### 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.

<table>
<thead>
<tr>
<th>Simple algorithms: $O(n^2)$</th>
<th>Fancier algorithms: $O(n \log n)$</th>
<th>Comparison lower bound: $\Omega(n \log n)$</th>
<th>Specialized algorithms: $O(n)$</th>
<th>Handling huge data sets</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insertion sort</td>
<td>Heap sort</td>
<td>Bucket sort</td>
<td>Radix sort</td>
<td>External sorting</td>
</tr>
<tr>
<td>Selection sort</td>
<td>Merge sort</td>
<td>Quick sort</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bubble sort</td>
<td>Quick sort</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Shell sort</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 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

```
1 2 3 7 8 9 10 12 15 16 17 23 14
1 2 3 7 8 9 10 12 15 16 17 23 14
1 2 3 7 8 9 10 12 15 16 17 23 14
1 2 3 7 8 9 10 12 15 16 17 23 14
1 2 3 7 8 9 10 12 15 16 17 23 14
1 2 3 7 8 9 10 12 15 16 17 23 14
```
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

```c
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
```

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

Mergesort Example
Recurring Student Activity

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
Recursive Quicksort

Quicksort(A[], left, right): {
    pivotindex : integer;
    if left + CUTOFF ≤ right then
        pivot := median3(A[left, right]);
        pivotindex := Partition(A, left, pivotindex - 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.