CS 104 (Fall 2013) — Assignment 9
Due: 11/21/2013, 11:59am

BitBucket directory name for this homework (case sensitive): HW9

(1) Review Chapters 11, 18, 19 from the textbook.

(2) When we analyzed the Quicksort algorithm in class, we derived the following recurrence relation for $T(n)$ (the running time of Quicksort on an array of size n):

$$T(n) = T(m) + T(n-m) + n \qquad\qquad T(1) = 1$$

(Well, technically, we wrote $T(n-m-1)$, but we can ignore the -1.) Here, m was the size of the left subarray, and $n-m$ the size of the other subarray. We were not able to analyze it in this generality, so we looked at two special cases: when $m = 1$ (or $m = n-2$), and when $m = n/2$.

In this problem, you should do the analysis for $m = \alpha \cdot n$, for $0 < \alpha < \frac{1}{2}$ a constant. What this means is that in each iteration, the pivot separates an α fraction of the array from the remainder. This rules out the case that we only separate a constant *number* of elements every time; because α remains constant, the larger the array, the more of it gets separated.

(a) Use the technique of drawing a tree of recursive function calls and bounding the total work across levels of the tree in order to derive an upper bound on $T(n)$. If your bound involves any logarithms, you should make sure to specify the base of your logarithm. (Explain and show how you derived your bound — don't just give us the result.)

(b) Use a proof by strong induction to formally prove the upper bound on $T(n)$ you derived in the previous part.

(3) Write a program solving the following problem. You get a file containing $n \leq 50000$ strings, one per line. The first line contains the number n, and the next n lines the n strings. Your goal is to find out which of those strings appears most frequently in the file. (There can be duplicates of the same string.) Of course, there may be ties, like 3 strings that appear 100 times each, and all others appear less often. In that case, output all of the strings that are tied for most frequent.

Include with your program a brief explanation of your idea for solving it. Your program's running time should be $O(n \log n)$. If your program takes $\Omega(n^2)$ time, you will only get a small percentage of the credit. Do not use any STL functionality here.

Sample Input:

```
6
USC
Caltech
UCLA
ucla
Caltech
USC
```

Sample Output:

```
USC
Caltech
```

(4) Write a program for the following problem. You get a file containing $n \leq 50000$ integers, one per line, and one target integer t. The first line contains n and t, and the next n lines contain the n numbers $x_1, x_2, \ldots, x_n$, in no particular order. Your goal is to find out whether there are two numbers x_i, x_j among the n numbers such that $x_i + x_j = t$.

Include with your program a brief explanation of your idea for solving it. Your program's running time should be $O(n \log n)$. If your program takes $\Omega(n^2)$ time, you will only get a small percentage of the credit. Do not use any STL functionality here.

Sample Input:

```
6 8
5
1
2
7
4
10
```

Sample Output:

```
8 = 1 + 7
```

(5) Implement 2-3 trees on top of a `vector<T>` (or your own `List<T>`, if you prefer). Your node and 2-3 tree signatures should be the following:

```
template <class KeyType, class ValueType>
class Node {
  public:
     int numberOfKeys;
     KeyType leftKey, rightKey;
     ValueType leftValue, rightValue;
     int leftChild, middleChild, rightChild;

     Node (const KeyType & key, const ValueType & value); // one key only
     Node (const KeyType & lKey, const ValueType & lValue,
           const KeyType & rKey, const ValueType & rValue); // two keys
     ~Node ();
};

template <class KeyType, class ValueType>
class TwoThreeTree : private vector<Node<KeyType, ValueType>> {
  public:
     add (const KeyType & key, const ValueType & value);
     remove (const KeyType & key);
     ValueType & get (const KeyType & key) const;
};
```

You are encouraged to check the textbook for a similar implementation. However, your implementation should not use pointers; instead, it should store indices in the `vector` structure you are inheriting from. (Instead of `vector`, you can also use your own `List<T>` structure from Homework 4, if you prefer.) Also, you should not use two different node types depending on the degree (which is what the textbook does), but instead have the above `Node` type with a variable that remembers how many keys are in the node.

Write a simple interface in which a user can enter new student/GPA pairs, delete a student from the tree, and look up a student's GPA based on the student's name.

(6) Chocolate Problem: Same rules as in the past; 1 chocolate bar.

Write a program which, given n points (x_i, y_i) in the 2-dimensional plane (with real-valued coordinates), finds the closest pair of points. You can assume that the input is given in a file in a similar format to the problems above. Of course $\Theta(n^2)$ is easy. But your algorithm should have running time $O(n \log n)$. Please don't go out Googling for an algorithm here — you'd find it readily. The fun is figuring one out on your own.