More Sorting

Homework, please!

Heaps and Heapsort

Quicksort

Linear-time sorting: Counting, Radix
Notes on Assignment 2, Q1

Merge Sort Redux

merge a:atail b:btail =
  if a <= b then a : merge atail b:btail
  else b : merge a:atail btail

merge a [] = a
merge [] b = b
merge [] [] = []

sort alist =
  let q = firsthalf alist
    r = secondhalf alist in
  merge (sort q) (sort r)
Appendable Linked Ordered Lists

```java
public class ALOList
    extends LinkedListOrderedList {
    protected Cons tail;
    public void append(ALOList a) {
        tail.rest = a.head; tail = a.tail;
        len+=a.len; }
    public ALOList rest() {
        ALOList r = new ALOList(); r.head =
            head.rest;
        r.len = len-1; r.tail=tail; return r; }
    public void setItem(int i, Comparable v) {
        if (i == len) {
            tail.rest = new Cons();
            tail.rest.item = v; tail = tail.rest;
            len++; }
```
else { super.setItem(i, v); } } };
class Merge {
    ALOList
    merge(ALOList a, ALOList b) {
        ALOList r = new ALOList();
        if (a.length() < 1) return b;
        if (b.length() < 1) return a;
        if (a.getItem(0).compareTo(b.getItem(0)) <= 0) {
            r.setItem(0, a.getItem(0));
            r.append(merge(a.rest(), b));
        } else {
            r.setItem(0, b.getItem(0));
            r.append(merge(a, b.rest()));
        }
        return r; }
    }
}
Notes on Assignment 2, Q2

Accessing successor:

```java
int cachedIndex; Cons cachedPred;
// in getItem, setItem....
if (i == cachedIndex) return cachedPred.item;
if (i == cachedIndex+1) {
    cachedPred = cachedPred.rest;
    cachedIndex = i;
    return cachedPred.item;
}
```

An elaboration:

```java
if (i > cachedIndex) {
    i -= cachedIndex;
    // now count forward from cachedPred....
}
```

And insert needs to increment cachedIndex if you insert before the index...
O(log n) Access with O(1) Append

Balanced tree

Adding an item to a balanced tree is O(log n)

Append is no longer O(1)

Remember: Items are not sorted, only indices

So think about O(n) sorting from today’s reading
Build a “radix list”

Linked list of arrays, each wide

Array points to nodes in list

Step down list by position of high order bit

Index into array based on lower-order bits

  Append sets tail as before, but also sets array
  If need a new layer, allocate a block of memory
Heaps

Definition: Empty or a value with two subheaps

S.t. value is >= (or <=, depending) values in subheaps

\[
\text{data Heap } a = E \mid H \text{ Int } a \ (\text{Heap } a) \ (\text{Heap } a)
\]

\[
\text{rank } E = 0
\]
\[
\text{rank } (H \ r \ _ \ _ \ _) = r
\]

\[
\text{makeH } x \ a \ b = \text{if rank } a >= \text{ rank } b \\
\quad \text{then } H \ (\text{rank } b+1) \ x \ a \ b \\
\quad \text{else } H \ (\text{rank } a+1) \ x \ b \ a
\]

\[
\text{merge } h \ E = h
\]
\[
\text{merge } E \ h = h
\]
\[
\text{merge } h1@(H \ _ \ x \ a1 \ b1) \ h2@(H \ _ \ y \ a2 \ b2) = \\
\quad \text{if } x <= y \text{ then makeH } x \ a1 \ (\text{merge } b1 \ h2) \\
\quad \text{else makeH } y \ a2 \ (\text{merge } h1 \ b2)
\]
Using Heaps to Sort

Find minimum, remove from heap and append to new list

First, find and or remove minimum

findMin E = error "empty heap"
findMin (H _ x _ _) = x

removeMin E = error "empty heap"
removeMin (H _ x a b) = merge a b

Build a heap from a list by making a 1-item heap and merging it with a heap of the rest of the list

heapify [] = E
heapify a:rest =
    merge (makeH a E E) (heapify rest)
Mergesort: The Meat

Make a list from a heap by getting the top item; then appending the list of heap resulting from the merge of the two subheaps

\[ \text{listHeapInOrder } E = [] \]
\[ \text{listHeapInOrder } (T \ _ \ x \ a \ b) = \]
\[ x : \text{listHeapInOrder } (\text{merge } a \ b) \]

Sorting is heapifying the list, then listing it in order

\[ \text{heapsort } [] = [] \]
\[ \text{heapsort } \text{sortme } = \]
\[ \text{case } (\text{heapify } \text{sortme}) \text{ of} \]
\[ E \quad \rightarrow \ [] \]
\[ H \ _ \ x \ a \ b \rightarrow x : \text{listHeapInOrder } (\text{merge } a \ b) \]
Analysis of Merge Sort

merge is \( O(\log n) \)

heapify is \( n \) merges, or \( n \ O (\log n) \) or \( O(n \log n) \)

listHeapInOrder is \( n \) merges or \( n \ O(\log n) \) or \( O(n \log n) \)

```haskell
-- heapify a:rest =
--    merge (makeH a E E) (heapify rest)
-- listHeapInOrder (T _ x a b) =
--    x : listHeapInOrder (merge a b)

heapsort' [] = []
heapsort' a:rest =
    listHeapInOrder (merge (makeH a E E)
        heapify rest)
```
Quicksort

Not optimal but performs well in practice

Except for small list sizes, where bubble sort or insertion sort is faster

Choose a partition element in a list such that:

Everything before partition is less than partition’s val

Everything after the partition is greater
QuickSort, the Meat

quickSort [] = []
quickSort [a] = [a]
quickSort [a b] = if a <= b then [a b] else [b a]
quickSort q@b:c:rest =
  let (front, p:back) = splitAt (length q/2) q in
  let frontAndBack = front ++ back in
  (quickSort
    [ lt | lt <- frontAndBack, lt <= p ])
  ++ [p] ++
  (quickSort
    [ gt | gt <- frontAndBack, gt > p ])
Comparison Sorts

Have used interface Comparable up til now

Gives us < 2 bits of info on values

greater, lesser, equal

could do it with just greater or lesser, i.e., 1 bit

Intuitive “proof”: Binary rep of n has \( \log_2 n \) digits

N items, \( \log n \) information on order for each
Improvements from Domain Knowledge

Assuming a maximum size for numbers, log n aspect of arithmetic goes away

Using numbers, we can sort in O(n) time
Counting Sort

Given numbers $\geq 0$ and $< k$, count how many of each value there are — $O(n)$

Write out the array

```java
int[] sort(int[] sortme, int k) {
    int[] count = new int[k];
    int[] r = new int[sortme.length];
    int i, j, q;
    for (i=0; i<sortme.length; i++)
        count[sortme[i]]++;
    for (i=0, q=0; i<k; i++)
        for (j=0; j<count[i]; j++) {
            r[q]=i; q++;
        }
    return r;
}
```
Radix Sort

For each digit, sort by digit

Low-order first, then higher order on successive passes

Fixed number of values for each digit

Fixed number of digits
Next Week

Reading: Chapters 10, 12, and 18; skim 13; in CLRS

Topic is balanced trees

We will do 2-3-4 trees, not red-black trees

Some authors (e.g., Sedgwick) consider 2-3-4 trees impractical to implement, though easier to understand

Au contraire, mon frère!

Assignment 3 is on the web