COS 226	Algorithms and Data Structures	Fall 2005

Midterm Solutions

1. **8 sorting algorithms.**

   0 5 7 1 2 6 4 8 3 9

2. **Algorithm Properties.**

   C Max height of a binary heap with N items.

   C Max height of red black tree with N items.

   C Max function call stack depth to mergesort N items.

   D Max number of probes to search for a key in a double hashing table with N key-value pairs.

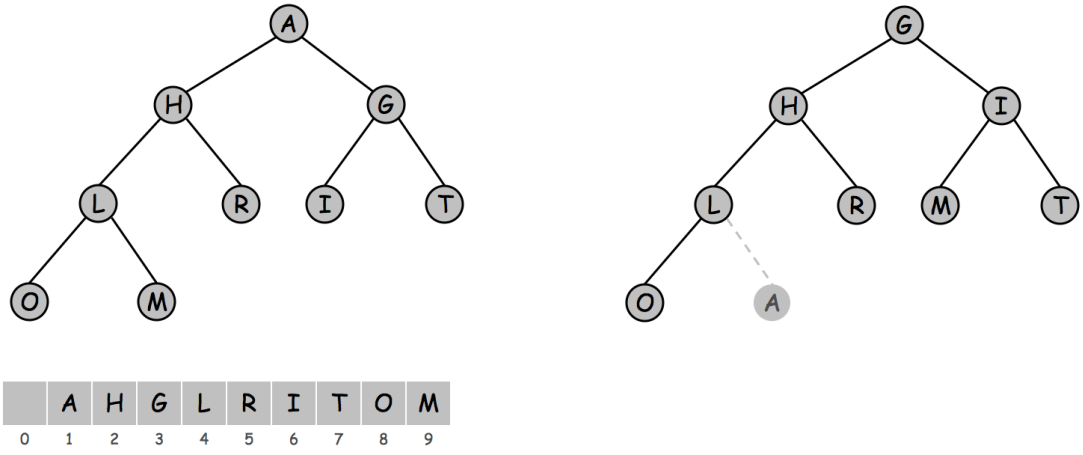
   C Max height of a WQUPC (weighted quick union with path compression) tree with N items.

   The only tricky one is WQUPC. The $\log^* N$ running time per operation is an amortized bound. We do know the height is $O(\log N)$ is for the same reason that it is for weighted quick union (Sedgewick, Property 1.3).

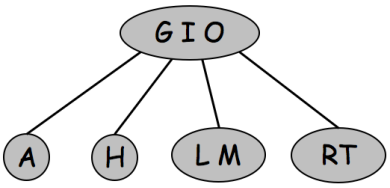
3. **Analysis of algorithms.**

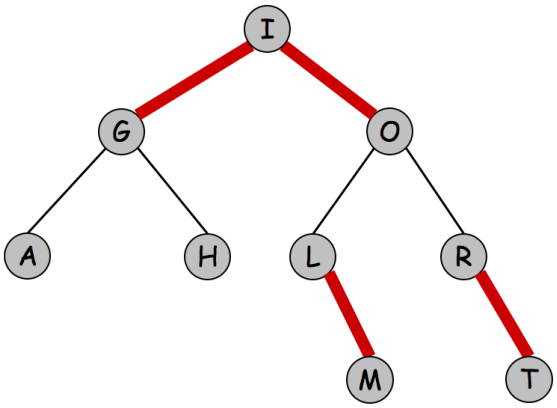
   D C B

4. **Priority queues.**



5. **Red-black trees.**





6. **Longest common substring.**

Our solution is very similar to finding the longest repeated substring, except that we must be careful to find a substring that appears in each book.

Algorithm. Form the suffixes of the two text corpuses and sort them. Now, among all adjacent suffixes in the sorted ordering *such that one suffix is from each book*, find the pair that has the longest common prefix.

Correctness. Similar to the longest repeated substring problem from lecture. Sorting brings the longest common substrings together.

Implementation detail. Before forming the suffixes, we append the character `'\1'` to the end of the first corpus and `'\2'` to the end of the second corpus. Now, by looking at the last character of a suffix, we can tell which book it came from. This enables us to easily implement the *such that one suffix is from each book* part of the algorithm. (Alternatively, we could sort the suffixes of each book independently, and then merge them together ala mergesort. While merging, we could identify the longest common substring.)

Running time. Assuming the length of the longest common substring (and longest repeated substring in each book) is not too long, we sort using 3-way radix quicksort with cutoff to insertion sort. We expect around $2N \ln N$ character comparisons, where N is the total number of characters in the input. Forming the suffixes is efficient using Java's `substring` method.

Notes. As with longest repeated substring, it's possible to solve the problem is linear time using suffix trees. However this approach is likely to be slower in practice unless the length of the longest common substring is relatively large.