Announcements

- Exam 1 is graded & look for mid-semester grades by Friday
- HW due Thursday
- Next HW - due Tues after break
- Lab tomorrow
Exam 1 stats: Average 44.75

55-60: 2
53-54.5: 2
51-52.5: 4
49-50.5: 1
47-48.5: 4
45-46: 4
40-42: 5

Std dev: 7.98

30-39: 4
under 30: 2
Vectors versus lists

Q: What would `[]` look like in a list?

Takes an integer and returns value at that spot:

```
mylist[10]
```

In list, loop through and count up to index.
### Vectors versus lists (cont)

<table>
<thead>
<tr>
<th>Running time</th>
<th>Vectors</th>
<th>Lists</th>
</tr>
</thead>
<tbody>
<tr>
<td>operator []</td>
<td>$O(1)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>find</td>
<td></td>
<td></td>
</tr>
<tr>
<td>insert</td>
<td>$O(n)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>erase/remove</td>
<td>$O(n)$</td>
<td>$O(1)$</td>
</tr>
</tbody>
</table>
Searching

What is linear search? $O(n)$
Go element by element + look for target.
(only makes sense if unordered)

Binary search? $O(\log n)$
Sorted list: look in middle
Reverse on left or right

$B(n) = 1 + B\left(\frac{n}{2}\right) + \cdots + B(1)$
$I: \ \frac{n}{2^d} = 1 \Rightarrow \ n = 2^d \Rightarrow d = \log_2 n$
Practical Considerations

Which is better?

Bin. search needs $O(1)$ - operator?!
Sorting

Name some sorting algorithms.

- Bubble sort
- Quick sort
- Insertion sort
- Merge sort
Insertion Sort

maintain sorted sublist
& insert next value in place

\[ 2 \rightarrow 1 \rightarrow 4 \rightarrow 1 \rightarrow 2 \rightarrow 1 \]

sorted first where this goes

Time: \[ \sum_{i=1}^{n} O(i) = O(n^2) \]
A be a list

for (i = 1 to n)
  Find where A[i] goes in Sublist A[0..i-1]
  Say j is result of
  Insert A[i] in position j

O(i) operation each time in either list or vector
Bubble Sort

\[ \text{for } j = n-2 \text{ down to } 1 \]
\[ \text{for } i = 0 \text{ to } n-j \]
\[ \text{compare } A[i] \text{ and } A[i-1] \]
\[ \text{swap if out of order} \]

\( \Theta(n^2) \)
Merge Sort

Base Case

7 3 1 6 2 5 4 8

Merge: recursively sort left and right halves

1 comparison + increment & 1 more element in sorted list

new: 1 2 3 4 5 6 7 8

Merge subroutine: O(n) time
\[ M(n) = 2M\left( \frac{n}{2} \right) + O(n) \]
\[ M(1) = O(1) \]
\[ M(n) = O(n \log n) \]
\[ M(n) = 2M\left(\frac{n}{2}\right) + O(n) \]

\[
M(n) = \sum_{i=0}^{\log_2 n} \frac{n}{2^i}
\]

Level \(i\):
- \(2^i\) nodes.
Quick Sort

1. Pick pivot element

2. Divide around pivot

3. 1 4 2 5 6 7 8

4. Recurse at end of O(n) comparisons, pivot is in correct spot
Quicksort: worst-case choose an awful pivot (1 or 8) every time.
\[ O(n^2) \] worst

Expected running time:
(with "high probability")
\[ O(n \log n) \]
**Bucket Sort**

$n$ elements, each between $0$ and $N-1$

Can we do better than $O(n \log n)$?

$0 \ 1 \ 2 \ \cdots \ N-1$

$O(n+N)$-time - just allocate $N$ buckets and loop through list
SppS 10 things:

1. 10,000 /1,000,000, 7, 6
2. -50, ...

need over 1,000,000 buckets to sort 10 things
Radix Sort: for multiple-key sorting

Ex: (1, 5), (2, 1), (4, 2), (3, 3), (5, 4),
(3, 1), (2, 2), (5, 1), (2, 4)

Sort lexicographically: (use repeated bucket sorts)
Practicality

Experimentally, quicksort runs faster than merge on small inputs.

Why? No need for allocating new arrays.
More practicalities

- If implemented well, the running time of insertion sort is $O(m+n)$, where $m$ = # of inversions
  (or out of order elements)

5 2 1 3 inversions
Conclusion: It depends!

- If the range of values is small, bucket sort (or radix sort) are faster.

- Quick sort, despite worst-case \( O(n^2) \) is actually the one usually implemented.

- Also, can depend on linked vs. array based structure.