Midterm March 10: practice test online along with answers.

**A. Sorting**

Many sorting algorithms have a time complexity of

| T(N) = N^2 | insertion sort, bubble sort, selection sort, quicksort, many others |
| T(N) = N \times \log_2(N) | merge sort, radix sort, heapsort, and some others |
| T(N) = N | only for some parallel sort, like POETS (requires lots of hardware!) |

```python
def selectionSort(mylist):
    for i in range(len(mylist) - 1):
        min_index = i
        for j in range(i+1, len(mylist)):
            if mylist[j] < mylist[min_index]:
                min_index = j
        if i != min_index:
            mylist[i], mylist[min_index] = mylist[min_index], mylist[i]
```

```python
def selectionSort2(mylist):
    for i in range(len(mylist) - 1):
        for j in range(i+1, len(mylist)):
            if mylist[i] > mylist[j]:
                mylist[i], mylist[j] = mylist[j], mylist[i]
```

```python
def bubblesort(mylist):
    for i in range(len(mylist) - 1):
        for j in range(len(mylist) - 1 - i):
            if mylist[j] > mylist[j+1]:
                mylist[j], mylist[j+1] = mylist[j+1], mylist[j]
```

There are so many different sorting algorithms! See:


\[
\frac{N(N-1)}{2} = \frac{N^2 - N}{2} = \frac{N^2}{2} - \frac{N}{2}
\]
B. Quick sort (a different partition algorithm)

The heart of Quicksort is the partition algorithm. Many different ways of finding the right pivot:

1. Select a random value position in the list
2. Select the value at a given place (the midpoint or position 0 are popular choices)
3. Pick 3 values ([0], [mid], [-1]) and take the middle value

In general you do NOT want the pivot value to be the largest or the smallest value! You want it to be in the middle of the final list of values. But how do you pick that unless you know it already?

```java
function partition(L, start, end)
    left = start
    right = end
    pivot = L[start]
    while left < right
        # move left pointer rightwards until it equals or exceeds the pivot
        while L[left] <= pivot
            left++
        # now move the right pointer leftwards until it equals the pivot or is less than
        while L[right] > pivot
            right--
        if left < right
            swap the left and right elements of L
            L[low] = L[right]
            L[right] = pivot
    return right
```

This one is a little easier to understand than the book's which uses the middle point always, because the one above leaves the pivot as the first cell always, and only at the end swaps it with the "right" value which will be the beginning of the upper half.

```
in-place sorting
```

```
0 1 len-1
```
merge sort

$N \times \log_2 N$

CON: 2N memory cells
C. Stable sorting

When the sorting uses only part of each record as the sortkey, i.e. when your record is multi-field, then you have the option of sorting on fields within fields.

<table>
<thead>
<tr>
<th>name</th>
<th>age</th>
<th>pet</th>
<th>whereBorn</th>
<th>faveAuthor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alex</td>
<td>59</td>
<td>squirrels</td>
<td>Valentine</td>
<td>the Bible</td>
</tr>
<tr>
<td>Joseph</td>
<td>37</td>
<td>dogs</td>
<td>Buffalo</td>
<td>Dan Brown</td>
</tr>
<tr>
<td>Kathy</td>
<td>26</td>
<td>orcas</td>
<td>Seattle</td>
<td>Togetherness is our home</td>
</tr>
<tr>
<td>Kathy</td>
<td>49</td>
<td>dogs</td>
<td>Greeley</td>
<td>C. S. Lewis, J.D. Salinger</td>
</tr>
<tr>
<td>Kristine</td>
<td>39</td>
<td>dogs</td>
<td>Lincoln</td>
<td>Wendell Barry</td>
</tr>
<tr>
<td>Mark</td>
<td>59</td>
<td>cats</td>
<td>Valentine</td>
<td>Tolkien, Dawkins</td>
</tr>
<tr>
<td>Sally</td>
<td>79</td>
<td>cats</td>
<td>Lincoln</td>
<td>Danielle Steele, E. L. James</td>
</tr>
<tr>
<td>Steffanie</td>
<td>26</td>
<td>iguanas</td>
<td>Buffalo</td>
<td>Danielle Steele</td>
</tr>
<tr>
<td>Zoyla</td>
<td>59</td>
<td>rats</td>
<td>Moscow</td>
<td>Tolstoy</td>
</tr>
</tbody>
</table>

Many sorting algorithms are not stable, which means they move things around however they feel like it as they are working.

If two items compare as equal, then their relative order in the new dataset will be identical to their relative order in the original dataset.

<table>
<thead>
<tr>
<th>Sorted by age already</th>
<th>then sort by name (stable)</th>
<th>then sort by name (unstable)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zach,8</td>
<td>Betty,26</td>
<td>Betty,26</td>
</tr>
<tr>
<td>Tom,8</td>
<td>Mike,15</td>
<td>Mike,15</td>
</tr>
<tr>
<td>Mike,15</td>
<td>Kathy,26</td>
<td>Kathy,49</td>
</tr>
<tr>
<td>Kathy,26</td>
<td>Kathy,49</td>
<td>Kathy,26</td>
</tr>
<tr>
<td>Betty,26</td>
<td>Tom,8</td>
<td>Tom,8</td>
</tr>
<tr>
<td>Kathy,49</td>
<td>Zach,8</td>
<td>Zach,8</td>
</tr>
</tbody>
</table>

We wanted it sorted by name then age within name.

Many examples of multi-field sorting in everyday life.

We would like a list of all courses, sorted by department, then course number, then section letter.

We would like a list of the populations of states (most people to least people) then alphabetized within that list.

We would like a list of all books in the library by genre, then title, then author. (e.g. there might be 2 different horror novels: “Nightmare on Elm Street” by John Doe and “Nightmare on Elm Street” by Mark Meyer).

In general sort on the most minor field first, then the next most minor field, etc. But use a stable sorting algorithm. E.g. insertion sort, merge sort, bubble sort.
One trick to making it stable is to form an artificial key by concatenating all the values of the major to minor sort fields and then comparing that.

But then you run into an interesting problem.

Windows sorts its filenames much differently than Unix!

<table>
<thead>
<tr>
<th>Unix (literal)</th>
<th>Windows (smart)</th>
</tr>
</thead>
<tbody>
<tr>
<td>day 1 notes.docx</td>
<td>day 1 notes.docx</td>
</tr>
<tr>
<td>day 10 notes.docx</td>
<td>day 2 notes.docx</td>
</tr>
<tr>
<td>day 11 notes.docx</td>
<td>day 3 notes.docx</td>
</tr>
<tr>
<td>day 12 notes.docx</td>
<td>day 4 notes.docx</td>
</tr>
<tr>
<td>day 2 notes.docx</td>
<td>day 5 notes.docx</td>
</tr>
<tr>
<td>day 3 notes.docx</td>
<td>day 6 notes.docx</td>
</tr>
<tr>
<td>day 4 notes.docx</td>
<td>day 7 notes.docx</td>
</tr>
<tr>
<td>day 5 notes.docx</td>
<td>day 8 notes.docx</td>
</tr>
<tr>
<td>day 6 notes.docx</td>
<td>day 9 notes.docx</td>
</tr>
<tr>
<td>day 7 notes.docx</td>
<td>day 10 notes.docx</td>
</tr>
<tr>
<td>day 8 notes.docx</td>
<td>day 11 notes.docx</td>
</tr>
<tr>
<td>day 9 notes.docx</td>
<td>day 12 notes.docx</td>
</tr>
</tbody>
</table>

Note that if you take the strings, then Unix is correct by treating numbers are “letters”.

For the same reason that Jo comes before Joseph: Jo is shorter.

Jo
Joseph

But if you followed that by a field like # then age, you might throw it off.

Jo#12
Joseph#36

How does “#” compare to “s”? “#”=35, “s”=115, so we are safe, # is less than any letter!

The only real way to be sure to pad:

Jo  #12
Joseph#36

Also do this with numbers: day 01 notes, day 02 notes, ... day 09 notes, day 10 notes, etc. This is a pain to remember and is kind of unnatural but it works and is used a lot.
D. Practice test Review

Questions?