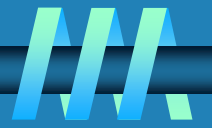
CS 395 – Analysis of Algorithms

Chapter 7 – Space and Time Trade-Offs

Read 7.1

- **Sorting by Comparison**
  - **(CSE 326: Zasha Weinberg in lieu of Steve Wolfman)**

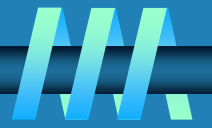
Sorting by Comparison algorithms



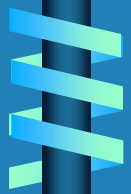
- **Simple: Selection Sort**
  - **(some say Insertion Sort, or Bubble Sort)**
- **Quick: QuickSort**
- **Good worst case: MergeSort, HeapSort**

- **Can we do better?**

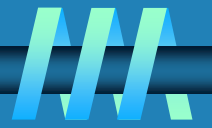
Selection Sort idea



- Find the smallest element, put it first
- Find the next smallest element, put it second
- Find the next smallest, put it next
- etc.



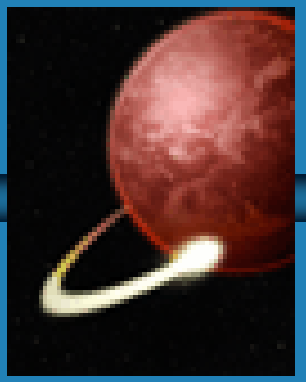
Selection Sort



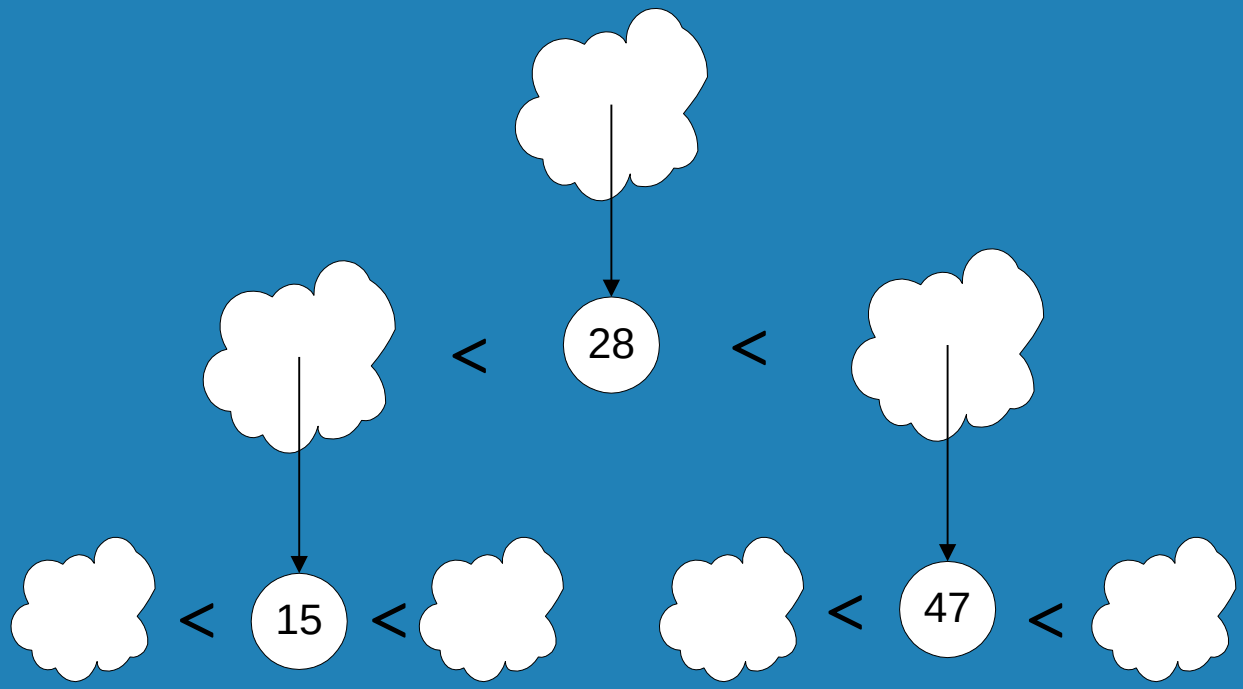
```
procedure SelectionSort (Array[1..N]

For i=1 to N-1
      Find the smallest entry in Array[i..N]
      Let j be the index of that entry
      Swap(Array[i],Array[j])
End For

While other people are coding QuickSort/MergeSort
      Twiddle thumbs
End While
```

(It may be O(n^2), but it's very easy to implement)

QuickSort

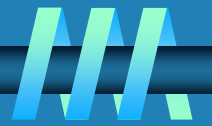


Picture from PhotoDisc.com



Pick a “pivot”. Divide into less-than & greater-than pivot.
Sort each side recursively.

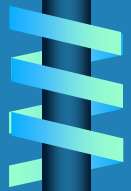
QuickSort Partition



Pick pivot:

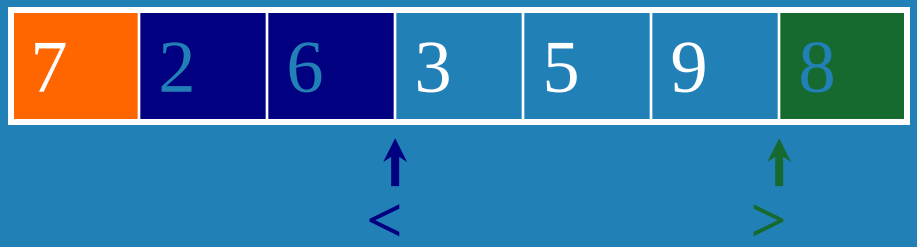
Partition with cursors

2 goes to less-than



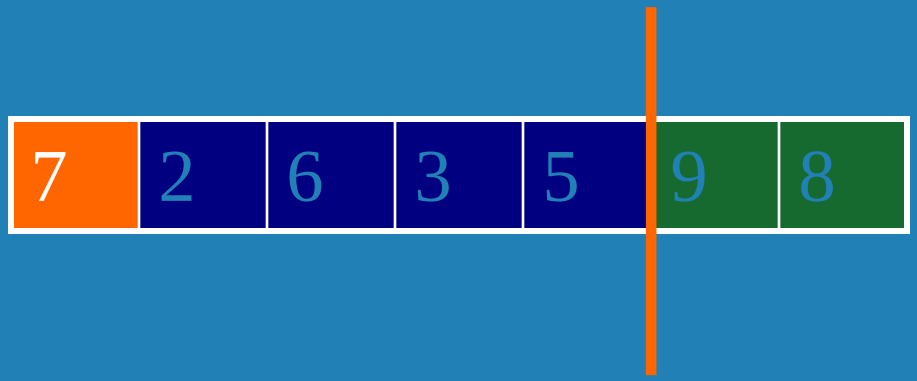
QuickSort Partition (cont'd)

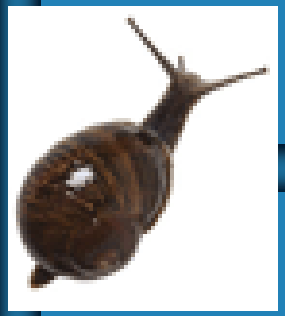
6, 8 swap
less/greater-than



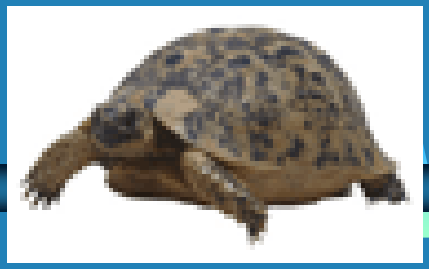
3,5 less-than
9 greater-than

Partition done.
Recursively
sort each side.



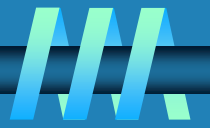


QuickSort :Worst case



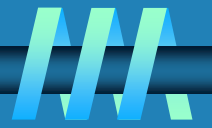
- Assume array was already sorted (but we didn't know it)… How long would Quicksort take in that case?
- Worst case is quadratic – but typical/avg. case is asymptotically optimal ~ n log n

Dealing with Slow QuickSorts



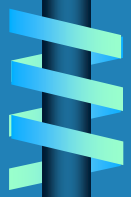
- **Randomly permute input**
  - **Bad cases more common than simple probability would suggest. So, make it truly random.**
- **Pick pivot cleverly**
  - **“Median-of-3” rule takes Median(first, middle, last element elements).**
- **Use MergeSort or HeapSort**

QuickSort:Average Case



- Model expected # of operations
- Manipulate the recurrences
- - Use Master Theorem!

- Get O(n log n)



MergeSort

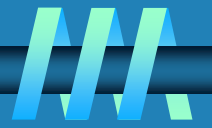
MergeSort (Table [1..n])
Split Table in half
Recursively sort each half
Merge two halves together

Merging Cars by key [Aggressiveness of driver]. Most aggressive goes first.

Photo from http://www.nrma.com.au/inside-nrma/m-h-m/road-rage.html

```
Merge (T1[1..n],T2[1..n])
i1=1, i2=1
While i1<n, i2<n
        If T1[i1] < T2[i2]
                Next is T1[i1]
                i1++
        Else
                Next is T2[i2]
                i2++
        End If
End While
```

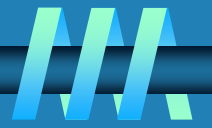
Could we do better?*



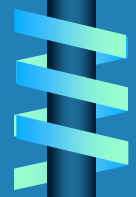
- … For any possible correct Sorting by Comparison algorithm?
  - What is lowest best case time?
  - What is lowest worst case time?
  - For the worst-case, Θ(n log n) is optimal

* (So the answer is: No, sorry.)

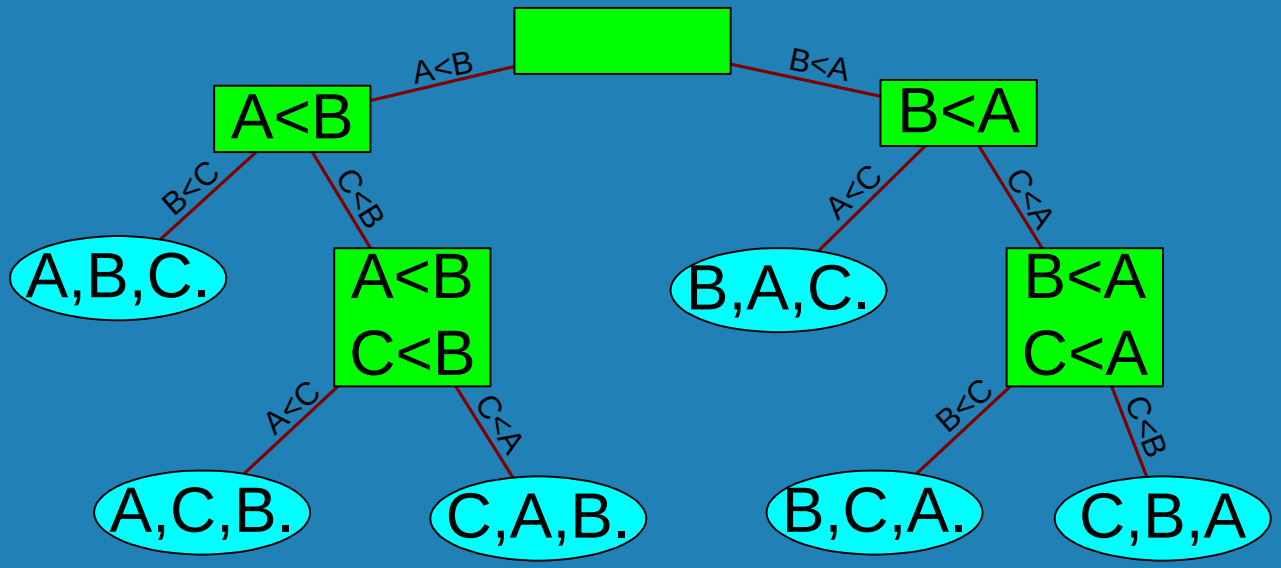
Worst case time



- How many comparisons does it take before we can be sure of the order?
- This is the minimum # of comparisons that any (comparison-based) sorting algorithm has to do!



Decision tree to sort list A,B,C



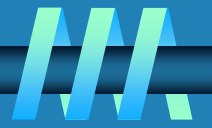
Legend

facts — Internal node, with facts known so far

A,B,C — Leaf node, with ordering of A,B,C

C<A — Edge, with result of one comparison

Max depth of the decision tree



- How many leaves does the tree have?

- What's the shallowest tree with a given number of leaves?
  (Hint: balanced binary trees)

Stirling's approximation

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

(this is where the formula with **$n \cdot \log n - 1.44 \cdot n$** comes from)