Introduction to Algorithms: 6.006
Massachusetts Institute of Technology
Professors Ron Rivest and Srini Devadas

September 25, 2007
Handout 6

Problem Set 2

This problem set is due **October 11** at **11:59PM**.
Solutions should be turned in through the course website in PS or PDF form using LaTeX. The course website has links to a number of editors that are useful for writing in LaTeX.
It is recommended that you download the LaTeX source for this problem set which includes placeholders for solutions.

1. Rotating Binary Search Trees

   In this problem we'll explore the rotation operation on binary search trees. As discussed in class, this operation changes the structure of a binary search tree without affecting the inorder of the underlying nodes. See CLR section 14.2 (page 265-7) for a description of the operation.

   (a) Let $l(v)$ denote the number of nodes in v's left subtree. Let $L(T)$ be the sum of $l(v)$ over all nodes of the tree T. Show that a right rotation decreases $L(T)$. Deduce that it is impossible to do more than $O(n^2)$ consecutive right rotations in an n node tree, i.e., with NO left rotations mixed in.

   (b) Show that on an n node left path (a tree where all children are left children) it is possible to do $\Omega(n^2)$ consecutive right rotations. (Again, no left rotations allowed.)

2. How Good is this Hash?

   The following hash functions are used to hash strings of characters. Describe at least one strength and one weakness of each of the hash functions. (Don't try to do a rigorous mathematical analysis of the properties of the functions.)

   (a) Direct addressing, using the first two bytes of the hash key as an address into a 65,536-entry hashtable. (Assume all keys are at least two bytes long.)

   (b) Add the values of all bytes in the hash key into a single-byte counter, ignoring overflow. Use the result as an index into a 256-element hash table.

   (c) Start with a fixed prime number constant p_0 and a second small number s. Set $p := p_0$. For each byte b_i in the hash key, set $p := p + (p << s) + b_i$, where $p << s$ means "shift p left by s bit positions". Mod the final value of p by a prime number to produce the hash code. (Just describe what could be good or bad about this scheme depending on the values of p_0 and s.)

3. Longest Common Substring

   Humans have 23 pairs of chromosomes, while other primates like chimpanzees have 24 pairs. Biologists claim that human chromosome #2 is a fusion of two primate chromosomes that they call 2a and 2b. We wish to verify this claim by locating long nucleotide chains shared between the human and primate chromosomes.

We define the *longest common substring* of two strings to be the longest contiguous string that is a substring of both strings e.g. the longest common substring of DEADBEEF and EA7BEEF is BEEF. [1] If there is a tie for longest common substring, we just want to find one of them.

(a) Ben Bitdiddle wrote a python program to find the longest common substring of two strings. What is the asymptotic running time of his code? Assume $|s| = |t| = n$.

```
def longest_substring(s, t):
  "Finds the longest substring that occurs in both s and t"
  best = ’’
  for s_start in range(0, len(s)+1):
    for s_end in range(s_start, len(s)+1):
      for t_start in range(0, len(t)+1):
        for t_end in range(t_start, len(t)+1):
          if(s[s_start:s_end] == t[t_start:t_end]):
            current = s[s_start:s_end]
            if(len(current) > len(best)):
              best = current
  return best
```

(b) Describe a simple algorithm that finds the longest common substring in $O(n^3)$ time.

(c) Describe a simple algorithm that finds the longest common substring in $O(n^2 \log n)$ time.

(d) Describe an algorithm that finds the longest common substring in $O(n \log n)$ time.

(e) Implement your algorithm from part (d). Be sure to thoroughly comment your code so the course staff can read it. Some simple test cases are available for download on the class website.

The human chromosome 2 and the chimp chromosomes 2a and 2b are quite large (over 100,000,000 nucleotides each) so we took the first and last million nucleotides of each chromosome and put them in separate files. These files are available on the class website, and also in /mit/6.006/dna/

`chr2_first_1000000` contains the first million nucleotides of human chromosome 2, and `chr2a_first_1000000` contains the first million nucleotides of chimpanzee chromosome 2a. Note: these files contain both uppercase and lowercase letters that are used by biologists to distinguish between parts of the chromosomes called introns and extrons.

[1] Do not confuse this with the *longest common subsequence*, in which the characters do not need to be contiguous. The longest common subsequence of DEADBEEF and EA7BEEF is EABEEF.

i. How long is the longest common substring of `chr2_first_1000000` and `chr2a_first_1000000`?

ii. How long is the longest common substring of `chr2_first_1000000` and `chr2b_first_1000000`?

iii. How long is the longest common substring of `chr2_last_1000000` and `chr2a_last_1000000`?

iv. How long is the longest common substring of `chr2_last_1000000` and `chr2b_last_1000000`?

When you are finished, submit your code through the class website.

(f) **Optional:** Make your algorithm run in $O(n \log k)$ time, where k is the length of the longest common substring.

(g) **Optional:** Run your algorithm on the entire chromosomes. They are available in /mit/6.006/dna in compressed form.