CSE 326: Data Structures

Sorting

Neva Cherniavsky
Summer 2006

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	Heap sort		Bucket sort	External sorting
Selection sort	Merge sort		Radix sort	
Bubble sort	Quick sort			
Shell sort	…			
…				

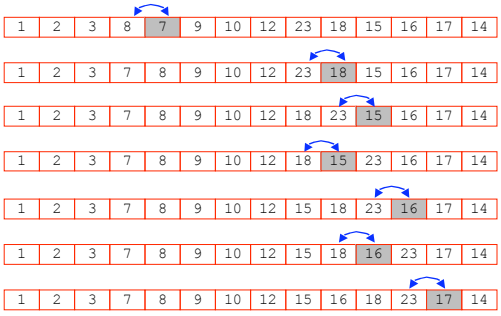
Insertion Sort: Idea

- 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.

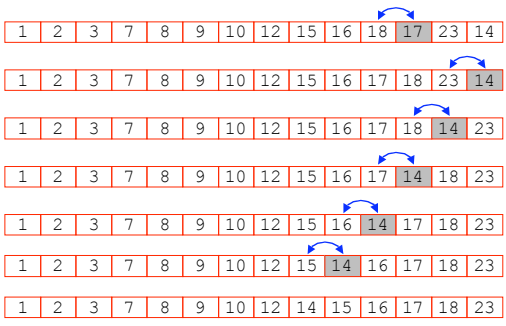
Runtime:

worst case :
best case :
average case :

Example



Example



Recurring Student Activity

Insertion Sort 31 16 54 4 2 17 6

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])
      }

}
```

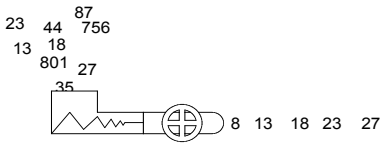
Runtime:

worst case :
best case :
average case :

Recurring Student Activity

Selection Sort 31 16 54 4 2 17 6

HeapSort: Using Priority Queue ADT (heap)



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

Runtime:

Merge Sort

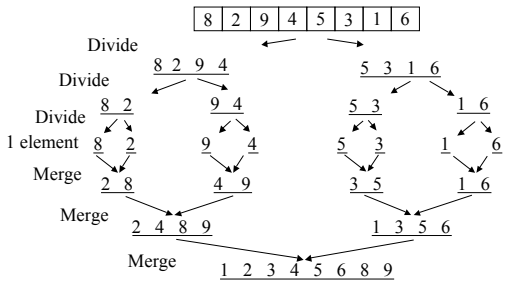
MergeSort (Array [1..n])
1. Split Array in half
2. Recursively sort each half
3. Merge two halves together

"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…
```

Mergesort Example

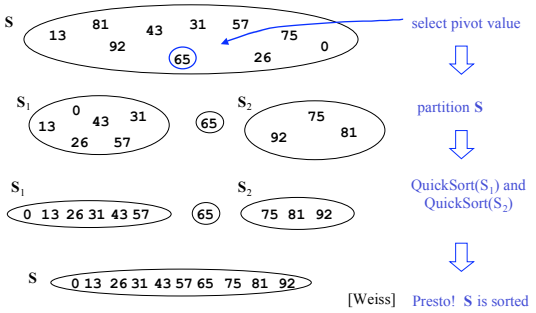


Recurring Student Activity

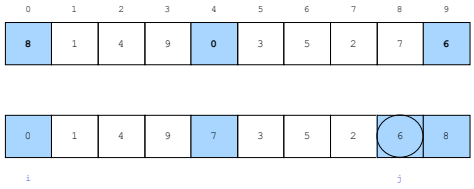
Merge Sort 31 16 54 4 2 17 6

Merge Sort: Complexity

The steps of QuickSort



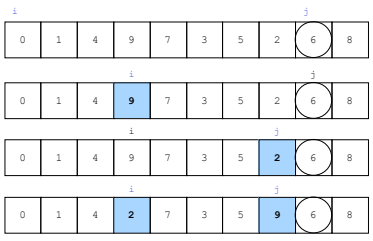
QuickSort Example



•Choose the pivot as the median of three.

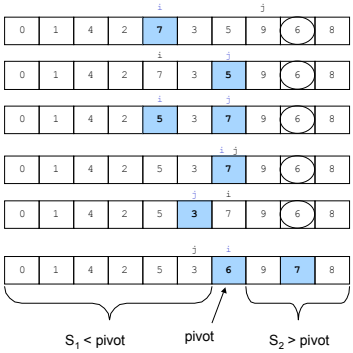
•Place the pivot and the largest at the right and the smallest at the left

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.

Recurring Student Activity

Quick Sort 31 16 54 4 2 17 6

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.