READ THIS NOW!

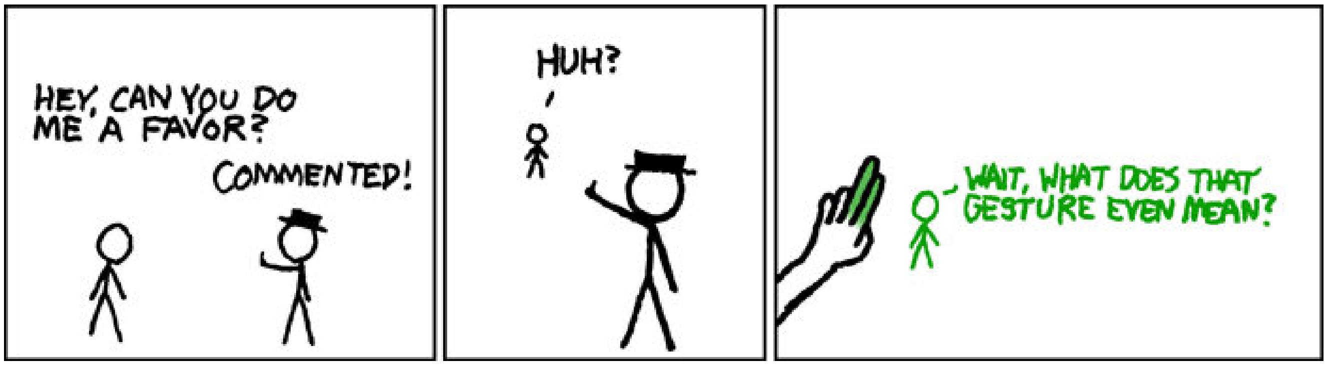
- Print your name in the space provided below.
- There are 8 short-answer questions, priced as marked. The maximum score is 100.
- This examination is closed book and closed notes, aside from the permitted one-page formula sheet. No calculators or other computing devices may be used. The use of any such device will be interpreted as an indication that you are finished with the test and your test form will be collected immediately.
- Until solutions are posted, you may not discuss this examination with any student who has not taken it.
- Failure to adhere to any of these restrictions is an Honor Code violation.
- When you have finished, sign the pledge at the bottom of this page and turn in the test and your fact sheet.

Name (Last, First) Solution

printed

Pledge: On my honor, I have neither given nor received unauthorized aid on this examination.

signed


HEY, CAN YOU DO ME A FAVOR?
COMMENTED!
HUH?
WAIT, WHAT DOES THAT GESTURE EVEN MEAN?

1. [12 points] Determine the exact count complexity function T(N) of the following algorithm. Your answer for T(N) should be in simplest form. Show supporting work!

```
for (i = 1; i <= N; i++) {          // Line  1:  1 before, 2 during, 1 exit
   x = 0;                           // Line  2:  1
   y = N;                           // Line  3:  1
   for (j = 1; j <= i; j = 2*j) {   // Line  4:  1 before, 3 during, 1 exit
      if ( i % j < 2 ) {            // Line  5:  2
         x = x*i + 2*j;             // Line  6:  4
         y--;                       // Line  7:  1
      }
      else {
         x = x - 2*j;               // Line  8:  3
         y++;                       // Line  9:  1
      }
   }
}
```

$$T(N) = 1 + \sum_{i=1}^{N}\left(2+1+1+1+\sum_{j=1}^{1+\lfloor \log i \rfloor}\left(3+2+\max(4+1, 3+1)\right)+1\right)+1$$

$$= \sum_{i=1}^{N}\left(\sum_{j=1}^{1+\lfloor \log i \rfloor}(10)+6\right)+2$$

$$= \sum_{i=1}^{N}\left(10\lfloor \log i \rfloor + 16\right)+2$$

$$= 10\sum_{i=1}^{N}\lfloor \log i \rfloor + 16N + 2$$

(if we drop the floor notation)

$$= 10\left(\log 1 + \log 2 + \cdots + \log N\right) + 16N + 2$$

$$= 10\log\left(1\cdot 2\cdot 3\cdots N\right) + 16N + 2$$

$$= 10\log\left(N!\right) + 16N + 2$$

2. Assume that $f(n)$ and $g(n)$ are positive-valued functions defined on the set of non-negative integers.

a) [6 points] What does the fact given below imply regarding any big-O, big-Ω and/or big-Θ relationships between the functions? Be complete and precise. No justifications are needed.

$$\forall n \geq 1000,\ \frac{1}{2} g(n) \leq 3 f(n) \leq g(n)$$

If you divide through by 3, you get: $\forall n \geq 1000,\ \frac{1}{6} g(n) \leq f(n) \leq \frac{1}{3} g(n)$

The first inequality yields a lower bound and the second yields an upper bound, and that gives us:

f is $\Omega(g)$
f is $O(g)$
f is $\Theta(g)$
g is $\Omega(f)$
g is $O(f)$
g is $\Theta(f)$

b) [6 points] What does the fact given below imply regarding any big-O, big-Ω and/or big-Θ relationships between the functions? Be complete and precise. No justifications are needed.

$$\forall n \geq 3,\ 7 f(n) \geq g(n)$$

If you divide through by 3, you get: $\forall n \geq 3, f(n) \geq \frac{1}{7} g(n)$

This inequality yields a lower bound, and that gives us:

f is $\Omega(g)$
g is $O(f)$

3. [10 points] A computer scientist has discovered a new algorithm for solving a certain type of problem. The fastest (previous) known algorithms for solving problems of that type have average and worst case performance that is $\Theta(N \log N)$. The new algorithm is known to be

$$\Theta\left(N^{1+\varepsilon}\right),$$

where ε is not entirely determined, but it's certain that $0 < \varepsilon < 1$. Does the new algorithm offer an improvement over the previously-known algorithms? Prove your conclusion.

We need to establish how the complexity functions compare;
we can do that by examining the limit of their ratio:

$$\begin{aligned}
\operatorname*{limit}_{N\to\infty} \frac{N \log N}{N^{1+\varepsilon}} &= \operatorname*{limit}_{N\to\infty} \frac{\log N}{N^{\varepsilon}} \\
&= \operatorname*{limit}_{N\to\infty} \frac{1/N \ln 2}{\varepsilon N^{\varepsilon-1}} \\
&= \frac{1}{\varepsilon \ln 2} \operatorname*{limit}_{N\to\infty} \frac{1}{N^{\varepsilon}} \\
&= \frac{1}{\varepsilon \ln 2} \cdot 0 \\
&= 0
\end{aligned}$$

Therefore, by the Corollary to Theorem 8 in the notes,
we see that $N \log N$ is strictly $O\left(N^{1+\varepsilon}\right)$.
So, the new algorithm is not an improvement.

4. Probing strategies for hash tables can be defined by giving a `NextSlot()` function that returns the index of the next slot in the table to be examined. For example, linear probing can be defined by the function

$$LinearNextSlot(k) = \begin{cases} 0 & k = 0 \\ LinearNextSlot(k-1)+1 & k > 0 \end{cases}$$

Another probing strategy is defined by the function

$$NextSlot2(k) = \begin{cases} 0 & k = 0 \\ LinearNextSlot2(k-1)+k & k > 0 \end{cases}$$

Yet another probing strategy is defined by the function

$$NextSlot3(k) = \begin{cases} 0 & k = 0 \\ LinearNextSlot3(k-1)+H(k) & k > 0 \end{cases}$$

where *H*(*key*) is an alternate hash function and *key* is the key for the record being inserted.

a) [7 points] Discuss advantages/disadvantages of linear probing versus the second strategy that uses *NextSlot2*().

The second strategy has a similar effect to quadratic probing, in that probing from two adjacent slots does not hit the same set of slots. That has the effect of reducing the probability of clustering when probing occurs.

That indicates that the second strategy has a decided advantage over linear probing.

b) [7 points] Discuss advantages/disadvantages of linear probing versus the third strategy that uses *NextSlot3*().

The third strategy uses variable-sized steps, like the second strategy, but the step size is determined by how the alternate hash function processes the step counter k… so two keys that collide in the same home slot will still follow the same sequence of probe slots, but keys that hash to adjacent home slots will have different probe sequences.

Again, this should reduce the probability of clustering, and that gives the third strategy a decided advantage over linear probing.

Note: this is not the same as double hashing.

c) [6 points] Discuss advantages/disadvantages of the second strategy versus the third strategy.

Both the second and third strategies will yield the same probe sequences for keys that collide in the same home slot; so neither has an advantage in that respect.

Both the second and third strategies will yield different probe sequences for keys that collide in adjacent (or nearby) slots; so neither has an advantage in that respect.

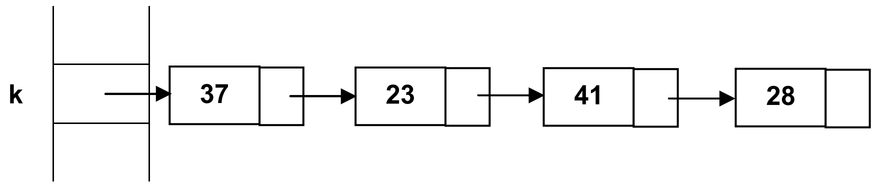
Evaluating the hash function for each probe step makes the third strategy a bit more expensive than the second.

Otherwise, there are no real advantages either way, but this is a fair advantage for the second strategy.

5. [12 points] Two developers are told to implement a hashing scheme for the same set of data records, and to use the same hash function and table size. One developer decides to use linear probing to resolve collisions. The other developer decides to use chaining with singly-linked lists in each table slot, appending new records to the lists.

Both developers test their implementations by inserting the set of data records, in the same order, into their tables, and studying the results.

When testing her implementation, the second developer discovers that one of the table slots ends up in the following state (displaying key values for the records):



Given that information, should the first developer be confident that if the record with key 41 is searched for in his implementation, no more than 3 record key comparisons will be required? Explain clearly why or why not. If not, could the number of required record comparisons be <u>less</u> than 3? <u>more</u> than 3? <u>more</u> than 4? Explain why.

We know from the given assumptions that the keys will be inserted in the same order both times, so the first developer knows that 41 will be inserted after 37 and 23 have been inserted.

Since he is using linear probing, any search for 41 must probe past both 37 and 23, so at least three comparisons are required (one per table cell that's accessed).

But… we do not know that other keys, that hashed to different home slots than *k*, were inserted before or intermingled with the keys shown above.

If so, the insertion of those keys could have occupied slots that would have otherwise been occupied by the three keys in question (37, 23 and 41), and that would result in even longer probe sequences for 41.

So, the number of comparisons to find 41 could be much larger than 4.

6. [10 points] When implementing the PR quadtree in Java, we use an inheritance hierarchy of node types in order to save memory, since different types of nodes store different amounts of data.

Consider implementing Pugh's probabilistic skip list in Java. Would it make sense to use an inheritance hierarchy of node types? If yes, describe the hierarchy. If no, explain why using inheritance would not make sense, and describe how you would implement the nodes.

The use of inheritance for the nodes in the PR quadtree let us take advantage of the fact that there was no overlap in data content between leaf nodes and internal nodes (in the PR quadtrees).

That issue does not arise in the skiplist. Every node stores a data value and one or more forward pointers. The only difference is <u>how many</u> forward pointers a node stores. So, we could implement the skiplist generically using a single node type:

```
class SkipNode {
   T          element;
   SkipNode[] forward;  // allocated dynamically in constructor to fit
                        // selected level number for node
};
```

Now, what about using inheritance?

The only variation is in the size of the pointer array, and we certainly would not want to declare a list of forward pointers as separate data members (think about the coding implications of that).

It seems we would have a linear hierarchy of node types, `SkipNode` as the abstract base for `SkipLevel0`, which is the base for `SkipLevel1`, which is the base for `SkipLevel2`, which is the base for… well, you get it…

And, we'd have to do runtime tests to determine the type of each node during a traversal of the skiplist, just as we do in the PR quadtree. That's added runtime cost.

What's the advantage gained from the use of inheritance? None.

7. [8 points] Under what circumstances would it make more sense to use a minimum-height BST to organize a collection of records, rather than an AVL tree? Why?

If we knew all the data values that would be stored in advance, and none were ever deleted, we could build a minimum-height BST and then just use it, with O(log N) cost for every subsequent operation.

But, if we had to add and/or delete elements on the fly, maintaining a minimum-height BST is simply too expensive to be useful.

Unless, of course, we didn't care about the performance of the tree…

8. Consider using a PR quadtree to organize data objects that have integer coordinates in the square region of the xy-plane that is bounded between the points (0, 0) and (1024, 1024).

a) [6 points] If there are 512 data points, and assuming a bucket size of 1, what is the smallest number of levels the quadtree could have? Justify your conclusion.

With a bucket size of 1, the maximum capacity of each level would be:

level	**capacity**
0	**1**
1	**4**
2	**16**
3	**64**
4	**256**
5	**1024**

So, there would have to be at least 6 levels in the PR quadtree.

b) [4 points] Suppose that you were to build a PR quadtree and insert records with the following coordinates:

A(400, 900) B(900, 600) C(600, 500) D(100, 100).

How many levels would the tree have if you used a bucket size of 1?

The four points here lie in different quadrants of the world, so 2 levels suffice.

How many levels would the tree have if you used a bucket size of 4?

Here one level suffices.

c) [4 points] Suppose that you inserted into the PR quadtree from the previous question a new record with the following coordinates: E(50, 50).

How many levels would the tree now have if you used a bucket size of 1?

The point lies in the same quadrant as D, and somewhat close to D, so the question is how many splits are necessary in order to separate D and E. The answer is that we'd have to split until we got leaf nodes for 64 x 64 regions (down the relevant branch of the tree). The details of that are best expressed via a diagram and are left to the student.

That would require a total of 5 levels.

How many levels would the tree now have if you used a bucket size of 4?

We'd have to split once (since the root would have been a full leaf before E was inserted), and that would be sufficient since A, B, C and D would all go into different leaf nodes before E was finally inserted in the same leaf as D. Although D and E are close, this will not require any splitting since the leaf in question is not full yet.

So, there would be 2 levels.