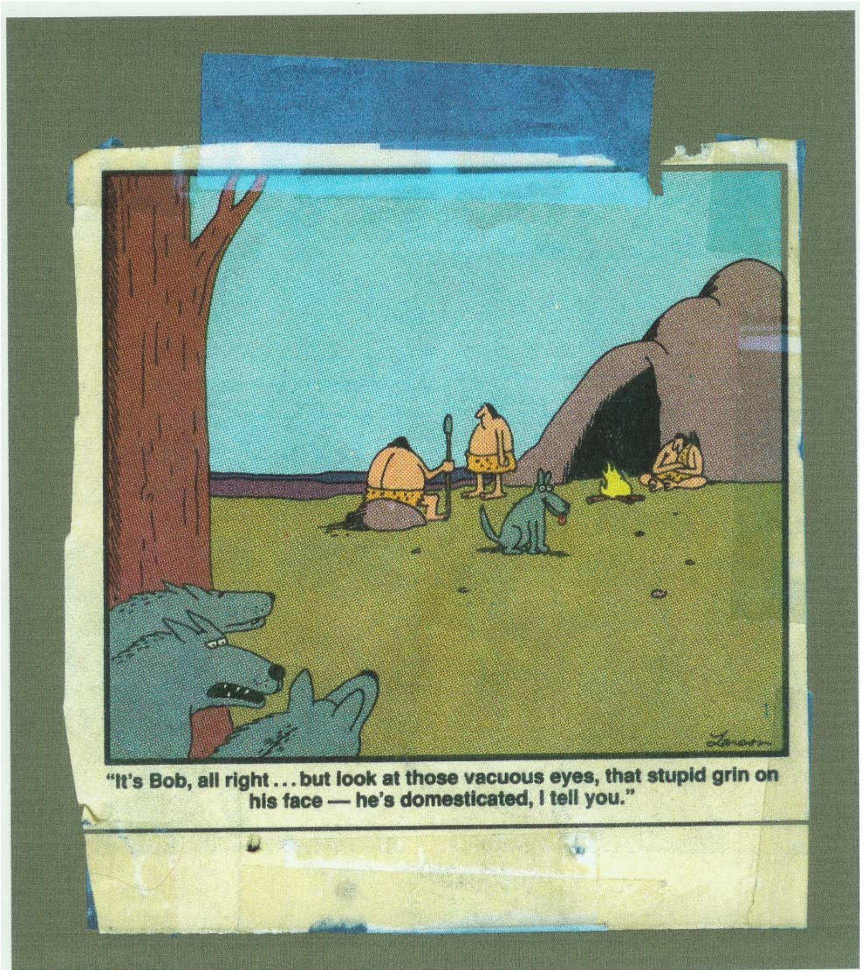


CSSE 230
Day 23

Quicksort
EditorTrees Work Time

Review: The Master Theorem works for divide-and-conquer recurrence relations ... and works well!

Q1-3

- For any recurrence relation of 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

Sorting Demos

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

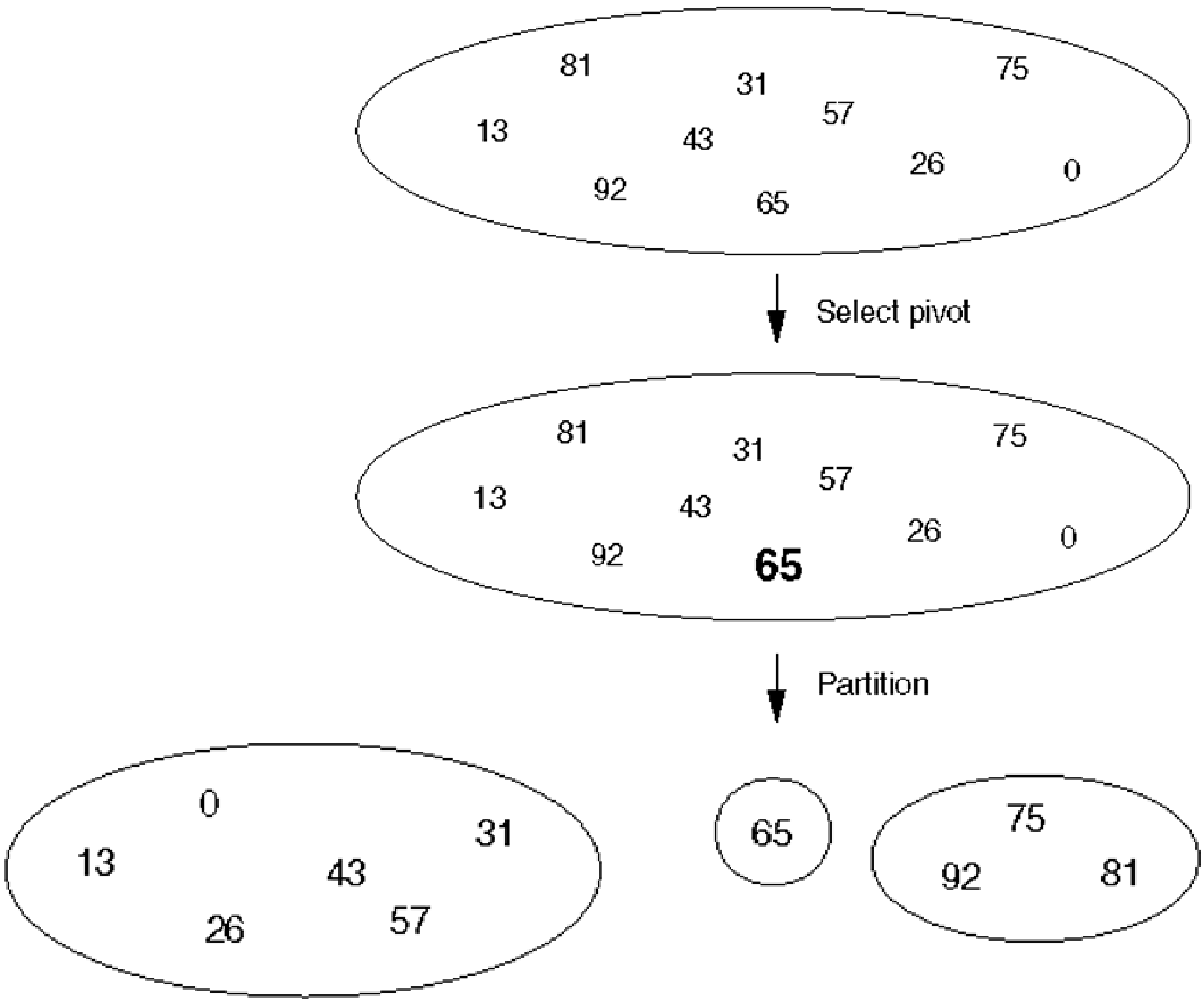
QuickSort (a.k.a. "partition-exchange sort")

- Invented by C.A.R. "Tony" Hoare in 1961*
- Very widely used
- Somewhat complex, but fairly easy to understand
  - Like in basketball, it's all about planting a good pivot.

*See Tony's own story about how it happened, at http://research.microsoft.com/en-us/people/thoare/.

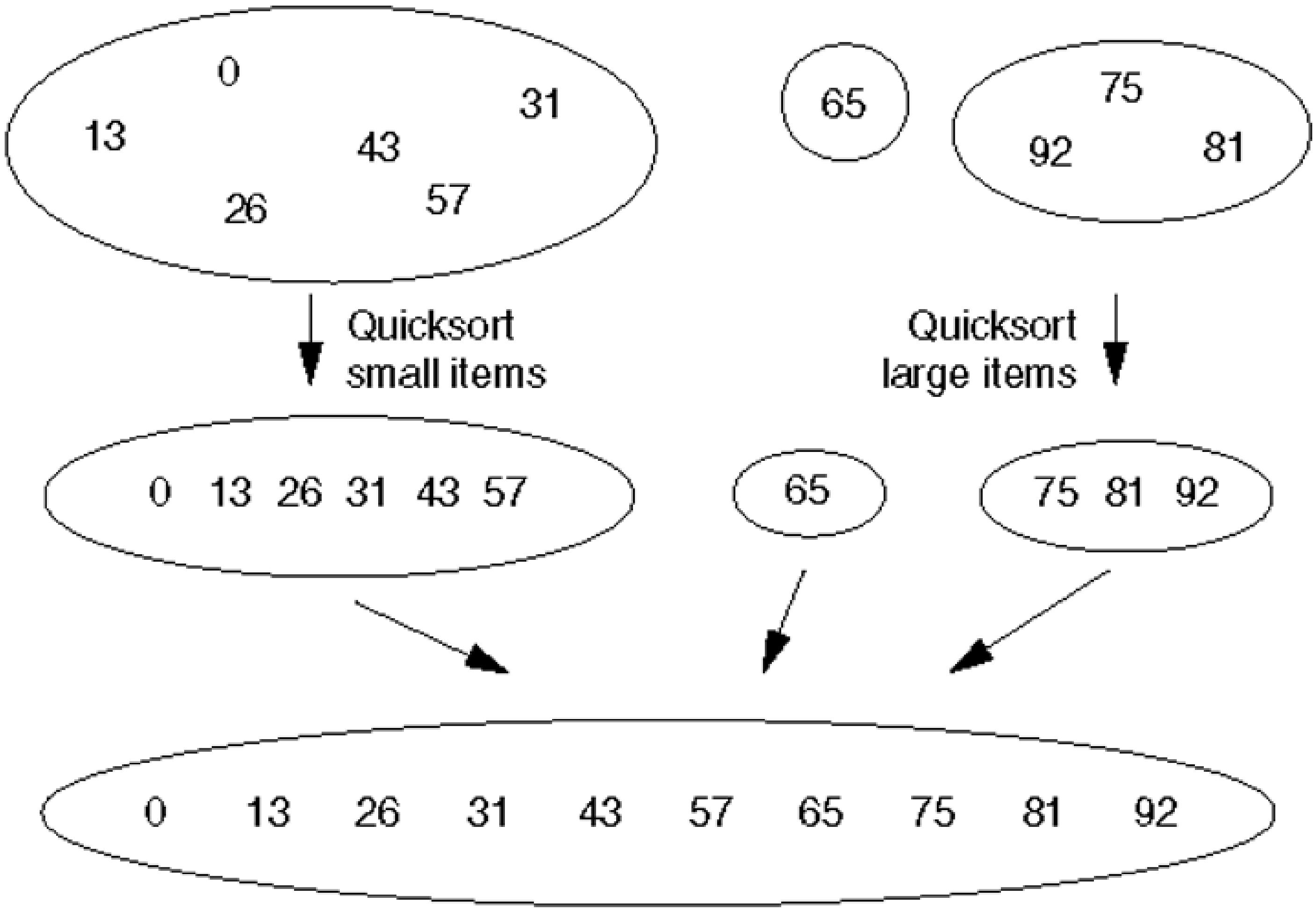
Image from http://www.ultimate-youth-basketball-guide.com/pivot-foot.html.

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
return j
```

Q6–7

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

Q8

- 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 $N+1$.

Recap

- Best, worst, average time for Quicksort
- What causes the worst case?

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

  Weiss's code does Median of 3 and switchover to insertion sort at 10.
  - Linked from schedule page