CSC207.01 2014S, Class 27: Merge Sort

Overview

- Preliminaries.
  - Admin.
  - Questions.
- Lower bounds on sorting.
- Divide and conquer algorithms.
- An introduction to merge sort.
- Analyzing merge sort.

Preliminaries

Upcoming work

- Homework 5 due tonight!
  - You can send email during class and I'll respond as quickly as I can.
  - That should give you an opoportunity to get some cyclic feedback.
- Today's lab writeup: Invariants for merge (part of Exercise 2a)
  - You can draw pictures on the computer
  - You can draw pictures on paper
  - You can write things a bit more mathematically
- Reading for Friday: Quicksort

Admin

- Continue to have fun with Earnest!
- KS is the note taker today.
- Earnest is happy to answer questions about Skip Lists, whether he knows it or not.
  - I'll also try to be on email, and you can collaborate on sending me messages.
- Extra credit:
  - Convocation, noon, today.
  - Presentations on Grinnell institutional image, noon on Thursday or Friday.
- Other things you should do (warning! tickets go quickly)
  - Neverland players.
  - Balancing acts.

Questions on the homework

Sorting Algorithms

- What makes a sorting algorithm stable?
    - If a is before b in the original array and a = b, then a is before b is the final array
    - Is selection sort stable?
        - It depends on `indexOfSmallest`
- If we update `small` when `order.compare(vals[i], vals[small]) < 0` then it is stable
- If we update `small` when `order.compare(vals[i], vals[small]) <= 0` then it is NOT stable

Merge Sort

- Take an array, divide it in half and keep dividing it in half until we have only pairs. Sort the pairs and go back up the tree, merging the parts as you go.
    - To merge check which of the first two elements (of the parts you are merging) comes first until one part has no remaining elements

Break down the array:

```
7 3 2 8 1 5 6 4

7 3 2 8   1 5 6 4

7 3   2 8   1 5   6 4
```

Build it back up:

```
3 7   2 8   1 5   4 6

2 3 7 8   1 4 5 6

1 2 3 4 5 6 7 8
```

[Sam notes that recursion doesn’t quite work this way. We do the left subarray completely before we do the right subarray.]

Skip Lists

- Functions that might be useful:
    - `contains`
    - `add`
        - Consider a singly linked list with a height cap
        - When you add an element, you need to make it point to what all the preceding pointers were pointing to before it was inserted. Then make the preceding pointers point to your new element.

    - `remove`
    - `findPlace`
- Skip list element object (SkipLO)
    - Useful fields:
        - `int level;` [Sam notes that you can get this with `next.length`.]
        - `String value;`
        - `SkipLO[] nexts;`
    - Constructor:
        - `Constructor (int l, String str)` [Sam notes there's an interesting design decision: Does the list class deside on the level, or does the node constructor decide upon the level?]
            - `level = l;` [If the node computes the level, this would be `computeLevel()`]
            - `value = str;`
            - `nexts = new SkipLO[level];`
    - Get fields
        - obj.level
        - obj.value
        - obj.next[l]
    - You need to keep track of what comes before the element you are interested in because they're singly linked and you can't just go back.

Do the lab!