# Sorting

Chapter 7 in Weiss



## Today's Outline

- Announcements
- **Sorting**



## Announcements

- Homework #5 – due NOW
- Project #3 –
  - Partner Selection due Tonight
- No Class on Monday Feb 19
- Reading:

  Today & Wednesday: Sorting (Chapter #7)

  Next: Graphs (Chapter #9)



# Sorting

Chapter 7 in Weiss



## Why Sort?



## Sorting: *The* *Big* *Picture*

**Problem**: Given $n$ comparable elements in an array, sort them in an increasing (or decreasing) order.

| Simple algorithms: $O(n^2)$ | Fancier algorithms: $O(n \log n)$ | Comparison lower bound: $\Omega(n \log n)$ | Specialized algorithms: $O(n)$ | Handling huge data sets |
|---|---|---|---|---|
| Insertion sort<br>Selection sort<br>Bubble sort<br>Shell sort<br>… | Heap 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 …



Student Activity

```
Mystery(int array a[]) {
  for (int p = 1; p < length; p++) {
     int tmp = a[p];
     for (int j = p; j > 0 && tmp < a[j-1]; j--)
          a[j] = a[j-1];
     a[j] = tmp;
  }
}
```

**What sort is this?**

**What is its running time?**
**Best?**
**Avg?**
**Worst?**



## 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])
     }
}
```

*Runtime:*

worst case :
best case :
average case :



Student Activity

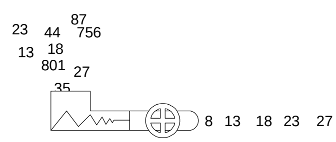
## Sorts using other data structures:

How? Runtime?

- AVL Sort?
- Heap Sort?
- Splay Sort?



## HeapSort: Using Priority Queue 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?**



# Merge Sort?



# 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"*

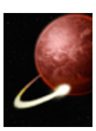
```
Merge (a1[1..n],a2[1..n])
i1=1, i2=1
While (i1<n, i2<n) {
      if (a1[i1] < a2[i2]) {
            Next is a1[i1]
            i1++
      } else {
            Next is a2[i2]
            i2++
      }
}
Now throw in the dregs…
```



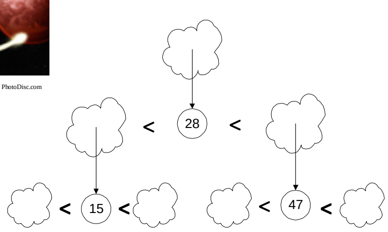
# Merge Sort: Complexity



# Quick Sort



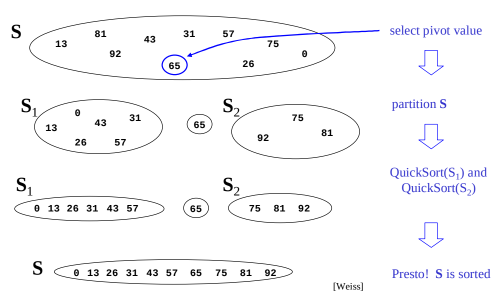
Picture from PhotoDisc.com



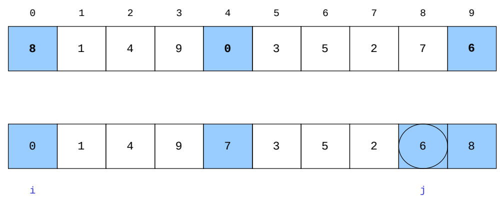

1. Pick a "pivot"
2. Divide into less-than & greater-than pivot
3. Sort each side recursively



# The steps of QuickSort



# QuickSort Example

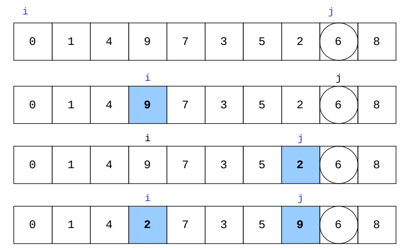



•Choose the pivot as the median of three.

•Place the pivot and the largest at the right
and the smallest at the left



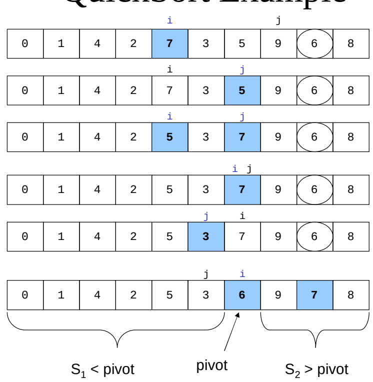
# QuickSort Example




•Move i to the right to be larger than pivot.
•Move j to the left to be smaller than pivot.
•Swap



# QuickSort Example






# Recursive Quicksort

```
Quicksort(A[]: integer array, left,right : integer): {
pivotindex : integer;
if left + CUTOFF ≤ right then
  pivot := median3(A,left,right);
  pivotindex := Partition(A,left,right-1,pivot);
  Quicksort(A, left, pivotindex – 1);
  Quicksort(A, pivotindex + 1, right);
else
  Insertionsort(A,left,right);
}
```

Don't use quicksort for small arrays.
CUTOFF = 10 is reasonable.



Student Activity

# Recurrence Relations

Write the recurrence relation for QuickSort:

- Best Case:

- Worst Case:



# QuickSort: Best case complexity

## QuickSort: Worst case complexity



## QuickSort: Average case complexity

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

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