CSC207.01 2013F, Class 32: Merge Sort

Overview

- Preliminaries.
  - Admin.
  - A C problem
  - Ushahidi and the project
  - Questions on HW 7
  - Questions on Exam makeup
  - HW 8
- An introduction to merge sort.
- Analyzing merge sort.
- Lab.

Admin

- While you're waiting, fork and clone https://github.com/Grinnell-CSC207/sorting
- Today we will do a quick analysis of merge sort and then follow it up with some lab exercises.
- I'm moving the due time for the electronic and printed versions of the exam to 10:30 pm on Friday night. Put the printed version under my door.
- Upcoming extra credit opportunities:
  - Study in Budapest Lunch, Today
  - Learning from Alumni, Thursday: Jordan Shkolnick '11 (Microsoft)
  - CS Table, Friday: Ambient Belonging
  - One Grinnell Prize Event next week

A C Problem

```
char *
foo(char *t, char *s)
{
  while (*t++ = *s++)
    ;
  return t;
} // for
```

Ushahidi and the Project

- We should discuss the project and the role of Ushahidi in this class. Clearly, we were less successful at getting the materials ready than we would have liked this summer, and I was as unsuccessful at getting them ready during the semester.
- I've removed Ushahidi from HW 8.
- Project summary:

    - Find a client; negotiate the design of an Ushahidi installation
    - Build an Ushahidi installation for that client
    - Write some custom report or tool for the client
- Do you still want to do the project? Revisit Friday.

HW 7

Due tonight

Why implement java.util.Iterator when we have cursors?

Real Java programmers build iterators for any collection class they design

```
public class MyIterator<T> implements java.util.Iterator<T> {
     // +--------+------------------------------------------------------------
     // | Fields |
     // +--------+

    Node<T> pos;

     // +--------------+-----------------------------------------------------
     // | Constructors |
     // +--------------+

     // +---------+-----------------------------------------------------------
     // | Methods |
     // +---------+

    public T next() {
         // Find out what's right after pos
         // Advance the position
         // Return the value we got in step 1
    }
    public boolean hasNext() {
    }

    public void remove() {
         throw new UnsupportedOperationException();
    }
}
```

Once you've implemented iterators, folks can write

```
DoublyLinkedList dll;
for (val : dll) {
}
```

_When should I put the type variable in brackets?

Usually, whenever you are referring to a generic/parameterized class.

Not when you are using it as a type

So

```
public T extractValue(Node<T> node)
```

Also when parameterizing static methods

```
public static <T> returnType methodName(...)
```

Exam

Any hints on DNF?

Write `isDNF`

HW 8

- Implement five different sorting methods;
- Do other stuff

An introduction to merge sort

- Two sorting algorithms, both O(n^2)
- Can I do better?
  - Practical: Look for other algorithms
  - Theoretical: Does one exist: A compare/swap sorting requires O(nlogn) steps
- There are O(nlogn) sorting algorithms based on compare/swap
  - Merge sort
  - Quicksort
  - Heap sort
- One key approach to speeding up algorithms: Divide and conquer
- Divide the array in half
- Sort each half
- Merge the two halves:
  - Create a new array
  - Repeateldy grab the smallest remaining thing from each array and copy to the new arraya - O(1) steps

Analyzing merge sort

- How do we figure out how fast this is? Recurrence relations!
- Write a function that describes the running time of our algorithm on input of size n
- t(n) = t(n/2) + t(n/2) + n
- t(n) = 2*t(n/2) + n

- How do we figure this out?
- Base case: t(1) = 1
- Build up
  - t(2) = 2*t(2/2) + 2 = 2*t(1) + 2 = 2*1 + 2 = 4*
  - *t(4) = 2*t(4/2) + 4 = 2*t(2) + 4 + 2*4 + 4 = 12
  - t(8) = 2*12 + 8 = 32*
  - *t(16) = 2*32 + 16 = 80
- Build down
  - t(n) = 2*t(n/2) + n // Note that t(n/2) = 2*t(n/4) + n/2
  - t(n) = 2*(2*t(n/4) + n/2) + n // Simplify
  - t(n) = 4*t(n/4) + 2n // Note that t(n/4) = 2*t(n/8) + n/4
  - t(n) = 4*(2*t(n/8) + n/4) + 2n // Simplify
  - t(n) = 8*t(n/8) + 3n // Note that t(n/8) = 2*t(n/16) + n/8
  - t(n) = 8*(2*t(n/16) + n/8) + 3n // Simplify
  - t(n) = 16t(n/16) + 4n
  - t(n) = (2^4)t(n/(2^4)) + 4n
  - t(n) = (2^x)t(n/(2^x)) + xn
  - Choose k s.t. 2^k = n
  - t(n) = nt(n/n) + kn
  - t(n) = nt(1) + kn
  - t(n) = n + kn
  - If 2^k = n, then k = log*2(n)*
  - *t(n) = n + n*log*2(n)
  - t(n) is in O(nlogn)

*

Lab