Algorithms
Homework 5

Saad Mneimneh
Computer Science
Hunter College of CUNY

Readings
Based on Hashing, binary search trees, and red-black trees...

Problem 1
An array of size n has a majority element x iff x appears more than $n/2$ times in A. Of course, not every array has a majority element.

(a) Describe how you can use hashing to find the majority element in expected $O(n)$ time.

Note: If elements are shown one at a time, you can easily modify your algorithm to provide the majority element (or declare that it does not exist) in expected $O(1)$ time upon seeing every new element.

(b) Describe an $O(n)$ time algorithm to do the same.

Problem 2
Consider an array of n points where each point has integer coordinates in two dimensions.

(a) Design an efficient algorithm to determine whether any three of the points make a right triangle of the form $\{(i,j),(i,\ell),(k,j)\}$.

(b) Do the same for the case of a square of the form $\{(i,j),(i,\ell),(k,j),(k,\ell)\}$.

Problem 3 (Optional)
This came up in class discussion. Consider a hash function for strings where

$$h(s) = \Big(\sum_{i=0}^{n-1} s[i](m+1)^i \Big) \bmod m$$

where s is a string of length n, and $s[i]$ is its i^{th} character interpreted as an integer in $[0, m]$. Prove that swapping two characters in s produces the same hash value. *Hint*: Use the binomial theorem.

Problem 4 (Optional, understand a little more the math of universal hashing)
In the analysis of hash tables, we often make the assumption that the hash function has the simple uniform hashing property: This means that given a key x, $P[h(x) = k] = 1/m$ for every x and $k = 0 \ldots m-1$, **independently** of where any other element hashes to.

(a) Show that a universal hash family does not imply simple uniform hashing. *Hint*: You can show this by an example of a universal hash family that always maps a given key to the same value.

We say that a family $\mathbb{H}$ of hash functions is 2-universal if for every pair of different keys (x, y), and for every h chosen at random from $\mathbb{H}$, $(h(x), h(y))$ is equally likely to be any of the m^2 possible values $(0, 0), \ldots, (m-1, m-1)$.

(b) Show that a 2-universal family is universal and has the simple uniform hashing property. (Therefore, a universal family is not necessarily 2-universal.)

Hint: Show $P(h(x) = h(y)) = 1/m$ and that $P(h(x) = k) = 1/m$. Finally, find $P(h(x) = k | h(y) = \ell)$.

(c) Let m be prime. Suppose we modify the universal hash family we saw in class in the following way (where each key is divided into r chunks, each of which is in $[0, m-1]$):

$$h(x_0, \ldots, x_{r-1}) = \Big(b + \sum_{i=0}^{r-1} a_i x_i \Big) \bmod \ m$$

and $(a_0, \ldots, a_{r-1}, b)$ are chosen at random in $[0, m-1]$. Show that this family is 2-universal. *Hint*: Assume $h(x) = k$ and $h(y) = \ell$ and subtract them to perform an analysis similar to the one we did in class.

(d) Suppose $\mathbb{H}$ is universal, and we choose a hash function randomly from $\mathbb{H}$ and keep it secret. Suppose that an adversary who knows $\mathbb{H}$ learned the value of $h(x)$. Show that the adversary can now find a $y \neq x$ such that $h(x) = h(y)$ with probability greater than $1/m$. Show this cannot be the case for a 2-universal hash family.

Hint: Show this family is universal and describe what the adversary can do:

	h_1	h_2
x	0	1
y	0	0
z	1	1

For 2-universal hashing, one should be able to argue that the adversary's chance is at most $1/m$ by looking for $P(h(x) = h(y)|h(k) = k)$.

Problem 5 (Optional, experimental)
In the past, I have written some code to test the C++ redblack tree and hash table implementations as part of its STL. I will share the code with you here:

`www.cs.hunter.cuny.edu/~saad/courses/alg/hw/redblack_vs_hash.c`

`www.cs.hunter.cuny.edu/~saad/courses/alg/hw/romeo.txt`

The file `redblack_vs_hash.c` has some code that illustrates the use of these structures. The objects are string/int pairs. The code uses the file `romeo.txt`, which contains text from Shakespeare's Romeo and Juliet.

(a) Download the files, compile, and run the code.

(b) Read the comments and make yourself familiar with the use of the STL `map` and `unordered_map` (redblack tree and hashtable, respectively). In the main function, you can switch between the use of a hash table and a redblack tree. Try both and observe the running time of the program. Is it consistent with what you know about the complexity of operations on redblack trees and hash tables?

(c) Explain what the program is doing (to make sure you understand what's going on). *Hint*: it's about counting words.

(d) You may choose to use a redblack tree or a hash table. Modify the program to do the following:

- Traverse the data structure using its iterator (an example is provided) and add all the pairs to a vector
- Look at the documentation for the sort function in C++ and how it can be used to sort vectors, make the appropriate `#include` and `using`

statements to be able to use it. For instance, you might want to look at: `https://cplusplus.com/reference/algorithm/sort/`.

- Provide a comparator function, a function that takes two string/int pairs and returns which one should come first based on the value of the integer.

- Use the sort function on the vector with the comparator function to sort the vector and obtain the 20 most frequent words in `romeo.txt`.

Problem 6
Assume that, in a binary search tree, we augment the information in a node to include the sum of all its predecessors keys plus its own. Consider the following pseudocode to compute `predsum` for each node x in the tree.

```
x = T.minimum()
while x! = T.NIL
   do x.predsum = 0
      y = x
      while y! = T.NIL
         x.predsum = x.predsum+y.key
         y = T.predecessor(y)
   x = T.successor(x)
```

(a) What is the running time of this algorithm?

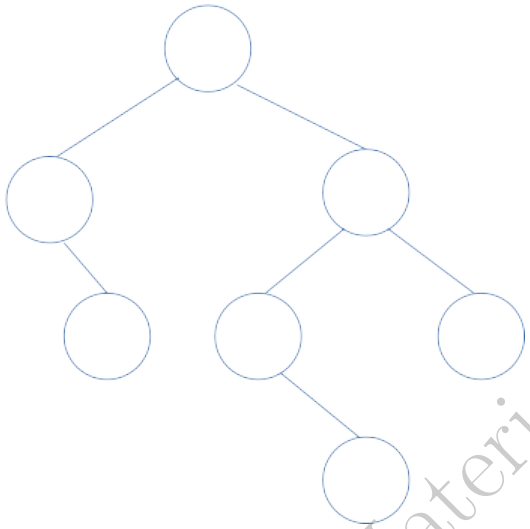
(b) Suggest a faster algorithm to compute all predsums. Provide a pseudocode similar to the one above.

(c) Describe a recursive algorithm (with pseudocode) that will compute for each node x the sum of all keys in it's subtree (including itself). Call this sum `treesum`. Your algorithm should have linear time complexity.

(d) Which of the two `predsum` or `treesum` you think is a better information to augment the tree with?.

Problem 7

(a) Place the keys $\{1, 2, 3, 4, 5, 6, 7\}$ in the following binary search tree.



(b) Color the nodes red and black to make the tree a valid red-black tree.

(c) Perform a Right-Rotate at the node containing key 6. Draw the binary search tree that results and label the tree with the same keys. Can you color the nodes to make the tree a valid red-black tree? Explain.

Problem 8

(a) Design an algorithm that converts a binary search tree on n elements into a min-heap containing the same elements in $\Theta(n)$ time.

(b) (Optional) Can you do the same **inplace**, i.e. convert the tree into a min-heap in $\Theta(n)$ without using any extra storage. Here adopt the standard representation of a tree for the heap with the three $left$, $right$, and $parent$ pointers. For this, you need to have a precise pseudocode to show how you will utilize existing pointers in the tree. (I have not attempted this but I believe it's possible)

(c) Can you reverse the process, i.e. convert a min-heap containing n elements into a binary search tree on the same elements in $\Theta(n)$ time? How, or why not?