# CSE 326: Data Structures

## Topic 12: Comparison-based Sorting

Ashish Sabharwal
Autumn, 2003

## Sorting: *The Big Picture*

Given $n$ `Comparable` elements in an array, sort them in an increasing (or decreasing) order.

| Simple algorithms: $\Theta(n^2)$ | Fancier algorithms: $\Theta(n \log n)$ | Comparison lower bound: $\Omega(n \log n)$ | Specialized algorithms: $\Theta(n)$ | Handling huge data sets |
|---|---|---|---|---|
| Insertion sort<br>Selection sort<br>Bubble sort<br>Shell sort<br>… | Heap sort<br>AVL sort<br>Merge sort<br>Quick sort<br>… | | Bucket sort<br>Radix sort | External sorting |



## Insertion Sort: Idea

- At the $k^{th}$ step, put the $k^{th}$ input element in the correct place among the first $k$ elements
- Result: After the $k^{th}$ step, the first $k$ elements are sorted.

*Runtime:*
worst case :
best case :
average case :



## Selection Sort: idea

- Find the smallest element, put it 1st
- Find the next smallest element, put it 2nd
- Find the next smallest, put it 3rd
- And so on …



## Selection Sort: Code

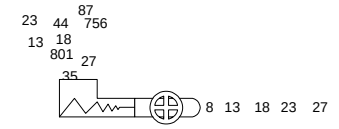
```
void SelectionSort (Array a[0..n-1]) {
      for (i=0, i<n; ++i) {
            j = Find index of smallest entry in a[i..n-1]
            Swap(a[i],a[j])
      }

      while (other people are coding QuickSort/MergeSort)
      {
            Twiddle thumbs
      }
}
```

*Runtime:*
worst case :
best case :
average case :



## HeapSort: Using Priority Q ADT (heap)




Shove all elements into a priority queue, take them out smallest to largest.

*Runtime:*

# AVL Sort

*Runtime:*

**Would the simpler "Splay sort" take any longer than this?**

7

# Merge Sort

```
MergeSort (Array [1..n])
1. Split Array in half
2. Recursively sort each half
3. Merge two halves together
```

*"The 2-pointer method"*

```
Merge (a1[1..n],a2[1..n])
i1=1, i2=1
While (i1<n, i2<n) {
      if (a1[i1] < a2[i2]) {
            Next is a1[i1]
            i1++
      } else {
            Next is a2[i2]
            i2++
      }
}
Now throw in the dregs…
```

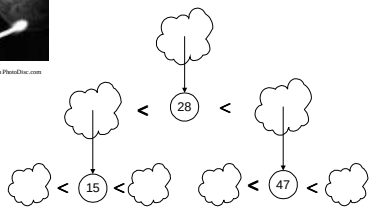
8

# Merge Sort: Complexity

9

Picture from PhotoDisc.com

# Quick Sort




1. Pick a "pivot"
2. Divide into less-than & greater-than pivot
3. Sort each side recursively

10

# Quick Sort Example

| 7 | 2 | 8 | 3 | 5 | 9 | 6 |
|---|---|---|---|---|---|---|

11

# QuickSort: Best case complexity

12

## QuickSort: Worst case complexity




## QuickSort: Average case complexity


Turns out to be $\Theta(n \log n)$

See Section 7.7.5 for an idea of the proof.
*Don't need to know proof details for this course.*



## Dealing with Slow Quick Sorts

- Step 0: Randomly permute given input!!
  - Bad cases more common than simple probability would suggest. So, make it truly random.
- Pick pivot cleverly
  - "Median-of-3" rule: pivot = Median(first, middle, last)
- Pick pivot randomly!

***With good choices, fastest in practice!!***



## Quick Select

What if we want to find the $k^{th}$ smallest element in an array?

Say, $k = n/2$ (i.e., we want to find the median)?



## QuickSelect (Array A, int *k*)

Pick pivot:

| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|
| 7 | 2 | 8 | 3 | 5 | 9 | 6 |

Partition array:

| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|
| 5 | 2 | 6 | 3 | 7 | 9 | 8 |
| | | | | **pindex** | | |

- $k$ = pindex?
- $k$ < pindex?
- $k$ > pindex?

*Runtime:*



## To Do

- Work on Project 3
- Read Chapter 7