CS 2150 Exam 2, fall 2015

Name ______________________________

You MUST write your e-mail ID on **EACH** page and bubble in your userid at the bottom of this first page. And put your name on the top of this page, too.

If you are still writing when "pens down" is called, your exam will be ripped up and not graded – even if you are still writing to fill in the bubble form. So please do that first. Sorry to have to be strict on this!

Other than bubbling in your userid at the bottom of this page, please do not write in the footer section of this page.

There are 8 pages to this exam. Once the exam starts, please make sure you have all the pages. Questions are worth different amounts of points.

If you do not bubble in this first page properly, you will not receive credit for the exam!

Answers for the short-answer questions should not exceed about 20 words; if your answer is too long (say, more than 30 words), you will get a zero for that question!

This exam is CLOSED text book, closed-notes, closed-calculator, closed-cell phone, closed-computer, closed-neighbor, etc. Questions are worth different amounts, so be sure to look over all the questions and plan your time accordingly. Please sign the honor pledge below.

____

____

____

____

A crash reduces
Your expensive computer
To a simple stone.

(the bubble footer is automatically inserted into this space)

Page 2: Older Stuffs

1. [3 points] Other than syntax, what are the three differences between references and pointers?

2. [3 points] Why do we prefer big-Theta over big-Oh?

3. [6 points] Convert 0x00008dc1 (which is in little-Endian) into a floating point number using the IEEE 754 floating point notation. You can leave your result in formula form. Show your work!

Page 3: Trees

4. [6 points] Write a *recursive* binary search tree `find()` algorithm. This may be in C++, pseudo-code, or a combination thereof; we aren't worried about C++-specific syntax.

5. [6 points] Give one advantage and one disadvantage of each of the four tree types studied in lecture and listed below. However, you can't use the same reason twice – so if *a* is faster than *b*, you can't also say that *b* is slower than *a*.

	Advantage	**Disadvantage**
Binary Search Tree		
AVL Tree		
Red-Black		
Splay Tree		

Page 4: Hashes (Bloom filters)

The question on the next page deal with the information on this page.

When dealing with incredibly large data sets in real life, it can often be computationally intensive to determine whether some element is in a given data set – even with tools like a hash table or a tree. The reason is because the data set can be too large to store in memory, and determining membership may involve many look-ups to the disk, which is slow. To help test for membership in a large data set like this, we can employ a data structure known as a *Bloom filter* that works like so: we maintain, in memory, an array of integers of size n. Initially, when there are no elements in our data set, each integer is set to 0. Then, for each element that gets added to the set, we apply k different hash functions to the element and take each result, mod'ed by n, to identify k (not necessarily distinct) positions in the array. We then set those values in the array to 1 and leave the rest as they were. Then, if we wish to check to see if an element is in the set, we apply the k hash functions to the possible value, take the results mod n, and check those positions in the array. If any one of those array elements is 0, then we know the element cannot be in the set. If every element is equal to 1, then we are not certain the element is in the set, but (if we choose good values for k and n) hopefully it will be most of the time.

As an example, suppose we let $n = 10$ and $k = 3$, and suppose the Bloom filter at a given point has the following contents: $[0, 0, 1, 0, 0, 1, 1, 0, 1, 0]$. Note that this array is indexed from zero. Now we want to see if the element 7 is in the set. Suppose our three hash functions are called h_1, h_2, and h_3, and that $h_1(7) = 35$, $h_2(7) = 8$, and $h_3(7) = 21$. We check each of those hash values (35, 8, and 21, respectively) mod'ed by 10 (the Bloom filter size) to see if there is at least one 0 present on those positions. Because there is a 0 in position $h_3(7) \mod 10 = 1$, we know that 7 has not yet been added to the set. If we then add 7 to the set, we would have to update positions $h_1(7) \mod 10 = 5$, $h_2(7) \mod 10 = 8$, and $h_3(7) \mod 10 = 1$, so the new Bloom filter array would have the following contents: $[0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0]$.

Consider the resulting Bloom filter from the previous paragraph: $[0, 1, 1, 0, 0, 1, 1, 0, 1, 0]$. Now we want to search for 8, and assume that $h_1(8) = 82$, $h_2(8) = 45$, and $h_3(8) = 11$. We check the positions of each of the hash functions mod'ed by the table size; so we check positions $h_1(8) \mod 10 = 2$, $h_2(8) \mod 10 = 5$, and $h_3(8) \mod 10 = 1$. Because *all* of these positions are 1, this tells us that element 8 is *possibly* in the Bloom filter, and we would have to perform the full (and expensive) search through the entire data set to retrieve the element (or tell if it does not exist).

Bloom filters, then, can only report "possibly in the data set" or "definitely not in the data set". If the former is reported, then a full search through the full data set is necessary to either definitively find the element, or determine if it is definitely not in the data set. Bloom filters can dramatically reduce the number of searches through the full data set, assuming appropriate values of n and k are chosen.

The questions on the next page deal with Bloom filters as presented here.

Page 5: Hashes (Bloom filters), page 2

6. [3 points] Suppose that $n = 10$ and $k = 3$ and the three hash functions are $h_1(x) = 5x + 7$, $h_2(x) = x^2 + 3$, and $h_3(x) = 7x + 2$. If the Bloom filter array looks like $[0, 0, 0, 0, 1, 0, 0, 1, 1, 0]$ at one point in time, what will it look like after 8 is added to the set? What will it look like if we then add 4 to the set as well?

7. [3 points] Suppose that $n = 10$ and $k = 3$ and the three hash functions are $h_1(x) = 3x + 1$, $h_2(x) = 7x$, and $h_3(x) = x^2 + 3x - 2$. If the Bloom filter array looks like $[0, 1, 1, 0, 0, 0, 1, 0, 0, 1]$ at one point in time, is it the case that 5 is in the set? Is it the case that 6 is in the set?

8. [3 points] What is the big-Theta running time of a Bloom filter that reports "possibly in set"? What if it reports "definitely not in set"? For both, briefly explain why. We are ignoring the running time of actually finding the element in the data set. You can use n and k in your big-Theta analyses. State any other reasonable assumptions that you make.

9. [3 points] What data structure would you use to store the Bloom filter values (the 0's and 1's)? Why?

Page 6: x86

10. [12 points] The x86 code, below, is supposed to implement the same functionality as the provided C++ function. This computes the Fibonacci sequence by counting down the terms in the first parameter, passing the previous term in the second parameter, and computing the ongoing sum in the third parameter. Fill in the missing x86 instructions. Assume that the standard calling convention described in lecture is followed.

```
int fib (int index, int prev, int cur) {
    if ( index <= 1 ) return cur;
    else return fib(index - 1, cur, prev + cur);
}
```

Produced x86 instructions (fill in the blanks):

```
.Z3fibiii:
.L1:    push    ebp
        mov     ebp, esp

        push    ________

        push    ________
        cmp     [ebp+8], 1

        jg      ________

        mov     ________, [ebp+16]

        _____   _________
.L2:    mov     eax, [ebp+16]
        mov     edi, [ebp+12]
        add     eax, edi
        mov     esi, [ebp+8]

        _____   ________, ________
        push    eax
        mov     eax, [ebp+16]

        _____   ________
        push    esi
        call    .Z3fibiii

        add     ________, 12

.L3:    pop     ________
        pop     esi
        mov     esp, ebp
        pop     ebp
        ret
```

Page 7: Miscellaneous

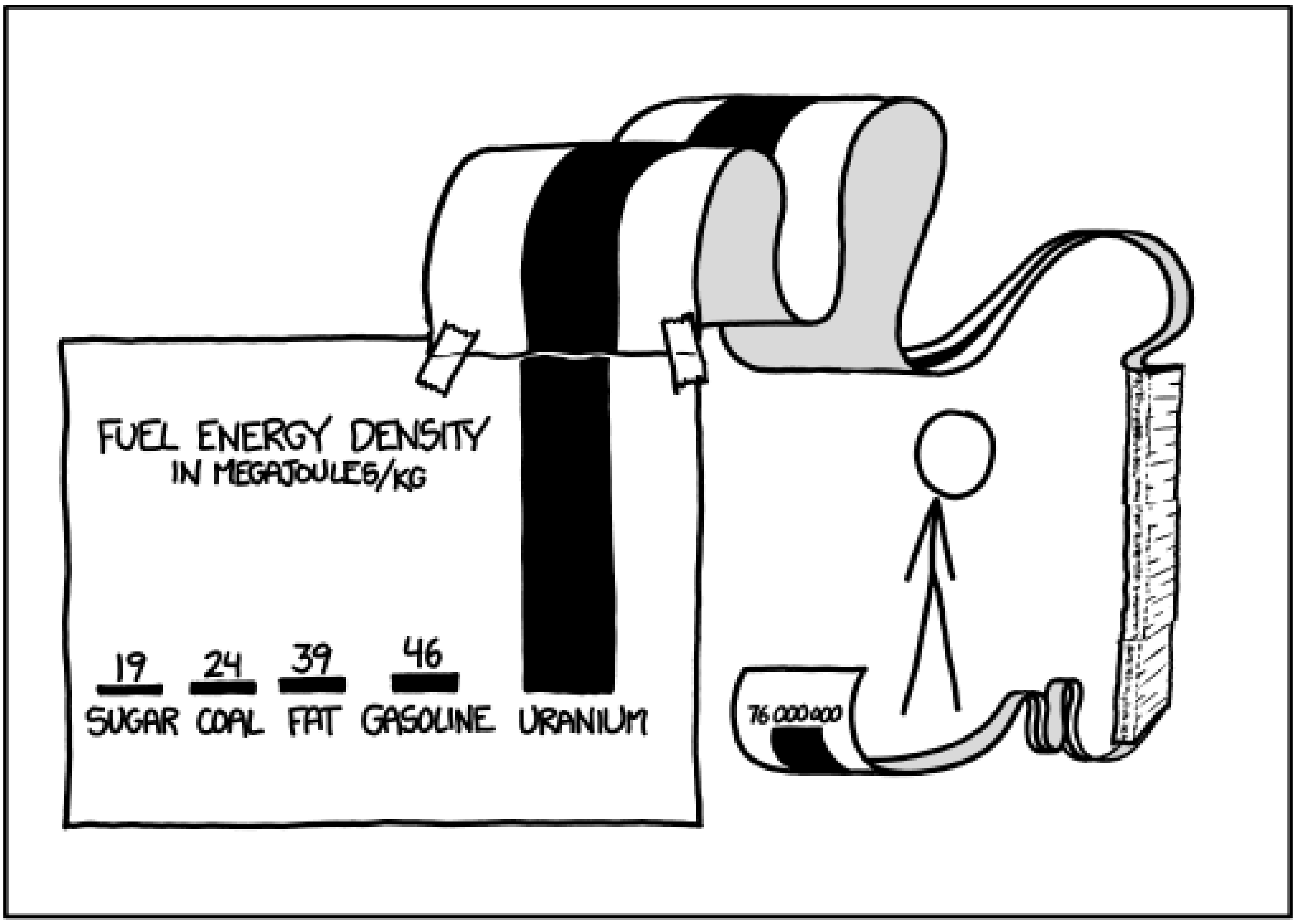
11. [3 points] Why do binary search trees often run in their worst-case run time (as opposed to hash tables, which rarely run in their worst-case run time)?

12. [3 points] How would you implement `findMax()` in a hash table?

13. [6 points] Write a snippet of IBCM code to create a `load` instruction for a multi-dimensional array. We are only looking for the relevant portion of IBCM code, not an entire program! The opcode for load is 3, and the instruction is to be stored at the `doit` label. The array value you are loading is `a[i][j]`, the array size is (x, y), and you should assume that a, i, j, x, and y are all properly initialized variables. You may assume a `mult` instruction exists that you can use. Your answer should be left in opcode form. Feel free to use any variables that you would like, but if their values are not obvious, briefly state what they are.

Page 8: No questions here

This page unintentionally left blank.



SCIENCE TIP: LOG SCALES ARE FOR QUITTERS WHO CAN'T FIND ENOUGH PAPER TO MAKE THEIR POINT *PROPERLY.*

xkcd #1162