CSSE 230 Day 22

Recurrence Relations

Sorting overview

More on Recurrence Relations

- A technique for analyzing recursive algorithms

Recap: Recurrence Relation

- An equation (or inequality) that relates the n^{th} element of a sequence to certain of its predecessors (recursive case)
- Includes an initial condition (base case)
- **Solution:** A function of n.

- Similar to differential equations, but discrete instead of continuous
- Some solution techniques are similar to diff. eq. solution techniques

Solve Simple Recurrence Relations

- One strategy: **guess and check**
- Examples:
  - $T(0) = 0,\ T(N) = 2 + T(N-1)$
  - $T(0) = 1,\ T(N) = 2\,T(N-1)$
  - $T(0) = T(1) = 1,\ T(N) = T(N-2) + T(N-1)$
  - $T(0) = 1,\ T(N) = N\,T(N-1)$
  - $T(0) = 0,\ T(N) = T(N-1) + N$
  - $T(1) = 1,\ T(N) = 2\,T(N/2) + N$

    (just consider the cases where $N=2^k$)

Another Strategy

- **Substitution**
- $T(1) = 1$, $T(N) = 2\,T(N/2) + N$
  (just consider $N=2^k$)
- Suppose we substitute N/2 for N in the recursive equation?
  - We can plug the result into the original equation!

Solution Strategies for Recurrence Relations

- Guess and check
- Substitution
- Telescoping and iteration
- The "master" method

Selection Sort

```
public static void selectionSort(int[] a) {
    //Sorts a non-empty array of integers.

    for (int last = a.length-1; last > 0; last--) {
        // find largest, and exchange with last
        int largest = a[0];
        int largePosition = 0;

        for (int j=1; j<=last; j++)
            if (largest < a[j]) {
                largest = a[j];
                largePosition = j;
        }
        a[largePosition] = a[last];
        a[last] = largest;
    }
}
```

What's N?

Another Strategy: Telescoping

- Basic idea: tweak the relation somehow so successive terms cancel

- Example: T(1) = 1, T(N) = 2T(N/2) + N
  where N = 2^k for some k
- Divide by N to get a "piece of the telescope":

$$\begin{aligned} T(N) &= 2T(\frac{N}{2}) + N \\ \Longrightarrow \frac{T(N)}{N} &= \frac{2T(\frac{N}{2})}{N} + 1 \\ \Longrightarrow \frac{T(N)}{N} &= \frac{T(\frac{N}{2})}{\frac{N}{2}} + 1 \end{aligned}$$

A Fourth Strategy: Master Theorem

- For Divide-and-conquer algorithms
  - Divide data into two or more parts
  - Solve problem on one or more of those parts
  - Combine "parts" solutions to solve whole problem
- Examples
  - Binary search
  - Merge Sort
  - MCSS recursive algorithm we studied last time

Theorem 7.5 in Weiss

Divide and Conquer Recurrence

$$T(N) = aT(\frac{N}{b}) + f(N)$$

$$a \geq 1, b > 1, \text{ and } f(N) = O(N^k)$$

- b = number of parts we divide into
- a = number of parts we solve
- f(N) = overhead of dividing and combining
- Merge sort: b =__ , a =___ , k =
- Binary Search: b =__ , a =___ , k =

The Master Theorem is convenient, but only works for divide and conquer recurrences

- For any recurrence relation *in the form*:

$$T(N) = aT(\frac{N}{b}) + f(N)$$

with $a \geq 1, b > 1, \text{ and } f(N) = O(N^k)$

- The solution is:

$$T(N) = \begin{cases} O(N^{\log_b a}) & \text{if } a > b^k \\ O(N^k \log N) & \text{if } a = b^k \\ O(N^k) & \text{if } a < b^k \end{cases}$$

Theorem 7.5 in Weiss

Summary: Recurrence Relations

- Analyze code to determine relation
  - Base case in code gives base case for relation
  - Number and “size” of recursive calls determine recursive part of recursive case
  - Non-recursive code determines rest of recursive case
- Apply one of four strategies
  - Guess and check
  - Substitution (a.k.a. iteration)
  - Telescoping
  - Master theorem

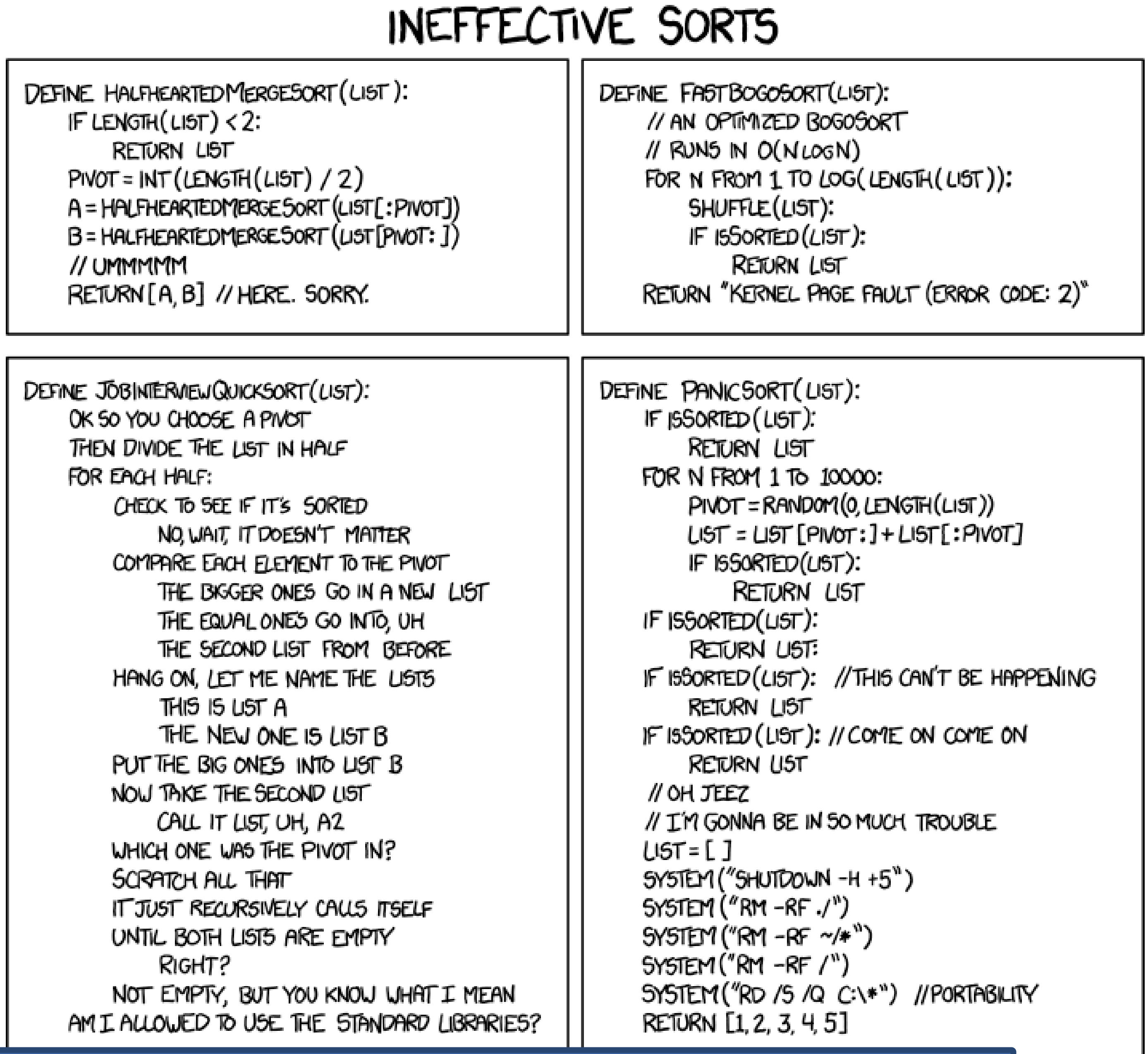
Sorting overview

- Quick look at several sorting methods
- Focus on quicksort
- Quicksort average case analysis

Elementary Sorting Methods

- Name as many as you can
- How does each work?
- Running time for each (sorting N items)?
  - best
  - worst
  - average
  - extra space requirements
- Spend 10 minutes with a group of three, answering these questions. Then we will summarize

Put list on board



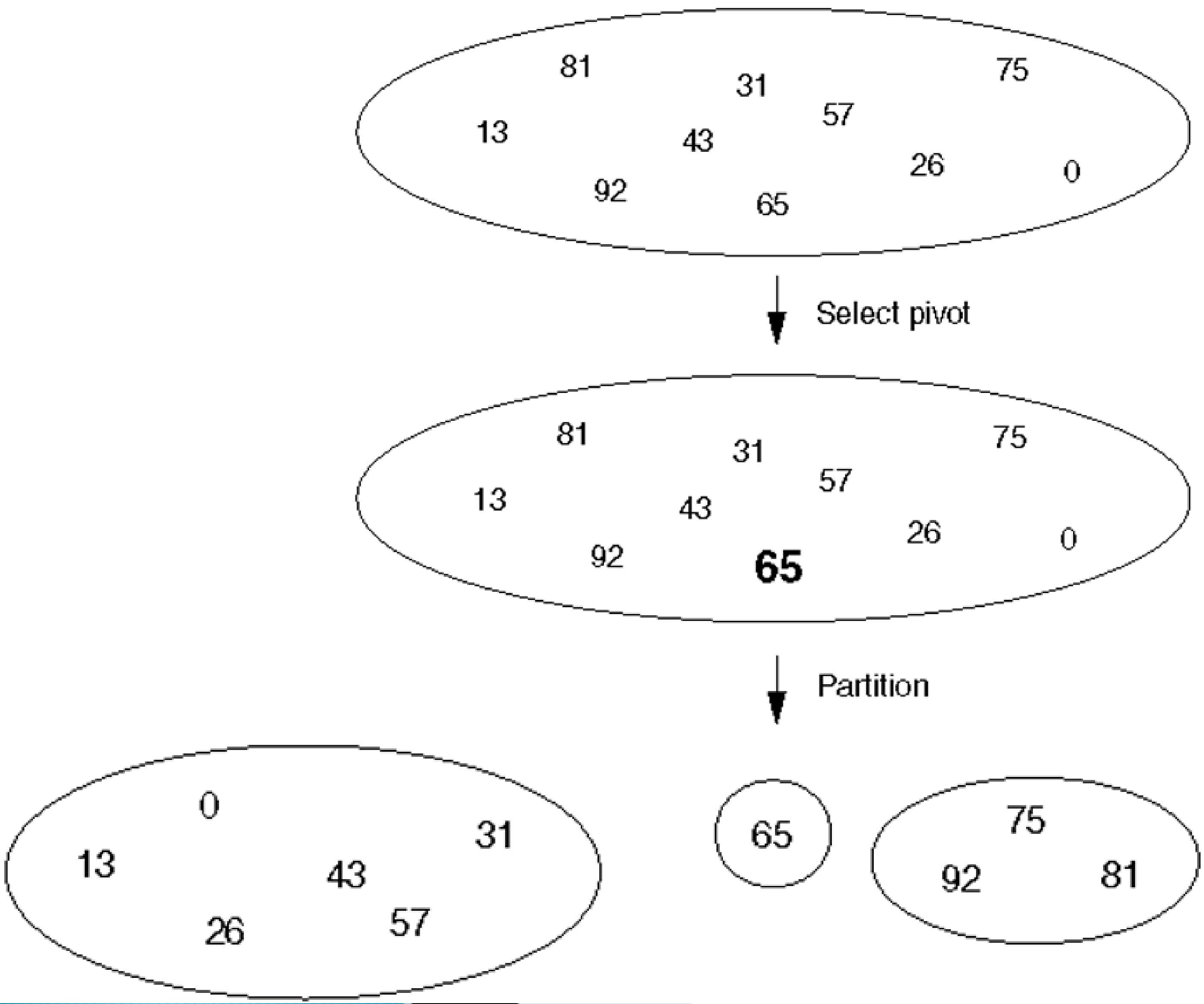
http://www.xkcd.com/1185/

Stacksort connects to StackOverflow, searches for “sort a list”, and downloads and runs code snippets until the list is sorted.

QuickSort (a.k.a. “partition-exchange sort”)

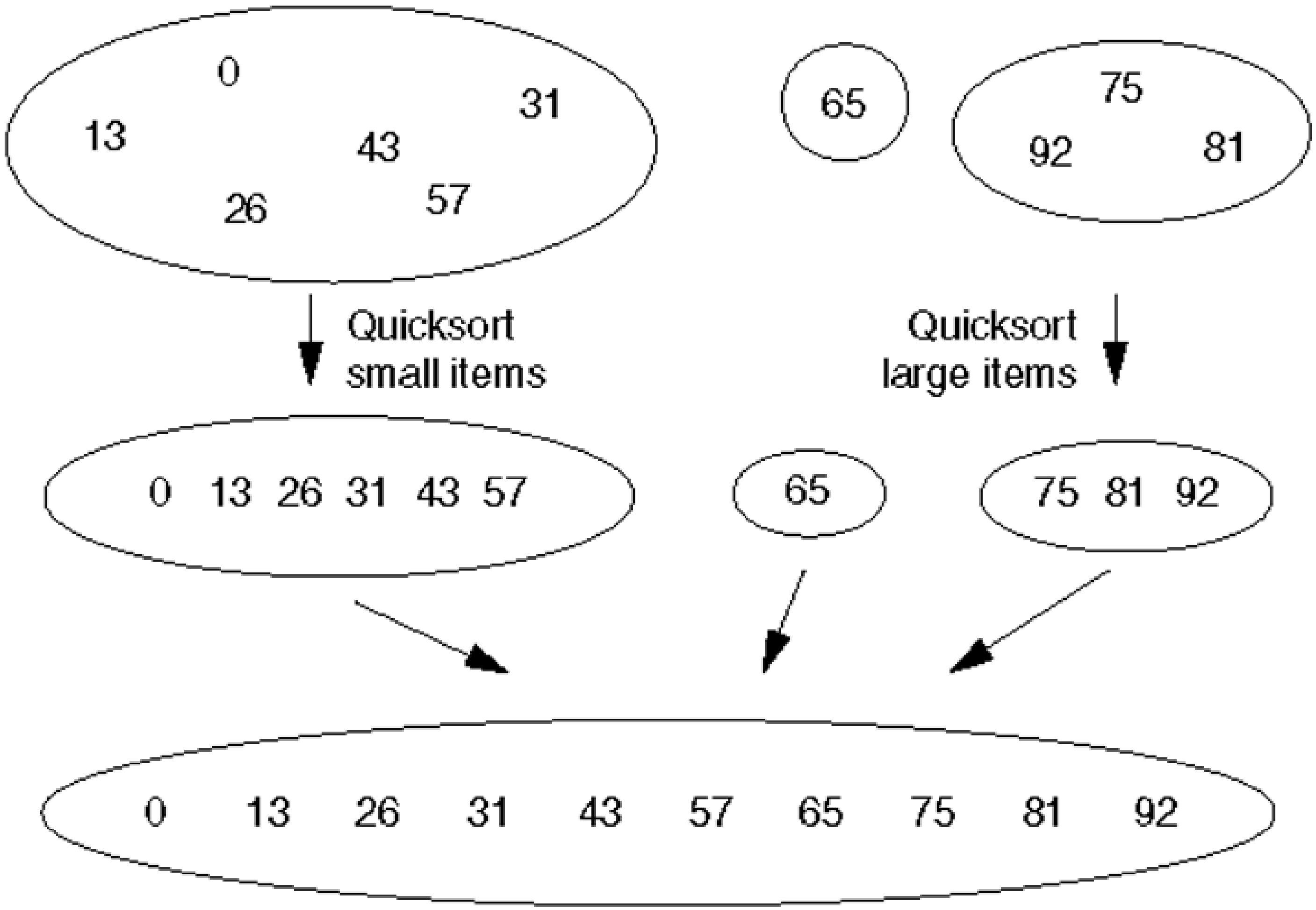
- Invented by C.A.R. Hoare in 1961
- Very widely used
- Somewhat complex, but fairly easy to understand

Partition: split the array into 2 parts: smaller than pivot and greater than pivot



Quicksort then recursively calls itself on the partitions

Q4



Partition: efficiently move small elements to the left of the pivot and greater ones to the right

Q5

```
// Assume min and max indices are low and high
pivot = a[low]
i = low+1, j = high
while (true) {
  while (a[i] < pivot) i++
  while (a[j] > pivot) j--
  if (i >= j) break
  swap(a, i, j)
}
swap(a, low, j) // moves the pivot to the
          // correct place
```

QuickSort Average Case

- Running time for partition **of N elements** is $\Theta(N)$
- Quicksort Running time:
  - call partition. Get two subarrays of sizes N_L and N_R (what is the relationship between N_L, N_R, and N?)
  - Then Quicksort the smaller parts
  - $T(N) = N + T(N_L) + T(N_R)$
- Quicksort Best case: write and solve the recurrence
- Quicksort Worst case: write and solve the recurrence
- average: a little bit trickier
  - We have to be careful how we measure

Average time for Quicksort

- Let $T(N)$ be the average # of comparisons of array elements needed to quicksort N elements.
- What is $T(0)$? $T(1)$?
- Otherwise $T(N)$ is the sum of
  - time for partition
  - average time to quicksort left part: $T(N_L)$
  - average time to quicksort right part: $T(N_R)$
- $T(N) = N + T(N_L) + T(N_R)$

We need to figure out for each case, and average all of the cases

- Weiss shows how **not** to count it:
- What if we picked as the partitioning element the smallest element half of the time and the largest half of the time?
- Then on the average, $N_L = N/2$ and $N_R = N/2$,
  - but that doesn't give a true picture of this worst-case scenario.
  - In every case, either $N_L = N-1$ or $N_R = N-1$

We assume that all positions for the pivot are equally likely

Q8a

- We always need to make some kind of "distribution" assumptions when we figure out Average case
- When we execute
  `k = partition(pivot, i, j)`,
  all positions i..j are equally likely places for the pivot to end up
- Thus N_L is equally likely to have each of the values 0, 1, 2, … N−1
- $N_L + N_R = N-1$; thus N_R is also equally likely to have each of the values 0, 1, 2, … N−1
- Thus $T(N_L) = T(N_R) =$

Continue the calculation

- $T(N) =$
- Multiply both sides by N
- Rewrite, substituting N–1 for N
- Subtract the equations and forget the insignificant (in terms of big-oh) –1:
  - $NT(N) = (N+1)T(N-1) + 2N$
- Can we rearrange so that we can telescope?

Continue continuing the calculation

- $NT(N) = (N+1)T(N-1) + 2N$
- Divide both sides by $N(N+1)$
- Write formulas for $T(N)$, $T(N-1)$,$T(N-2)$ …$T(2)$.
- Add the terms and rearrange.
- Notice the familiar series
- Multiply both sides by $N+1$.

Improvements to QuickSort

- Avoid the worst case
  - Select pivot from the middle
  - Randomly select pivot
  - Median of 3 pivot selection.
  - Median of k pivot selection
- "Switch over" to a simpler sorting method (insertion) when the subarray size gets small.

Other Sorting Demos

- http://maven.smith.edu/~thiebaut/java/sort/demo.html
- http://www.cs.ubc.ca/~harrison/Java/sorting-demo.html