Exam 1  Note, I have pulled questions from a variety of exams to fit our current coverage. So your exam may differ from this one in length.

Multiple Choice (3 points each)

1. For the timing information below, what is the complexity?

<table>
<thead>
<tr>
<th>n</th>
<th>T(n)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>16</td>
<td>9</td>
</tr>
<tr>
<td>32</td>
<td>17</td>
</tr>
</tbody>
</table>

(a) O(1) (b) O(log n) (c) O(n) (d) O(n log n) (e) O(n^2)

2. For the timing information below, what is the complexity?

<table>
<thead>
<tr>
<th>n</th>
<th>T(n)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>16</td>
<td>4</td>
</tr>
<tr>
<td>32</td>
<td>4</td>
</tr>
</tbody>
</table>

(a) O(1) (b) O(log n) (c) O(n) (d) O(n log n) (e) O(n^2)

3. Algorithm C has complexity O(n). What do you expect to happen to the execution time if the problem size doubles?
   1. doubles
   2. slightly more than doubles
   3. quadruples
   4. increases by a constant

4. Algorithm D has complexity O(n^2). What do you expect to happen to the execution time if the problem size doubles?
   1. stays the same
   2. doubles
   3. slightly more than doubles
   4. quadruples
   5. increases by a constant
5. You have two solutions for the same problem. One is $O(n^2)$ and the other is $O(n \log n)$. When you compare the times for various problems, the $O(n \log n)$ algorithm is always significantly slower. How do you explain this?
   1. The clock is not accurate.
   2. Your problem sizes are too small to see the advantages of the $O(n \log n)$ algorithm.
   3. You are running on a very fast machine.
   4. The $O(n^2)$ algorithm has a higher overhead.

6. From our theorem we know:
Assume $T(n) = aT(n/b) + O(n^k)$ is the time for the function.
   1. If $a > b^k$, the complexity is $O(n^{\log_b a})$.
   2. If $a = b^k$, the complexity is $O(n^k \log n)$.
   3. If $a < b^k$ the complexity is $O(n^k)$.

Consider the following algorithm:

```c
int doit(int A[], int n){
    if (n <=1) return 1;
    int t;
    for (int i =0; i < n; i++)
        t++;
    t+= 3*doit(A,n/2)
    for (int i =0; i < n; i++)
        t++;
    return t;
}
```

What is the complexity?
(a) $O(1)$ (b) $O(\log n)$ (c) $O(n)$ (d) $O(n \log n)$ (e) $O(n^2)$

7. For the problem above, what are the values of $a$, $b$ and $k$?
   1. $a=1$, $b=1$, $k=1$
   2. $a=1$, $b=2$, $k=1$
   3. $a=1$, $b=2$, $k=0$
   4. $a=3$, $b=2$, $k=2$

8. Consider the following recursive function:
```c
void qq(int n)
{
    int i; int x;
    if ( n <= 0 ) return;
    x++; //do something
    qq(n-1);
    qq(n-1);
} // qq
```

The complexity of this function is:
   1. $O(n)$
   2. $O(n \log n)$
   3. $O(n^2)$
   4. $O(2^n)$
   5. None of the above
9. You write a hash table implementation that uses separate chaining. Which technique is most efficient for handling collisions?
   1. Linear probing
   2. Quadratic probing
   3. Double hashing
   4. None of the above

11. Consider the following code on a binary tree. What is the result of the call doit(Tree1)?

```c
void doit(Node * t)
{
    if (t==NULL) return;
    t->left= t->right;
    doit(t->left);
    doit(t->right);
}
```

(a) ![Tree diagram](a.png)
(b) ![Tree diagram](b.png)
(c) ![Tree diagram](c.png)
(d) ![Tree diagram](d.png)
(e) None of the above
12. For the code below, what is the result of a call to also(Tree2)?

```
int also(Node *t)
{
    if (t==NULL) return 0;
    return t->val + also(t->left)+ also(t->right);
}
```

(a) 10 (b) 12 (c) 62 (d) 37 (e) none of the above

13. For the code below, what is the result of a call to compute(Tree3)?

```
int compute(Node * t)
{
    if (t==NULL) return 0;
    if (t->left ==NULL && t->right ==NULL)
        return 1;
    return compute(t->left) + compute (t->right);
}
```

(a) 3 (b) 5 (c) 0 (d) 1 (e) none of the above

14. Finding a node (in an AVL tree) has what complexity?

a. O(n log n)
b. O(n)
c. O(log n)
d. O(1)
e. none of the above

15. Why is lazy deletion used to delete from a hash table?

a. To ensure efficient insertion
b. To prevent the search routine from assuming an item is not there when it really is there.
c. To reduce the load factor
d. All of the above
e. None of the above
16. In double hashing, each key has a personalized step value it uses to find the next probe location. What makes us think that by adding a value repeatedly (mod table size) we will eventually try all positions in the hash table?
   a. All positions will not be hit, but that isn’t important
   b. The fact that the table size is prime makes this more likely
   c. The personalized step value is chosen to guarantee this
   d. If all positions aren’t tried, the method is inappropriate
   e. None of the above

17. You have a hash table of size \(N\) with less than \(N\) elements in the table. In trying to place a new key, collision resolution claims it has made \(N\) probes yet it has not found a free location. Which best explains how this could happen?
   a. You have primary clustering
   b. You have secondary clustering
   c. Every unused position is marked as deleted
   d. The collision resolution algorithm is in a cyclic state of retrying the same few locations repeatedly
   e. None of the above

18. What item is at the root after the following sequence of insertions into an empty splay tree: 1, 11, 3, 10, 8, 4, 6, 5, 7, 9, 2?
   a. 1
   b. 2
   c. 4
   d. 8
   e. None of the above

**Short Answer**

1. (15 points) Find the complexity of the following pieces of code. For each problem, draw an appropriate picture to justify your answer.

   (a) void doit(int n)
   { if (n/2 <=1) return;
     int x = 0;
     for (int i = 0; i < n; i++) x++;
     doit(n/2);
   }

   (b) void doit(int n)
   { if (n/2 <=1) return;
     doit(n/2);
     doit(n/2);
   }

   (c) void doit(int n)
   { if (n <=0) return;
     for (int i=0;i<n;i++)
       for (int j=0;j<i;j++)
         cout << i*j;
   }
2. (20 points) Given a binary tree, write the function which returns TRUE if every node in a tree
is bigger in value than it's children.

For example, for the following trees

```
isBig(tree1) = true;
isBig(tree2) = false;
```

```
   tree1
     15
    / \   \
   8   12
  / \   / \
 3   6 9
```

```
   tree2
     35
    /  \
   22  12
   /      \
  3       6
     |      |
     14     |
```

3. (7 points) Given the following AVL tree, insert the value 1, doing any necessary rotations to
maintain the AVL property. Draw and label (with the name of the rotation) the tree after each rotation.

```
   5
  /   \
 3     8
 /     |
2 6
```

4. (20 points) Consider the following hash table of size 15. Use double hashing with
step(key) = 1+key%(tablesize-2)
hash(key) = key%tablesize;

A -1 in the table indicates the cell is unused.

a. Since the table size isn’t prime, show a case that is problematic.
b. Show the work needed to delete 9.
c. Show the work needed to insert 21 (assuming 9 has been deleted)

<table>
<thead>
<tr>
<th>Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>15</td>
</tr>
</tbody>
</table>
5. For the following B tree of order 5, show the tree after inserting 61.

6. For the following B-tree of order 5, show the tree after inserting 40 and deleting 85.