Advanced Sorting

15-121
Introduction to Data Structures

Ananda Gunawardena
July 29, 2005

Objectives

- Understanding quadratic and sub quadratic sorting algorithms
- Learning the algorithmic details about sub quadratic algorithms
- Understanding informal complexity arguments
- Worst case scenarios
- Summary of Sorting Algorithms

So far we learned…

- Sorting a data set can lead to easy algorithms for finding
  - Median, mode etc..
- $O(n^2)$ sorting algorithms are easy to code, but inefficient for large data sets. However insertion sort is used widely in practice for small data sets
- Question : Why are bubble sort and selection sorts do not perform as well as insertion sorts?

- Next we consider
  - O(n log n) sorting algorithms

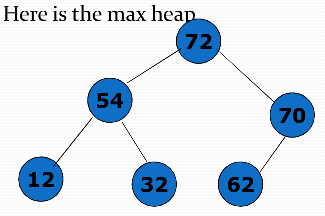
Heap Sort

Heap Sort

- Suppose we have an unsorted array of comparable objects that need to be sorted. Assume that entries are in 1..n
- First we will build a max heap by percolating down - this time we need to use the larger of the two children
- Then we deleteMax. We will place the max at the end of the array, reduce the size of the heap by 1 and percolate down to maintain maxHeap property

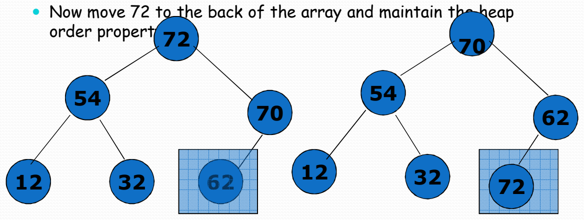
Heap Sort Demo

- Start with 54 32 62 12 72 70
- Here is the max heap



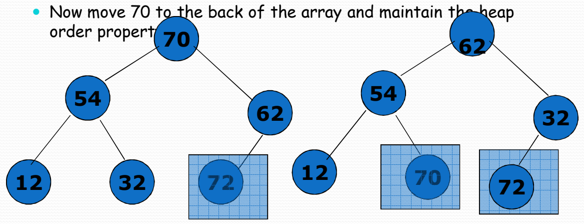
Heap Sort Demo ctd..

- Now move 72 to the back of the array and maintain the heap order property



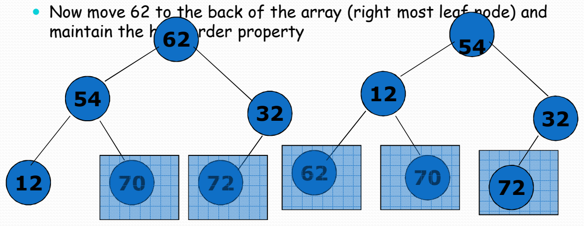
Heap Sort Demo ctd..

- Now move 70 to the back of the array and maintain the heap order property



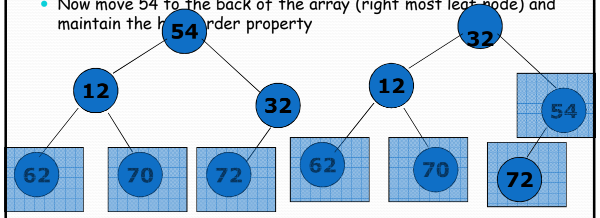
Heap Sort Demo ctd..

- Now move 62 to the back of the array (right most leaf node) and maintain the heap order property



Heap Sort Demo ctd..

- Now move 54 to the back of the array (right most leaf node) and maintain the heap order property



Heap Sort Exercise

- Sort 12 10 15 8 32 20 14 in descending order

Heap Sort Code

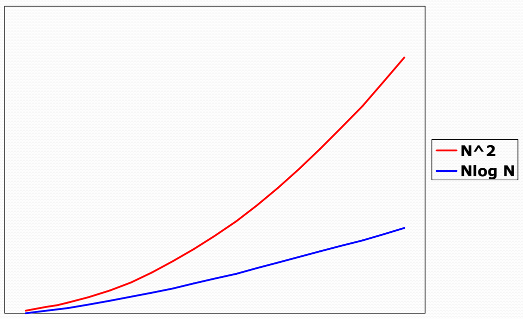
- Pseudo code

```
Build a max/min heap
for i from n-1 down to 1
    swap(A[i], A[1])
    Percol ate A[1] down
```

Heapsort Analysis

- Recall from last time about *heaps*:
  - buildHeap has $O(N)$ worst-case running time.
  - removeMax has $O(\log N)$ worst-case running time.
- Heapsort:
  - Build heap. $O(N)$
  - DeleteMax until empty. $O(N\log N)$
  - *Total worst case*: $O(N\log N)$

N^2 vs Nlog N



Sorting in $O(N\log N)$

- Heapsort establishes the fact that sorting can be accomplished in $O(N\log N)$ worst-case running time.
- The average-case analysis for heapsort is somewhat complex.
  - Beyond the scope

Heapsort in practice

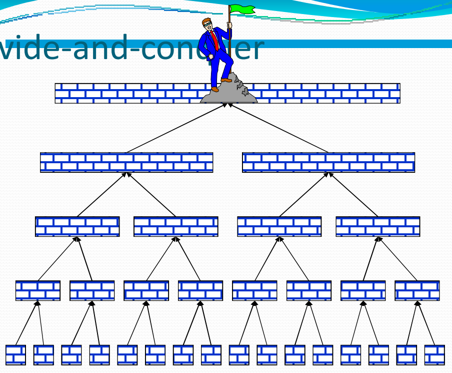
- In practice, heapsort consistently tends to use nearly Nlog N comparisons.
- So, while the worst case is better than N^2, other algorithms sometimes work better.

Recursive Sorting

Merge Sort

- Intuitively, *divide* the problem into pieces and then *recombine* the results.
  - If array is length 1, then done.
  - If array is length N>1, then split in half and sort each half.
    - Then combine the results.
- An example of a *divide-and-conquer* algorithm.

Divide-and-conquer

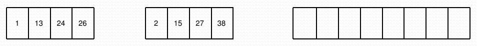


Merge Sort - Code

```
void mergeSort( AnyType [ ] A, AnyType [ ] tmpArray,
   int left, int right )
 { if( left < right )
     { int center = ( left + right ) / 2;
      mergeSort( A, tmpArray, left, center);
      mergeSort( A, tmpArray, center + 1, right );
      merge( A, tmpArray, left, center + 1, right );
     }
 }
```

Merging Two Sorted Arrays

- All the work in merge sort is done at the merge step.
- Example



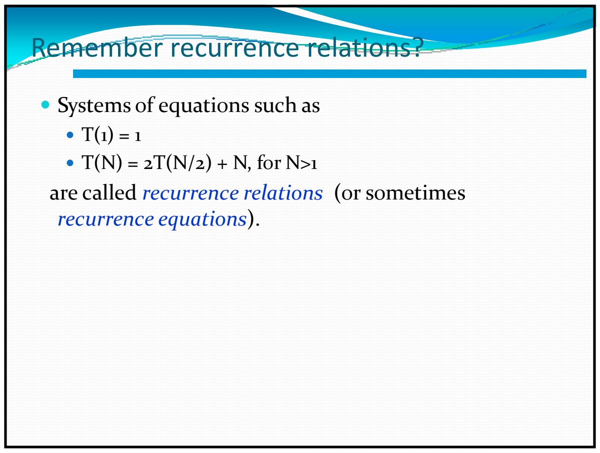
Exercise

- Write the merge code
- public void merge(A, tmpArray, left, center + 1, right) ;
  - Merges two sorted Arrays A[left, center] and A[center+1, right] using tmpArray

Tracing the code

Analysis of Merge Sort

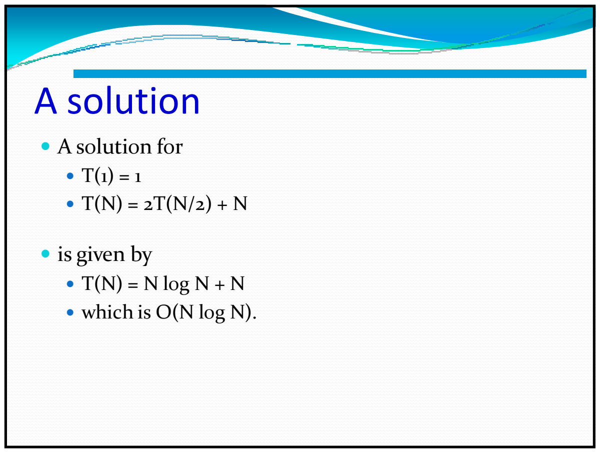
- Suppose it takes time T(N) to sort N elements.
- Suppose also it takes time N to combine the two sorted arrays.
- Then:
  - $T(1) = 1$
  - $T(N) = 2T(N/2) + N$, for $N>1$
- *Solving for T gives the running time for the merge sort algorithm.*



Remember recurrence relations?

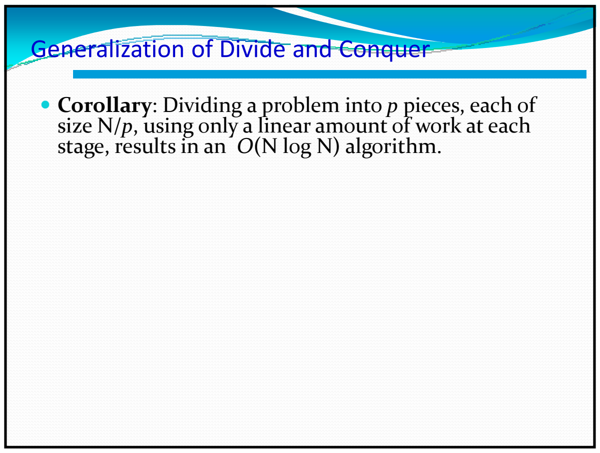
- Systems of equations such as
  - $T(1) = 1$
  - $T(N) = 2T(N/2) + N$, for $N>1$

are called *recurrence relations* (or sometimes *recurrence equations*).



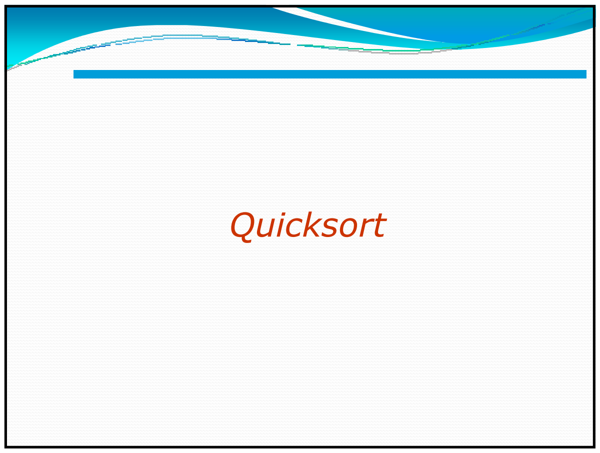
A solution

- A solution for
  - $T(1) = 1$
  - $T(N) = 2T(N/2) + N$
- is given by
  - $T(N) = N \log N + N$
  - which is $O(N \log N)$.

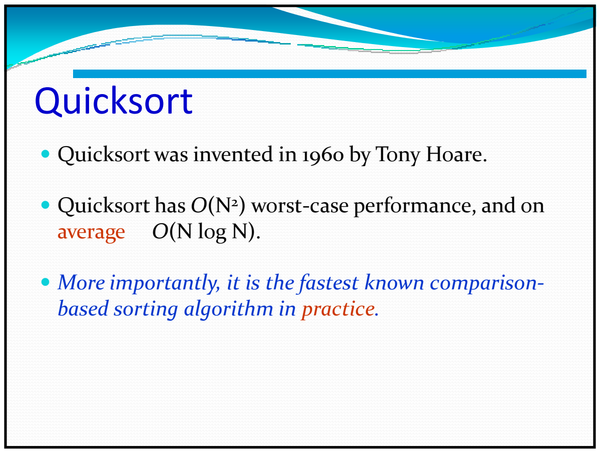


Generalization of Divide and Conquer

- **Corollary**: Dividing a problem into *p* pieces, each of size N/*p*, using only a linear amount of work at each stage, results in an *O*(N log N) algorithm.

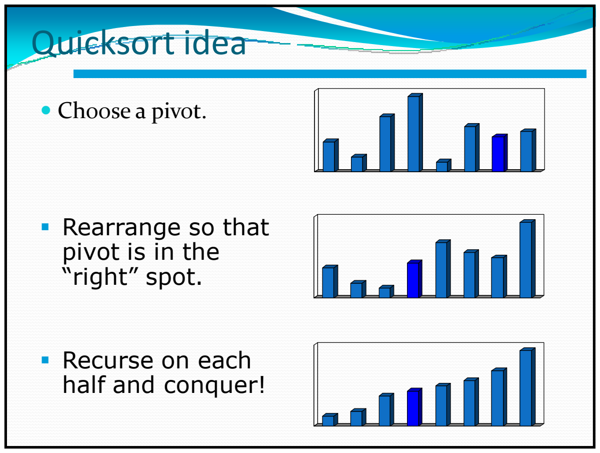


Quicksort



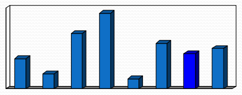
Quicksort

- Quicksort was invented in 1960 by Tony Hoare.
- Quicksort has $O(N^2)$ worst-case performance, and on average $O(N \log N)$.
- *More importantly, it is the fastest known comparison-based sorting algorithm in practice.*

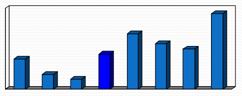


Quicksort idea

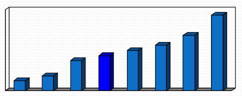
- Choose a pivot.

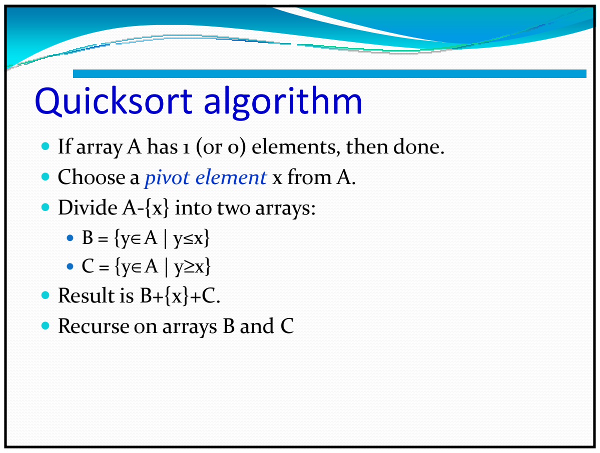


- Rearrange so that pivot is in the "right" spot.



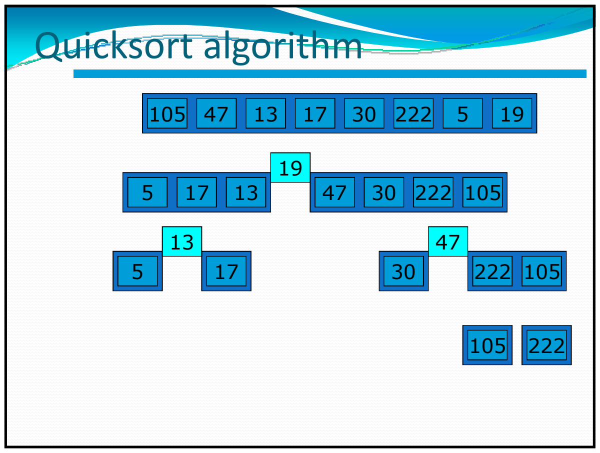
- Recurse on each half and conquer!



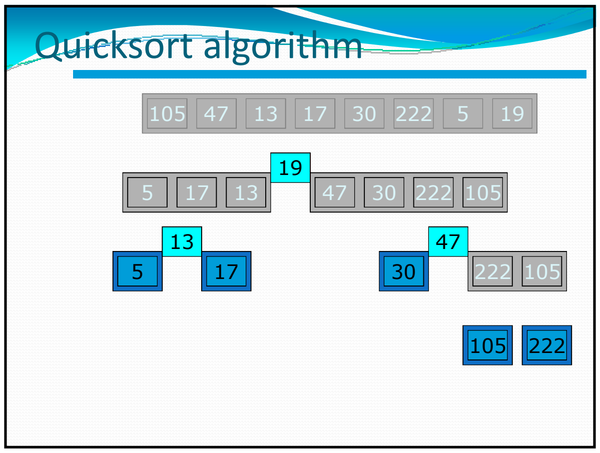


Quicksort algorithm

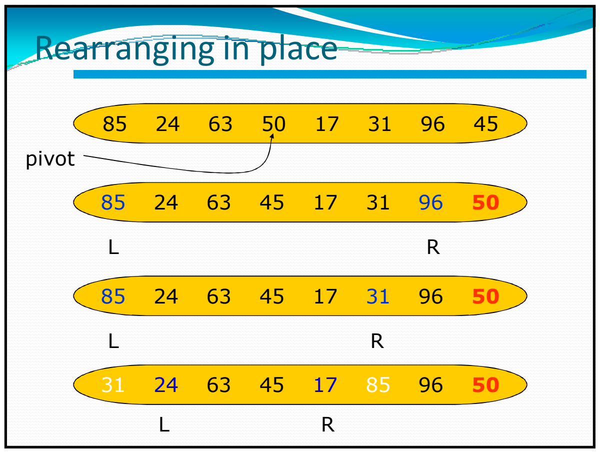
- If array A has 1 (or 0) elements, then done.
- Choose a *pivot element* x from A.
- Divide A-{x} into two arrays:
  - B = {y∈A | y≤x}
  - C = {y∈A | y≥x}
- Result is B+{x}+C.
- Recurse on arrays B and C



Quicksort algorithm

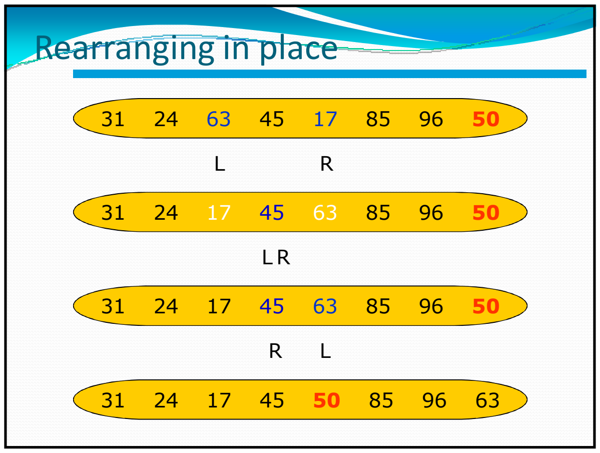


Quicksort algorithm



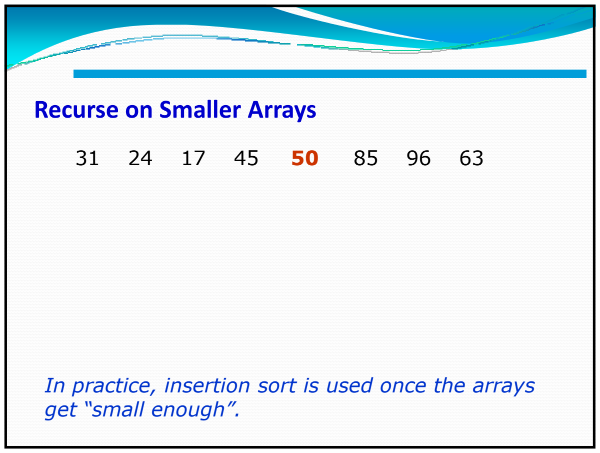
Rearranging in place

```
     85  24  63  50  17  31  96  45
pivot
     85  24  63  45  17  31  96  50
     L                           R
     85  24  63  45  17  31  96  50
     L                       R
     31  24  63  45  17  85  96  50
         L               R
```



Rearranging in place

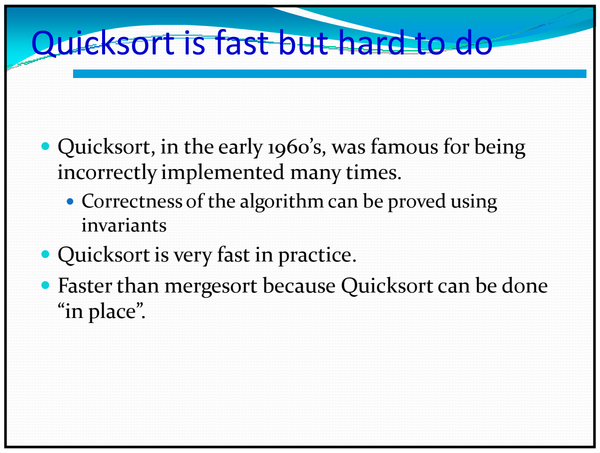
```
31  24  63  45  17  85  96  50
        L           R
31  24  17  45  63  85  96  50
            L R
31  24  17  45  63  85  96  50
            R   L
31  24  17  45  50  85  96  63
```



Recurse on Smaller Arrays

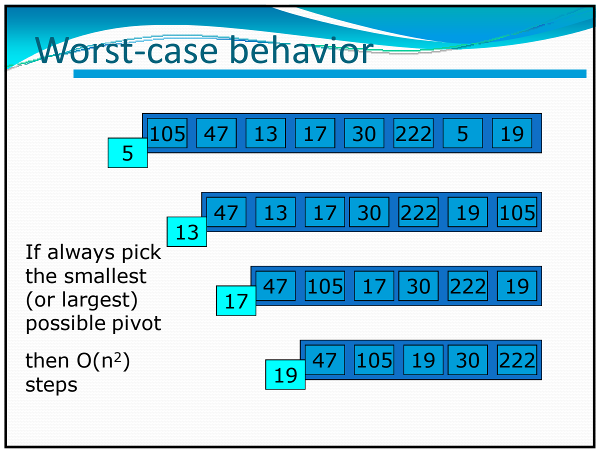
31 24 17 45 **50** 85 96 63

In practice, insertion sort is used once the arrays get "small enough".

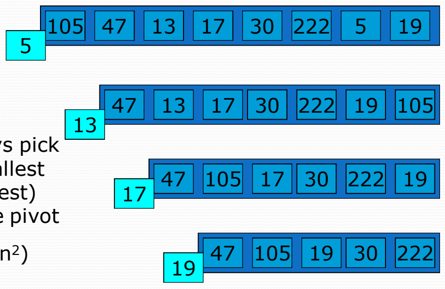


Quicksort is fast but hard to do

- Quicksort, in the early 1960's, was famous for being incorrectly implemented many times.
  - Correctness of the algorithm can be proved using invariants
- Quicksort is very fast in practice.
- Faster than mergesort because Quicksort can be done "in place".

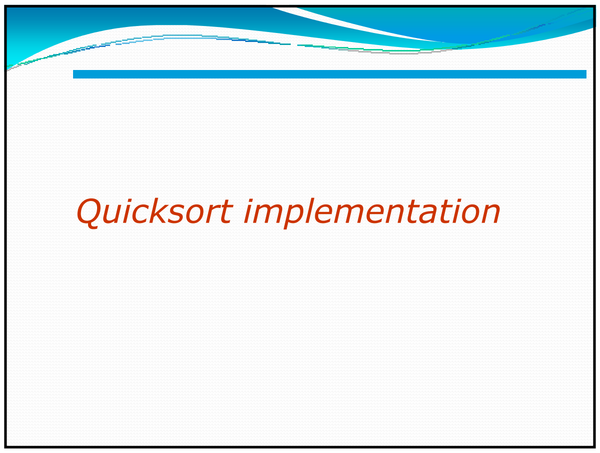


Worst-case behavior

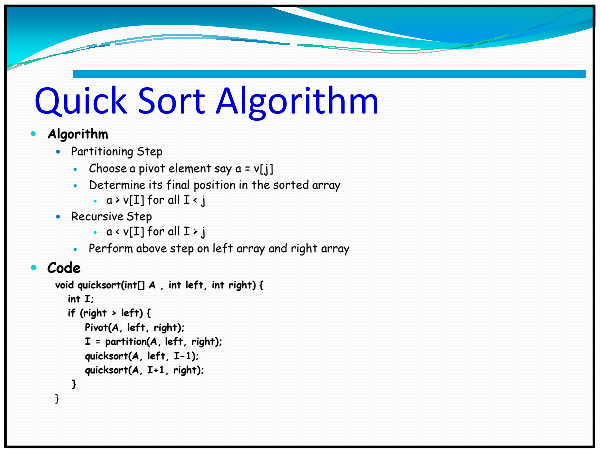


If always pick the smallest (or largest) possible pivot

then $O(n^2)$ steps



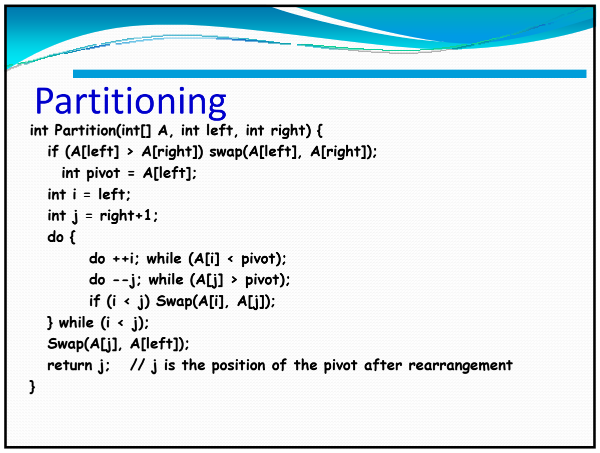
Quicksort implementation



Quick Sort Algorithm

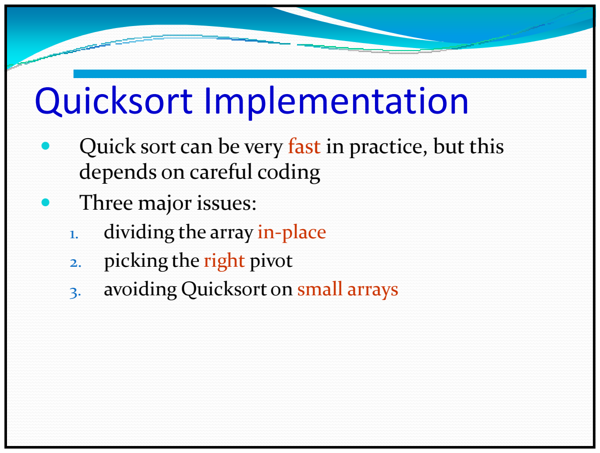
- **Algorithm**
  - Partitioning Step
    - Choose a pivot element say a = v[j]
    - Determine its final position in the sorted array
      - a > v[I] for all I < j
  - Recursive Step
      - a < v[I] for all I > j
    - Perform above step on left array and right array
- **Code**

```
void quicksort(int[] A , int left, int right) {
   int I;
   if (right > left) {
       Pivot(A, left, right);
       I = partition(A, left, right);
       quicksort(A, left, I-1);
       quicksort(A, I+1, right);
   }
}
```



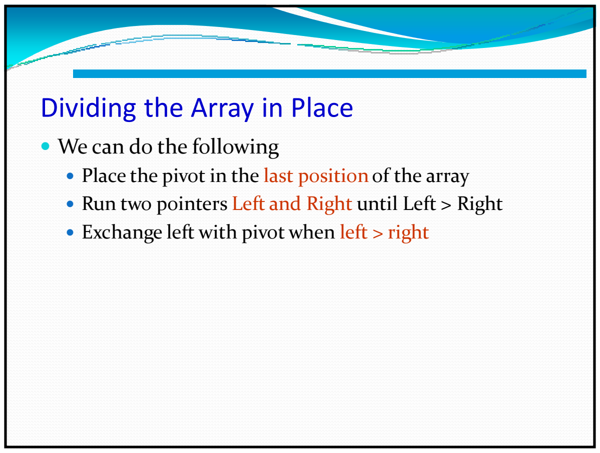
Partitioning

```
int Partition(int[] A, int left, int right) {
   if (A[left] > A[right]) swap(A[left], A[right]);
      int pivot = A[left];
   int i = left;
   int j = right+1;
   do {
           do ++i; while (A[i] < pivot);
           do --j; while (A[j] > pivot);
           if (i < j) Swap(A[i], A[j]);
   } while (i < j);
   Swap(A[j], A[left]);
   return j;   // j is the position of the pivot after rearrangement
}
```



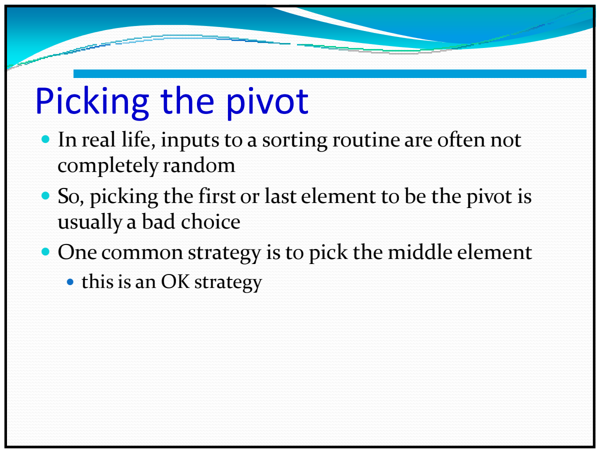
Quicksort Implementation

- Quick sort can be very fast in practice, but this depends on careful coding
- Three major issues:
  1. dividing the array in-place
  2. picking the right pivot
  3. avoiding Quicksort on small arrays



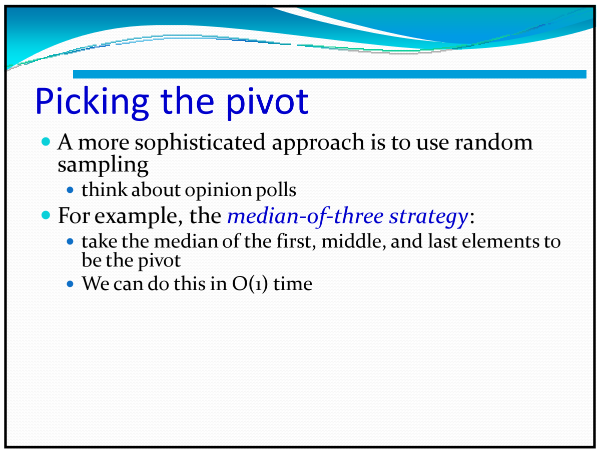
Dividing the Array in Place

- We can do the following
  - Place the pivot in the last position of the array
  - Run two pointers Left and Right until Left > Right
  - Exchange left with pivot when left > right



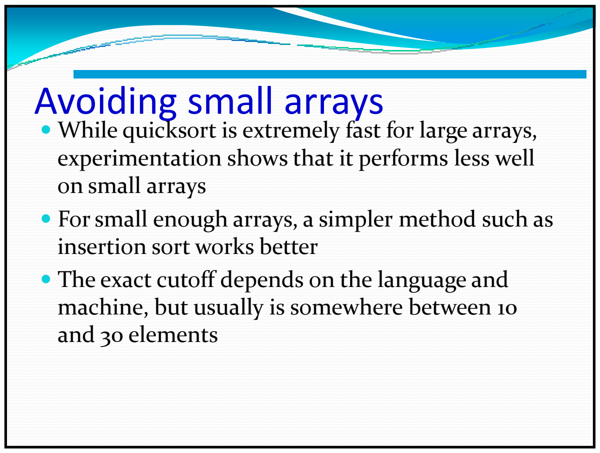
Picking the pivot

- In real life, inputs to a sorting routine are often not completely random
- So, picking the first or last element to be the pivot is usually a bad choice
- One common strategy is to pick the middle element
  - this is an OK strategy



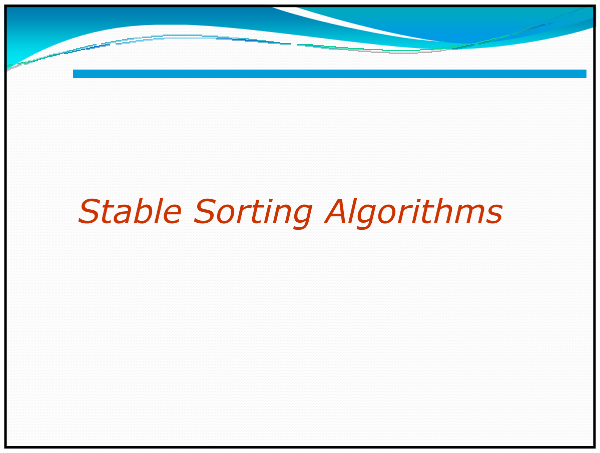
Picking the pivot

- A more sophisticated approach is to use random sampling
  - think about opinion polls
- For example, the *median-of-three strategy*:
  - take the median of the first, middle, and last elements to be the pivot
  - We can do this in O(1) time

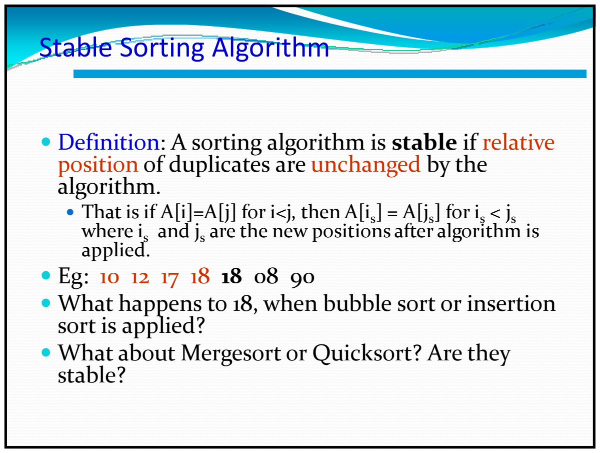


Avoiding small arrays

- While quicksort is extremely fast for large arrays, experimentation shows that it performs less well on small arrays
- For small enough arrays, a simpler method such as insertion sort works better
- The exact cutoff depends on the language and machine, but usually is somewhere between 10 and 30 elements



Stable Sorting Algorithms



Stable Sorting Algorithm

- Definition: A sorting algorithm is **stable** if relative position of duplicates are unchanged by the algorithm.
  - That is if $A[i]=A[j]$ for $i<j$, then $A[i_s] = A[j_s]$ for $i_s < j_s$ where i_s and j_s are the new positions after algorithm is applied.
- Eg: 10 12 17 18 **18** 08 90
- What happens to 18, when bubble sort or insertion sort is applied?
- What about Mergesort or Quicksort? Are they stable?

Non-Comparison based Sorting

Non-comparison-based sorting

- If we can do more than just compare pairs of elements, we can sometimes sort more quickly
- What if we can look at the bits of information?
- Two simple examples are *bucket sort* and *radix sort*

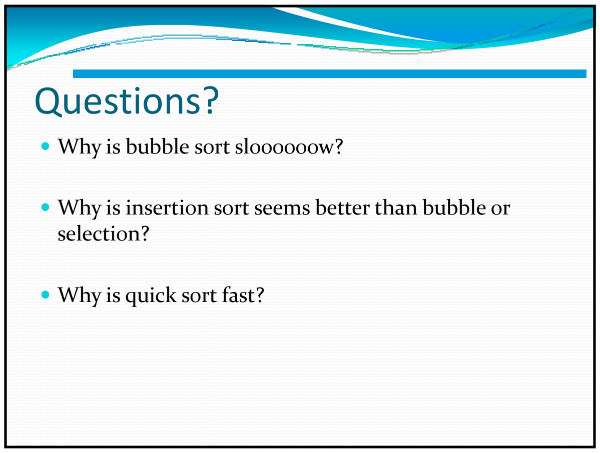
Radix Sort

Radix Sort Revisited

World's Fastest Sorters

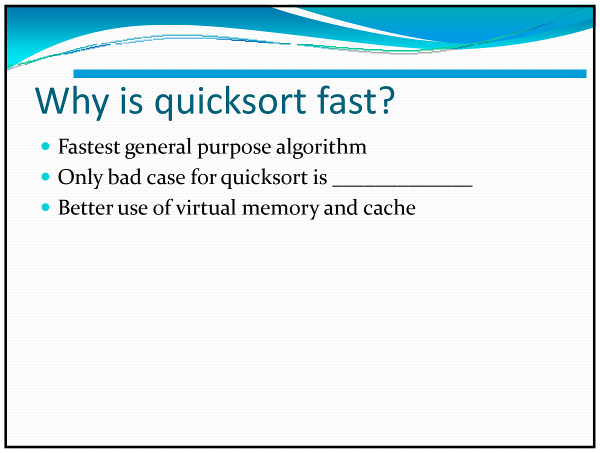
Sorting competitions

- There are several world-wide sorting competitions
  - Unix CoSort has achieved 1GB in under one minute, on a single Alpha
  - Berkeley's NOW-sort sorted 8.4GB of disk data in under one minute, using a network of 95 workstations
  - Sandia Labs was able to sort 1TB of data in under 50 minutes, using a 144-node multiprocessor machine



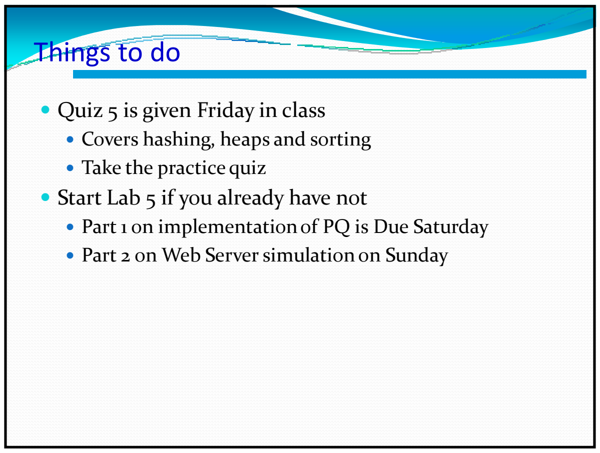
Questions?

- Why is bubble sort slooooooow?
- Why is insertion sort seems better than bubble or selection?
- Why is quick sort fast?



Why is quicksort fast?

- Fastest general purpose algorithm
- Only bad case for quicksort is ____________
- Better use of virtual memory and cache



Things to do

- Quiz 5 is given Friday in class
  - Covers hashing, heaps and sorting
  - Take the practice quiz
- Start Lab 5 if you already have not
  - Part 1 on implementation of PQ is Due Saturday
  - Part 2 on Web Server simulation on Sunday