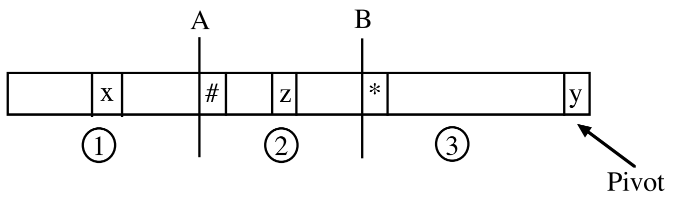
After partitioning, each of the two subarrays, p . . . q – 1 and q + 1 . . . r, are sorted recursively.

Subscript q is returned from PARTITION.

Like MERGESORT, QUICKSORT is a divide-and-conquer technique:

	MERGESORT	QUICKSORT
Divide	Trivial	PARTITION
Subproblems	Sort Two Parts	Sort Two Parts
Combine	MERGE	Trivial

Fast PARTITION (in $\Theta(n)$ time, see `partition.c`)



① Already known to have x ≤ y

② Already known to have y < z

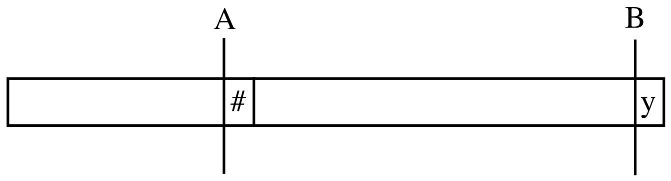
③ Untouched

y < *: Move B over

* ≤ y: Swap # & *
Move B over
Move A over

A and B can be the same . . .

Termination



Swap # & y to place y in its final position.

Example:

AB $\underline{6}$ 3 7 2 8 4 9 0 1 5

A $\underline{6}$ B $\underline{3}$ 7 2 8 4 9 0 1 5

3 A 6 B $\underline{7}$ 2 8 4 9 0 1 5

3 A $\underline{6}$ 7 B $\underline{2}$ 8 4 9 0 1 5

3 2 A 7 6 B $\underline{8}$ 4 9 0 1 5

3 2 A $\underline{7}$ 6 8 B $\underline{4}$ 9 0 1 5

3 2 4 A 6 8 7 B $\underline{9}$ 0 1 5

3 2 4 A $\underline{6}$ 8 7 9 B $\underline{0}$ 1 5

3 2 4 0 A $\underline{8}$ 7 9 6 B $\underline{1}$ 5

3 2 4 0 1 A $\underline{7}$ 9 6 8 B $\underline{5}$

3 2 4 0 1 < 5 > 9 6 8 7

Analysis

Worst Case – Pivot is smallest or largest key in subarray *every time*. (Includes ascending or descending order.) Let $T(n)$ be the number of comparisons.

$$T(n) = T(n-1) + n - 1 = \sum_{i=1}^{n-1} i = \Theta\left(n^2\right)$$

Best Case – Pivot always ends up in the middle. $T(n) = 2T\left(\frac{n}{2}\right) + n - 1$ (Similar to mergesort.)

Expected Case – Assume all $n!$ permutations are equally likely to occur. Likewise, each element is equally likely to occur as the pivot (each of the n elements will be the pivot in $(n-1)!$ cases). $E(n)$ is the expected number of comparisons. $E(0) = 0$.

$$E(n) = n - 1 + \sum_{i=0}^{n-1} \frac{1}{n}\left(E(i) + E(n-1-i)\right) = n - 1 + \frac{2}{n}\sum_{i=1}^{n-1} E(i)$$

Show $O(n \log n)$. Suppose $E(i) \le ci \ln i$ for $i < n$.

$$E(n) \le n - 1 + \frac{2c}{n}\sum_{i=1}^{n-1} i \ln i \le n - 1 + \frac{2c}{n}\int_1^n x \ln x \, dx \quad \text{[Bound by integral, CLRS, p. 1068]}$$

$$= n - 1 + \frac{2c}{n}\left[\frac{1}{2}x^2 \ln x - \frac{x^2}{4}\right]_1^n \quad \text{[From http://integrals.wolfram.com]}$$

$$= n - 1 + \frac{2c}{n}\left(\frac{n^2}{2}\ln n - \frac{n^2}{4} + \frac{1}{4}\right) = n - 1 + cn \ln n - \frac{cn}{2} + \frac{c}{2n}$$

$$\le cn \ln n \text{ for } c \ge 2$$

Selection and Ranking Using Quicksort Partitioning Ideas

Finding *k*<u>th</u> largest (or smallest) element in unordered table of *n* elements (`selection.c`)

Sort everything?

Use PARTITION several times. Always throw away the subarray that cannot include the target.

$\Theta\left(n^2\right)$ worst case (e.g. input ordered)

$\Theta(n)$ expected. Let $E(k,n)$ represent the expected number of comparisons to find the *k*th largest in a set of *n* numbers. (Assume all $n!$ permutations are equally likely.)

Suppose $n = 7$ and $k = 3$. After 6 comparisons to place a pivot, the 7 possible pivot positions require different numbers of additional comparisons:

1	$E(3,6)$
2	$E(3,5)$
3	$E(3,4)$
4	$E(3,3)$
5	0
6	$E(1,5)$
7	$E(2,6)$

Suppose $n = 8$ and $k = 6$. After 7 comparisons to place a pivot, the 8 possible pivot positions require different numbers of additional comparisons:

1	$E(6,7)$
2	$E(6,6)$
3	0
4	$E(1,3)$
5	$E(2,4)$
6	$E(3,5)$
7	$E(4,6)$
8	$E(5,7)$

Observation: Finding the median is slightly more difficult than all other cases.

$$E(k,n) = n-1+\frac{1}{n}\sum_{i=1}^{k-1}E(i,n-k+i)+\frac{1}{n}\sum_{i=k}^{n-1}E(k,i)$$

Show O(n). Suppose $E(i,j) \le cj$ for $j < n$.

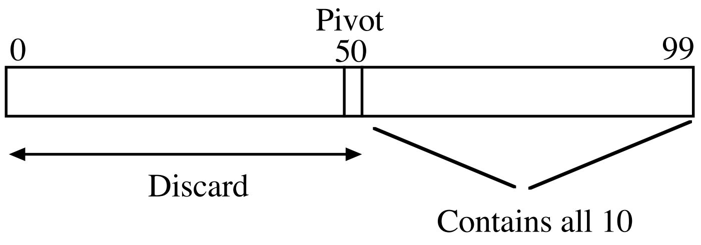
$$\begin{aligned}
E(k,n) &\le n-1+\frac{c}{n}\sum_{i=1}^{k-1}(n-k+i)+\frac{c}{n}\sum_{i=k}^{n-1}i \\
&= n-1+\frac{c}{n}\sum_{i=1}^{k-1}(n-k)+\frac{c}{n}\sum_{i=1}^{k-1}i+\frac{c}{n}\sum_{i=k}^{n-1}i \\
&= n-1+\frac{c}{n}(k-1)(n-k)+\frac{c}{n}\sum_{i=1}^{n-1}i = n-1+\frac{c}{n}(k-1)(n-k)+\frac{c}{n}\frac{(n-1)n}{2} \\
&\le n-1+\frac{c}{n}\left(\frac{n^2}{4}-\frac{n}{2}+\frac{1}{4}\right)+\frac{c}{2}(n-1) \qquad k=\frac{n+1}{2} \text{ maximizes } \frac{c}{n}(k-1)(n-k) \\
&= n-1+\frac{cn}{4}-\frac{c}{2}+\frac{c}{4n}+\frac{cn}{2}-\frac{c}{2} = n-1+\frac{3cn}{4}-c+\frac{c}{4n} = cn-\frac{cn}{4}+n-1+\frac{c}{4n}-c \\
&\le cn \text{ for } c \ge 4
\end{aligned}$$

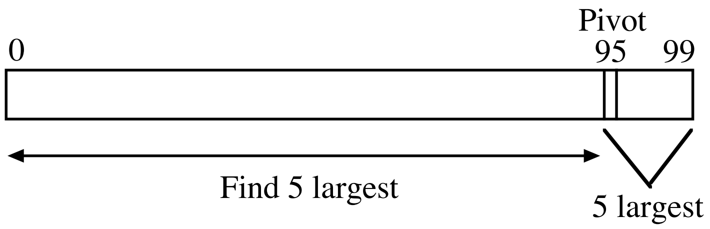
Find *k* largest (or smallest) element<u>*s*</u> (`klargest.c`)

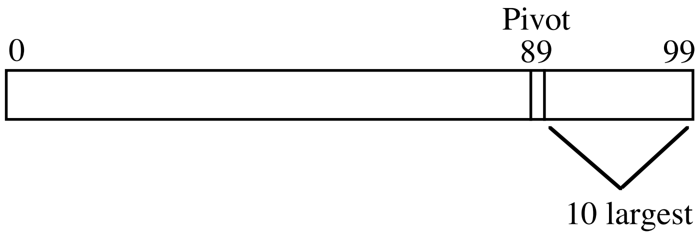
Use partial HEAPSORT ($\Theta(n + k\log n)$ worst case)

Use PARTITION ($\Theta(n)$ expected) – consider the returned pivot subscript:

Suppose *k*=10:





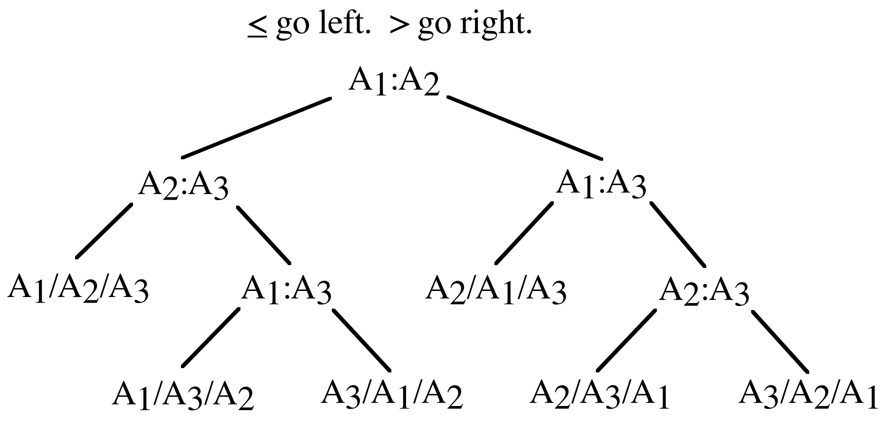


```
low=0;
high=n-1;
while (1)
{
  q=partition(arr,low,high);
  if (q<n-k-1)
    low=q+1;
  else if (q>n-k)
    high=q-1;
  else
    break;
}
```

LOWER BOUNDS ON SORTING

Since a lower bound on a problem is to apply to a number of algorithms, it is necessary to have a *model* that captures the essential features of those algorithms. It is possible for algorithms to exist that do not follow the model.

Example Decision Tree



Decision Tree Model for Sorting

Two keys may be compared in $\Theta(1)$ time.

The time for other processing is proportional to the number of comparisons.

All $n!$ possible input permutations must be successfully sorted. (Leaves are labeled to show how input array has been rearranged.)

A tree with outcomes as leaves and decisions as internal nodes may be constructed for an algorithm and a specific value of n.

Worst-case comparisons?

Expected comparisons?

What is the minimum height of a decision tree for sorting n keys?

Since there must be $n!$ leaves, then the height is $\Omega(\lg(n!)) = \Omega(n \lg n)$.

Which of the following will not be true regarding the decision tree for MERGE-SORT for sorting n input values?

A. Every path from the root to a leaf will have $O(n \log n)$ decisions.
B. The height of the tree is $\Omega(n \log n)$.
C. There will be a path from the root to a leaf with $\Omega\left(n^2\right)$ decisions.
D. There will be $n!$ leaves.

Which of the following will not be true regarding the decision tree for QUICKSORT for sorting n input values?

A. Every path from the root to a leaf will have $O(n \log n)$ decisions.
B. The height of the tree is $\Omega(n \log n)$.
C. There will be a path from the root to a leaf with $\Omega\left(n^2\right)$ decisions.
D. There will be $n!$ leaves.

Other Examples of Decision Trees

Binary search on ordered table – n leaves for the outcomes. $\Omega(\lg n)$ lower bound.

Searching unordered table? Still get $\Omega(\lg n)$ as a weak lower bound. (Use *adversary* instead.)

Problem 8-6: Give decision tree model for merging two ordered tables with n elements each.

1. Number of outcomes is based on:

   a. $2n$ elements in output table.
   b. n elements of output table will receive n elements of first table *in their original order*.

2. Number of leaves = number of outcomes:

$$\binom{2n}{n} = \frac{(2n)!}{n!n!} = \frac{(2 \bullet 4 \bullet 6 \bullet \cdots \bullet 2n)(1 \bullet 3 \bullet 5 \bullet \cdots \bullet 2n-1)}{n!n!}$$

$$= \frac{2^n(1 \bullet 2 \bullet 3 \bullet \cdots \bullet n)(1 \bullet 3 \bullet 5 \bullet \cdots \bullet 2n-1)}{n!n!}$$

$$= \frac{2^n}{n} \prod_{i=1}^{n-1} \frac{2i+1}{i} \geq \frac{2^n}{n} \prod_{i=1}^{n-1} \frac{2i}{i} = \frac{2^{2n-1}}{n}$$

3. Height of tree is bounded below by lg(number of leaves) = $\lg\binom{2n}{n} \geq \lg \frac{2^{2n-1}}{n} = 2n - 1 - \lg n$.

STABLE SORTING

A sort is *stable* if two elements with equal keys maintain their original (input) order in the output.

Practical significance is for situations with a *compound key*:

1. Each time a user logs into a computer, a record is created with user name, date, and time.

2. Once a year, each user receives a chronological report listing their log-ins.

3. If a stable sort is available, then the sort for (2) can use just the user name as the sort key.

Which sorts can be coded "naturally" to maintain stability?

Insertion   Merge   Heap   Quick

How can an unstable sort be forced to behave like a stable sort?

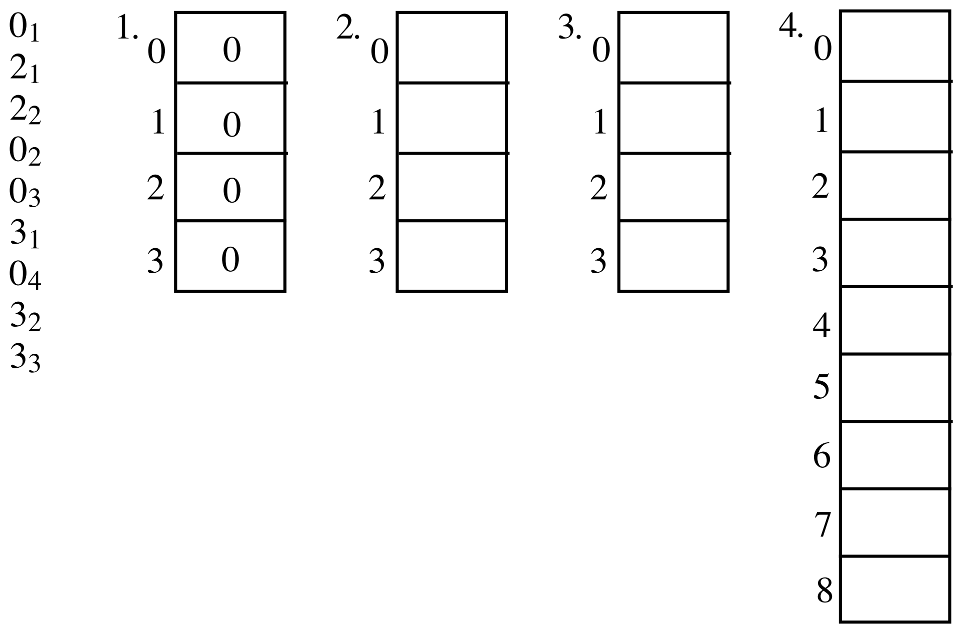
Linear Time Sorting

If the range of keys is limited, then sorting by direct key comparisons might not be the fastest method.

Counting Sort – Sort n records with keys in range 0 . . . k – 1.

1. Clear count table – one counter for each value in range. $\Theta(k)$
2. Pass through input table – add to appropriate counter for each key. $\Theta(n)$
3. Determine first slot that will receive a record for each range value. $\Theta(k)$
4. Copy each record to output, increment index in table from (3). $\Theta(n)$

Overall, takes time $\Theta(k+n)$ which will be $\Theta(n)$ if k is bounded.



(LSD) Radix Sort

Example: Sorting records whose keys are 9-digit Social Security Numbers.

1. Treat each SSN as three digit number ABC where each digit is in the range 000 . . . 999.
2. Use counting sort to sort all records on C.
3. Use counting sort to sort all records on B. (Must be done in stable fashion.)
4. Use counting sort to sort all records on A. (Must be done in stable fashion.)

Time is $\Theta(d(k+n))$ where d is the number of digits (3), k is the size of the radix (1000), and n is the number of records.

Inconvenient to compare to key-comparison based sorts.

If the radix is binary, code similar to PARTITION may be used instead of counting sort.