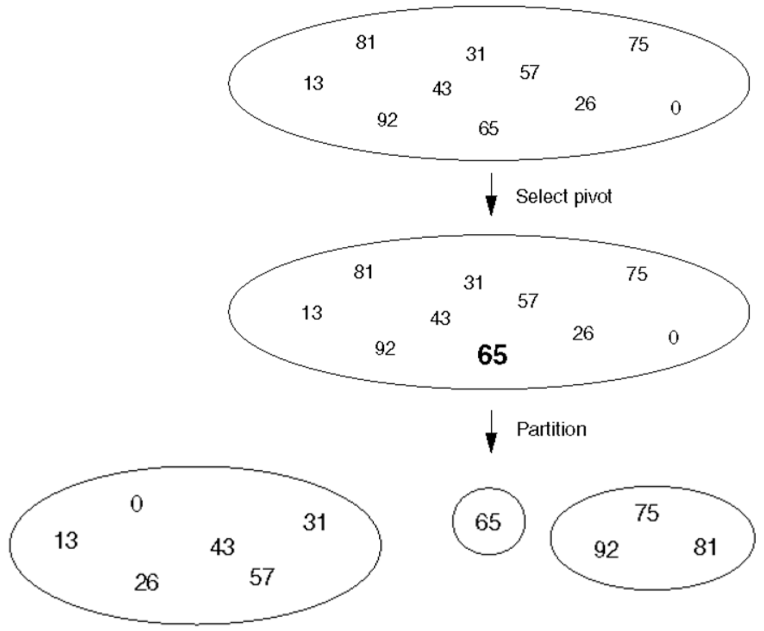


CSSE 230

Quicksort algorithm
Average case analysis

After today, you should be able to…
…implement quicksort
…derive the average case runtime of quick sort and similar algorithms

Announcements

Review: The Master Theorem works for divide-and-conquer recurrence relations only … but works well!

For any recurrence relation ***in the form***:

$$T(N) = aT\left(\frac{N}{b}\right) + \theta(N^k), with\, a \geq 1, b > 1$$

The solution is:

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

Theorem 7.5 in Weiss

Sorting Demos

- Check out now:
  - www.sorting-algorithms.com
  - https://www.youtube.com/watch?v=kPRA0W1kECg
  - https://www.youtube.com/watch?v=y9Ecb43qw98
  - http://www.cs.ubc.ca/~harrison/Java/sorting-demo.html

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

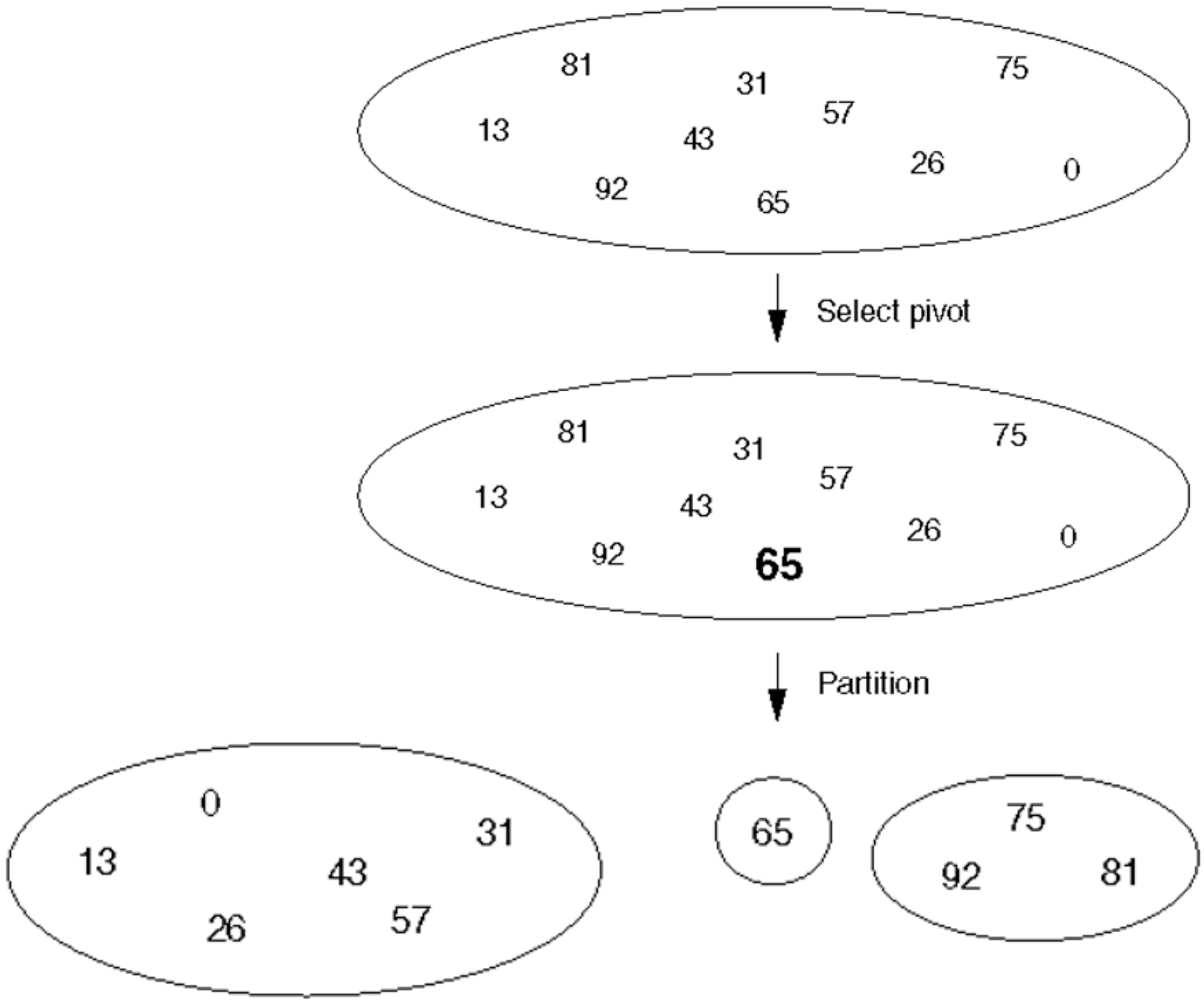
- Invented by C.A.R. "Tony" Hoare in 1961*
- Very widely used
- Guiding principle:
  - Like in basketball, it's all about planting a good pivot.

A quote from Tony Hoare:

> There are two ways of constructing a software design: One way is to make it so simple that there are obviously no deficiencies, and the other way is to make it so complicated that there are no obvious deficiencies. The first method is far more difficult.

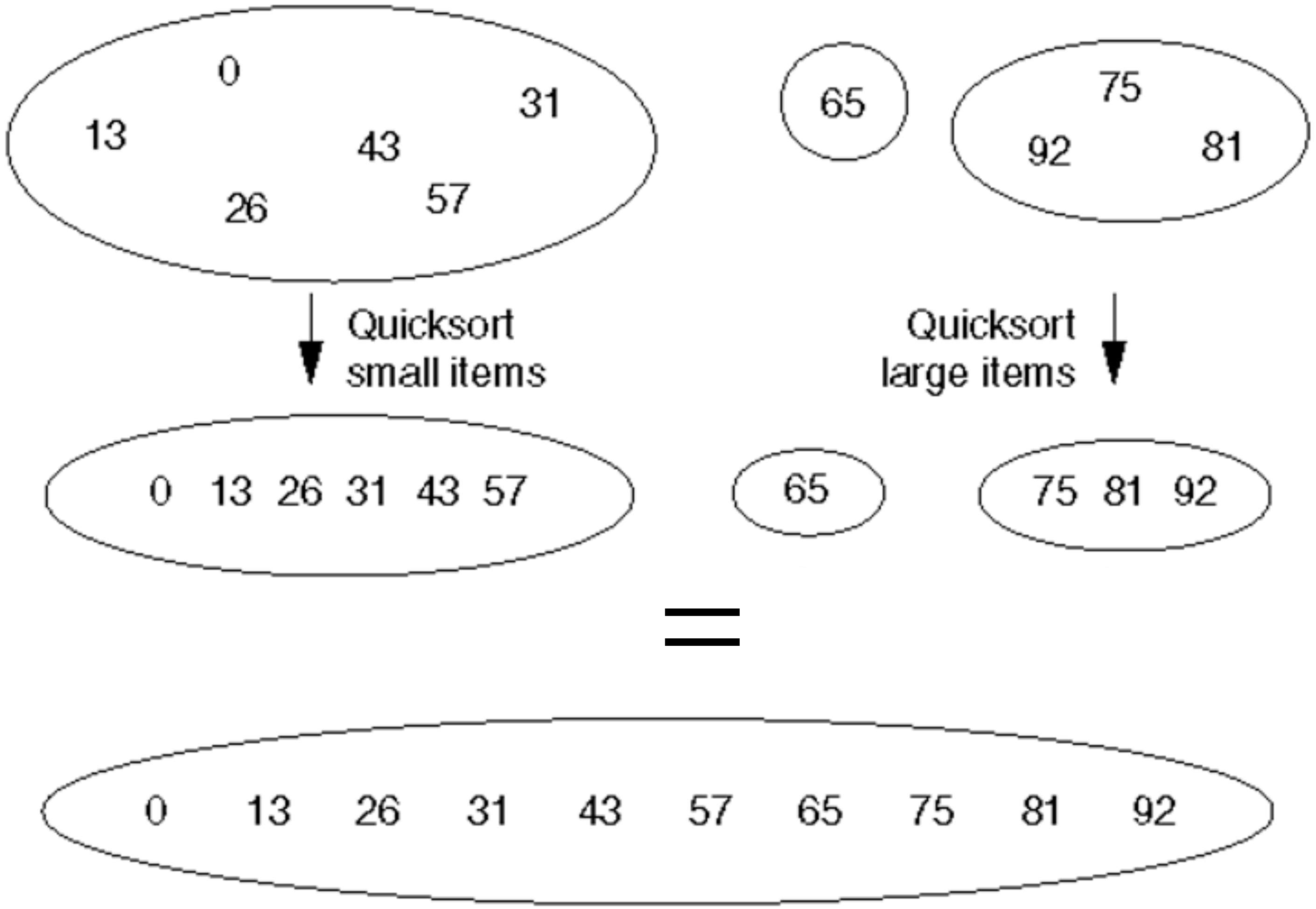
Image from http://www.ultimate-youth-basketball-guide.com/pivot-foot.html.

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] // will do better later…
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
```

Recurrences for Quicksort

- Let T(N) be the average # of comparisons of array elements needed to quicksort N elements.
- What is T(1)?
- Otherwise T(N) is the sum of
  - time for partition
  - time to quicksort left part: $T(N_L)$
  - time to quicksort right part: $T(N_R)$
- $T(N) = N + T(N_L) + T(N_R)$
- What's the best case? What's the worst case?
- Write and solve each now!

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

- Harder than just a single case…

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
- Assume that 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 $T(N_L) = T(N_R) =$

Q9-10

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

- $NT(N) = (N+1)T(N-1) + 2N$

- Solve using telescoping and iteration:
  - 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?

- We can guarantee we never hit the worst case
  - How?
  - But this makes quicksort slower than merge sort in practice.

Improvements to QuickSort

- Avoid the worst case
  - Select pivot from the middle
  - Randomly select pivot
  - **Median of 3 pivot selection. (You'll want this.)**
  - 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

What does the official Java Quicksort do? See the source code!
(Search for "OpenJDK collections", "OpenJDK Arrays", etc.)

Final notes

The partition code I gave you has 2 bugs:

1. **It can walk off the end of the array**
2. **If the chosen pivot is duplicated, it can go into an infinite recursion (stack overflow)**

```
// Assume min and max indices are low and high
pivot = a[low] // can do better
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
```