CS61B Fall 2015 Guerrilla Section 3 Worksheet

8 November 2015

Directions: In groups of 4-5, work on the following exercises. Do not proceed to the next exercise until everyone in your group has the answer and *understands why the answer is what it is*. Of course, a topic appearing on this worksheet does not imply that the topic will appear on the midterm, nor does a topic not appearing on this worksheet imply that the topic will not appear on the midterm.

1 Which Sort to Use

For each of the following scenarios, choose the best sort to use and explain your reasoning.

(a) The list you have to sort was created by taking a sorted list and swapping N pairs of adjacent elements.

(b) The list you have to sort is the list of everyone who took the US Census and you want to sort based on last name.

(c) You have to sort a list on a machine where swapping two elements is much more costly than comparing two elements (and you want to do the sort in place).

(d) Your list is so large that not all of the data can fit into RAM at once. As in, at any given time most of the list must be stored in the external memory, where accessing it is much slower.

STOP!

Don't proceed until everyone in your group has finished and understands all exercises in this section!

2 Balanced Search Tree Mechanics

(a) Draw the result of inserting the following numbers into a 2-4 tree (in the order that they are listed): 1, 3, 5, 7, 2, 4, 8, 9, 10, 0, -1. For the purposes of this exercise, when a node becomes overfull, promote the second element from the left.

(b) Now draw the result of removing the following numbers from the last 2-4 tree from above: -1, 4, 0.

(c) Draw a red-black tree that corresponds to the last 2-4 tree you drew in part (a).

STOP!

Don't proceed until everyone in your group has finished and understands all exercises in this section!

3 Identify the Sort

Match up the sorting algorithms I–VI to the sequences a–f, which represent an array being sorted at some intermediate steps in the computation (not necessarily consecutive or evenly spaced). In each case, the original array to be sorted consists of the integers

```
5103 9914 0608 3715 6035 2261 9797 7188 1163 4411
```

```
                 Intermediate Steps                        Algorithm
a. 2261 4411 5103 1163 9914 3715 6035 9797 0608 7188
   6035 5103 1163 7188 2261 4411 0608 3715 9797 9914     ______________

b. 5103 9914 0608 3715 2261 6035 7188 9797 1163 4411
   0608 2261 3715 5103 6035 7188 9797 9914 1163 4411     ______________

c. 0608 1163 2261 3715 4411 5103 6035 7188 9914 9797
   0608 1163 2261 3715 4411 5103 6035 7188 9797 9914     ______________

d. 0608 1163 5103 3715 6035 2261 9797 7188 9914 4411
   0608 1163 2261 3715 6035 5103 9797 7188 9914 4411     ______________

e. 9797 7188 5103 4411 6035 2261 0608 3715 1163 9914
   4411 3715 2261 0608 1163 5103 6035 7188 9797 9914     ______________

f. 5103 0608 3715 2261 1163 4411 6035 9914 9797 7188
   0608 2261 1163 3715 5103 4411 6035 9914 9797 7188     ______________
```

I. Heap sort

II. Quicksort

III. Merge sort

IV. Selection sort

V. LSD radix sort

VI. MSD radix sort

STOP!

Don't proceed until everyone in your group has finished and understands all exercises in this section!

4 Trie Mechanics

In this exercise we will implement the insert method for a Trie. Specifically, we are assuming that all strings are over an alphabet of the four characters 'A', 'C', 'T', and 'G' and that the children of each node in the trie can thus be stored in an array of length 4. Fill in the insert methods below in accordance with the comments.

```java
public class TrieNode {

    /** Initializes the character of this TrieNode to VAL and the
     *  child pointer array to be an empty array of size 4. */
    public TrieNode(char val) {
        this.val = val;
        isWord = false;
        children = new TrieNode[4];
    }

    /** Inserts S into this Trie assuming that this node is the root.
     *  Returns true iff S was already in the Trie. This method assumes
     *  that s contains only the characters A, C, T, and G. */
    public boolean insert(String s){


    }

    /** Inserts S into this Trie assuming that this node is an internal
     *  node of depth i + 1 in the Trie (where the root is at depth 0).
     *  Returns true iff S was already in the Trie. */
    public boolean insert(String s, int i) {


    }

    /** Returns the child of this node corresponding to the character NEXT. */
    public TrieNode getChild(char next) {
        return children[getIndex(next)];
    }

```

```java
    /** Returns the index in the child pointer array corresponding to the
     *  character A. */
    private static int getIndex(char a) {
        switch (a) {
        case 'A':
            return 0;
        case 'C':
            return 1;
        case 'T':
            return 2;
        case 'G':
            return 3;
        default:
            return -1;
        }
    }

    /** The children of this node. */
    private TrieNode[] children;
    /** The character on the edge leading to this node. */
    private char val;
    /** True if and only if this node represents a complete word in the Trie. */
    private boolean isWord;
}
```

STOP!

Don't proceed until everyone in your group has finished and understands all exercises in this section!

5 Merging Many Sorted Lists (CS61B Fall 2013 Midterm 2)

Suppose that we have an array of `Iterator<String>` objects, where each `Iterator` is guaranteed to deliver its Strings in sorted order. We would like to form a single sorted list containing all items produced by these iterators. Describe how to do this efficiently. You do not need to give actual Java code, but be precise and clear in your description. Give the running time of your algorithm in terms of k, the number of iterators, and N, the total number of strings in the final list.

STOP!

Don't proceed until everyone in your group has finished and understands all exercises in this section!

6 Local Extrema

Given an array of integers, define a local maximum to be an element of the array that is larger than the adjacent integers in the array. A local minimum is defined similarly. Given an array of N integers, devise an $O(N \log(N))$ algorithm to rearrange the array so that every element is either a local maximum or a local minimum.

7 Partial Matches

Given a list of N input words all of length at most k and M query words, we would like to find, for each query word, the number of input words that match the first $k/2$ letters of the query word. Describe an algorithm that accomplishes this and give its running time as a function of N, M, and k.

STOP!

Don't proceed until everyone in your group has finished and understands all exercises in this section!

8 Analyze mergeAll (CS61BL Summer 2014 Midterm 2)

We would like to write a method called `mergeAll` that merges together N sorted lists with M elements each. To do so, we will use a method called `merge` that returns the result of merging two sorted linked lists. If one of the lists has length l and the other has length r then the merge method will run in $O(l + r)$ time. Below are two versions of the `mergeAll` method. For each one, give the running time in terms of N and M.

```
public static LinkedList<Integer> mergeAll(ArrayList<LinkedList<Integer>> lists) {
    for (int i = 1; i < lists.size(); i++) {
        lists.set(0, merge(lists.get(0), lists.get(i)));
    }
    return lists.get(0);
}
```

Running Time:

```
public static LinkedList<Integer> mergeAll(ArrayList<LinkedList<Integer>> lists) {
    while(lists.size() > 1) {
        ArrayList<LinkedList<Integer>> newLists = new ArrayList<LinkedList<Integer
            >>();
        int numLists = list.size();
        for(int i = 0; i < numLists; i++) {
            if (numLists % 2 == 1 && i == numLists - 1) {
                newLists.add(lists.get(i));
            } else {
                newLists.add(merge(lists.get(i), lists.get(i + 1)));
                i++;
            }
        }
        lists = newLists;
    }
    return lists.get(0);
}
```

Running Time: