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: $O(n^2)$
- Insertion sort
- Selection sort
- Bubble sort
- Shell sort

Fancier algorithms: $O(n \log n)$
- Heap sort
- Merge sort
- Quick sort
- ... 

Comparison lower bound: $\Omega(n \log n)$

Specialized algorithms: $O(n)$
- Bucket sort
- Radix sort

Handling huge data sets
- 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 1\textsuperscript{st}
- Find the next smallest element, put it 2\textsuperscript{nd}
- Find the next smallest, put it 3\textsuperscript{rd}
- 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 :

Sorts using other data structures:

- 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

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…

Merge Sort: Complexity

Quick Sort

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_L) and QuickSort(S_R)

Prefix S is sorted
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.

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: 
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.