Sorting Algorithms

Seen insertion sort - good for nearly sorted data but worst-case $\Theta(n^2)$ time

Seen merge sort - worst-case $\Theta(n \log n)$ time $\left(T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n)\right)$

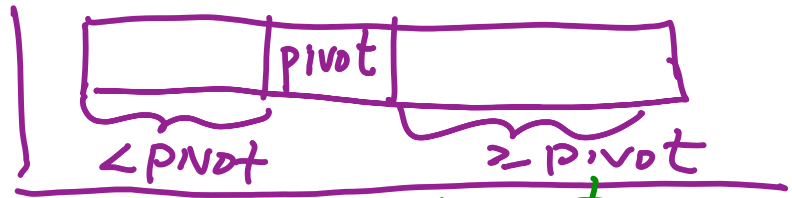
Today we'll study quicksort

Quicksort

Divide-and-Conquer Alg

Do all the hard work in splitting (+ recursive call). No combine.

Divide: Partition array into two subarrays where all elements in left portion are less than all elements in right portion



Example of Partition

pivot

… 11, 4, 9, 7, 3, 10, 2, 6, 8 …

i j

… 6, 4, 9, 7, 3, 10, 2, 11, 8

i j

≥ pivot

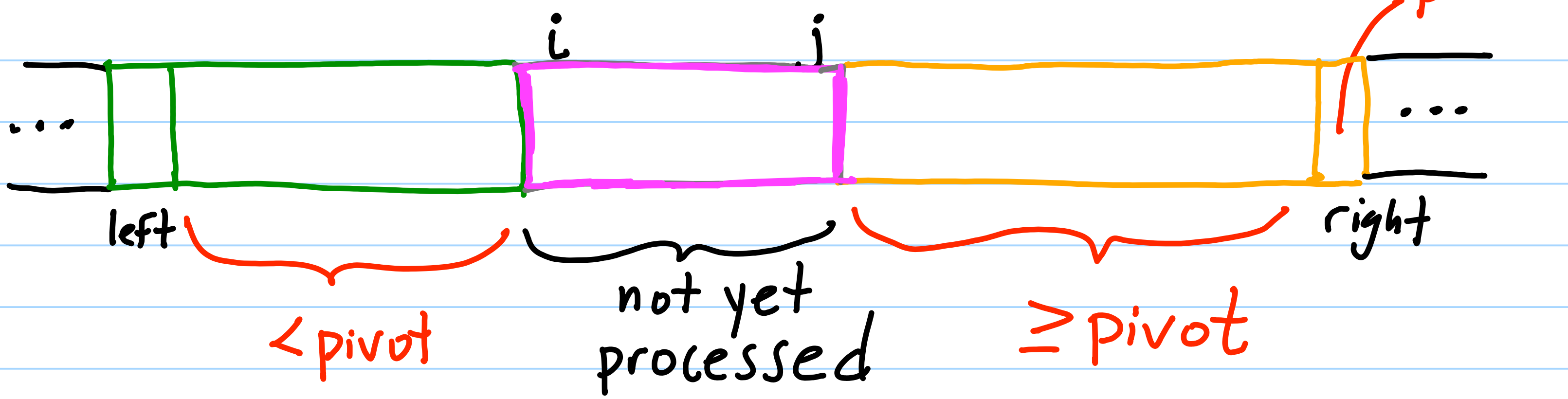
…. 6, 4, 2, 7, 3, 10, 9, 11, 8

j i

…. 6, 4, 2, 7, 3, 8, 9, 11, 10

i

General invariant maintained



Java Code For Partition

final position of pivot element

comparator that defines the sorted order

```
int partition(int left, int right, Comparator<? super E> sorter){
    E pivot = read(right);    a[right]                     //pivot around the right element
    int i = left;                                          //positions left...i-1 hold elements < pivot
    int j = right;                                         //positions j...right-1 hold elements >= pivot
    while (i < j){
        while (i < j && sorter.compare(read(i), pivot) < 0)  //pos. i element < pivot
            i++;
        while (j > i && sorter.compare(read(j), pivot) ≥ 0)  //pos i element ≥ pivot
            j--;
        if (i < j)
            swapImpl(i, j);  // swaps pos i and pos j elements
    }
    swapImpl(i, right);
    return i;
}
```

method of positional collection interface

```
Quicksort (Comparator sorter) {
    quicksortImpl (0, n-1, sorter)
}
```

int, int, comp.

```
quicksortImpl (left, right, sorter) {
    if (left<right) {
```

swap what's in pos right using median-of-three or by random

```
        mid = partition(left, right, sorter);
        quicksortImpl (left, mid-1, sorter);
        quickSortImpl (mid+1, right, sorter);
    }
}
```

Time Complexity

Asymptotic time complexity for Partition is $\Theta(n)$

median

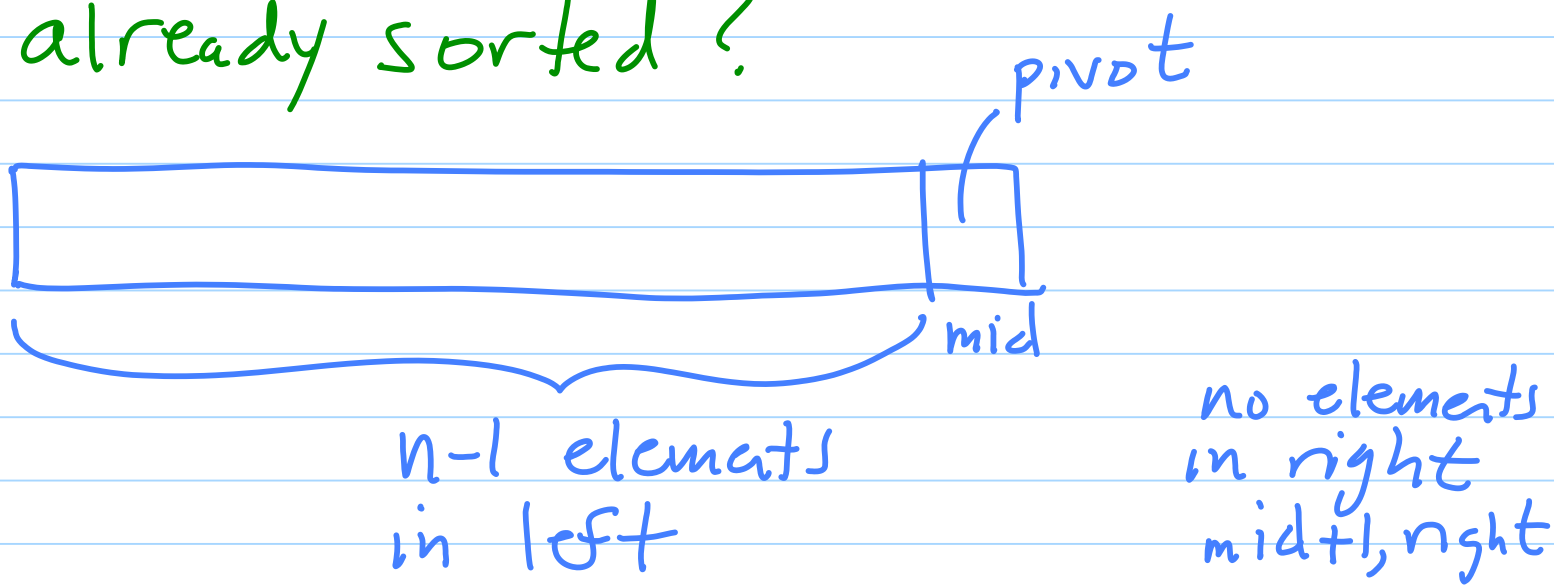
Best Case: Pivot is in middle

$$T(n) = 2T\left(\frac{n-1}{2}\right) + \Theta(n) = \Theta(n \log n)$$

What's the worst-case?

What if pivot is min or max?

What if positional collection was already sorted?



Leads to recurrence

$$T(n) = T(n-1) + \Theta(n)$$

$c \cdot n$

n	cn	
$n-1$	$c(n-1)$	
$n-2$	$c(n-2)$	
$\vdots$		
2	$c \cdot 2$	
1	c	

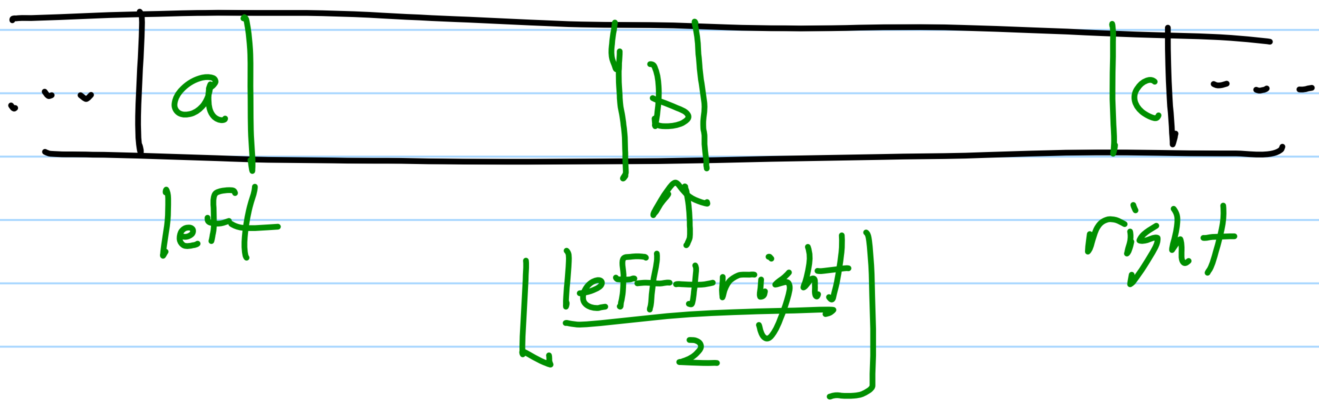
sum

$$c(1+2+\cdots+n)$$

$$\frac{c \cdot n(n+1)}{2} = \Theta(n^2)$$

How can we pick the partition to try to avoid bad behavior.

Median-of-three partitioning



Find median of a, b & c & then if this is not c, swap c with median

The other common solution is

Randomized Quicksort

Pick a random element (uniformly) from subarray + swap that with element in rightmost position of the subarray

if X items in subarray pick each with prob $1/X$

Show highlights that expected (average) time is $\Theta(n \log n)$

One common optimization is as follows.

1. Terminate when subarray size is relatively small ($n \sim 30$)

not sorted	not sorted	...	not sorted
~ 30	~ 30		~ 30

all $\leq$ all $\leq \cdots \leq$ all

2. Run insertion sort after quick sort is done