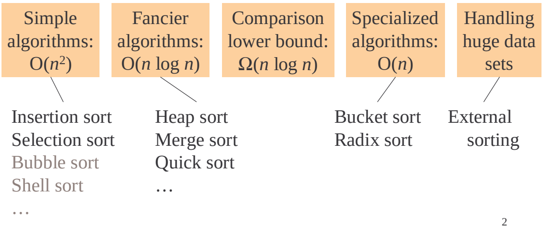
Sorting

Chapter 7 in Weiss

Sorting: *The* Big *Picture*

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 Selection sort Bubble sort Shell sort …	Heap sort Merge sort Quick sort …		Bucket sort 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 …

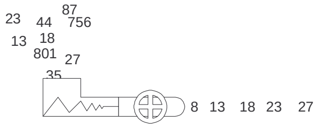
Selection Sort: Code

```
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])
      }

}
```

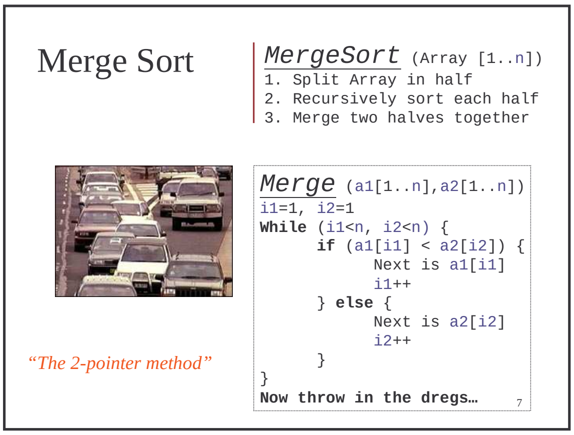
Runtime:
worst case :
best case :
average case :

HeapSort:
Using Priority Queue ADT (heap)



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

Runtime:



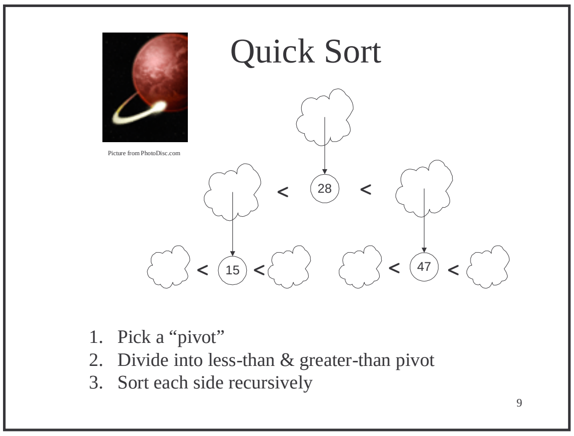
Merge Sort

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

```
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…
```

"The 2-pointer method"

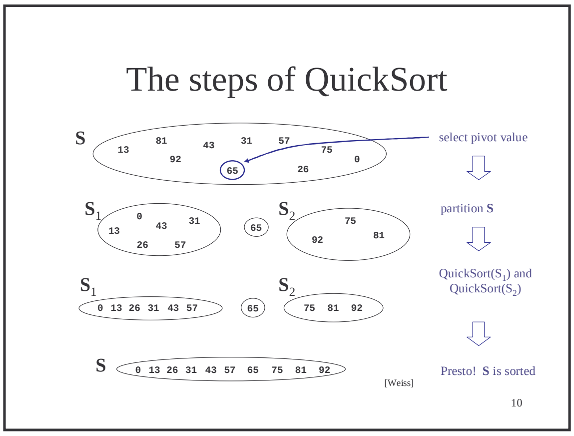
Merge Sort: Complexity



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

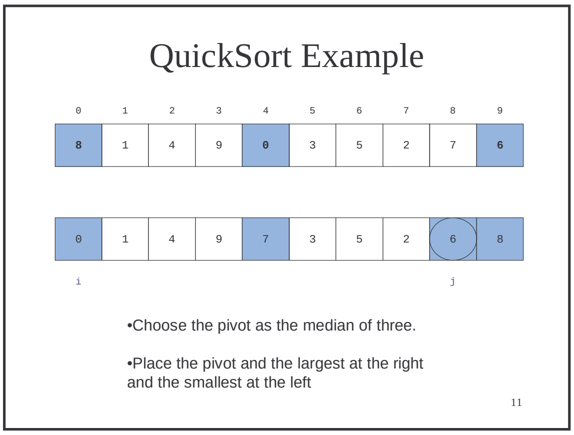
select pivot value

partition **S**

QuickSort(S$_1$) and QuickSort(S$_2$)

Presto! **S** is sorted

[Weiss]



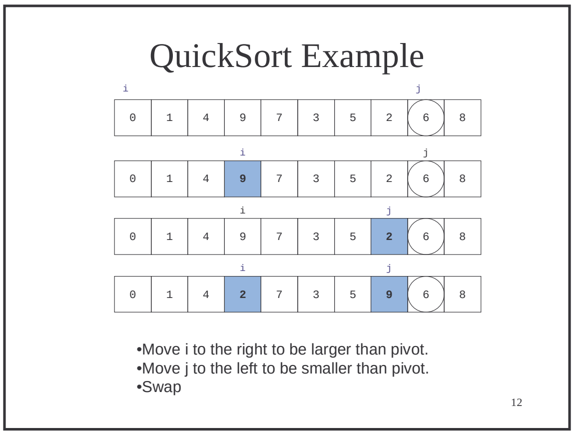
QuickSort Example

0	1	2	3	4	5	6	7	8	9
8	1	4	9	0	3	5	2	7	6

0	1	4	9	7	3	5	2	6	8
i								j	

•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

i								j	
0	1	4	9	7	3	5	2	6	8

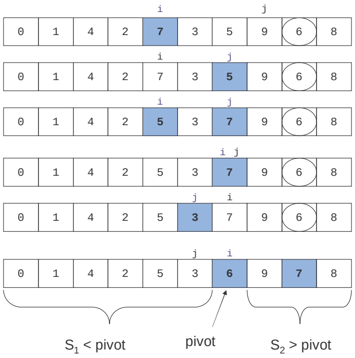
			i					j	
0	1	4	9	7	3	5	2	6	8

			i				j		
0	1	4	9	7	3	5	2	6	8

			i				j		
0	1	4	2	7	3	5	9	6	8

•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.

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.

Features of Sorting Algorithms

- In-place
  - Sorted items occupy the same space as the original items. (No copying required, only O(1) extra space if any.)
- Stable
  - Items in input with the same value end up in the same order as when they began.

Sort Properties

Are the following:	**stable?**		**in-place?**	
Insertion Sort?	No	Yes	Can Be No	Yes
Selection Sort?	No	Yes	Can Be No	Yes
MergeSort?	No	Yes	Can Be No	Yes
QuickSort?	No	Yes	Can Be No	Yes