CS 240 S2001 Midterm

1. a) Big-Omega notation is the lower bound for the running time of an algorithm. If an algorithm g(n) is big-Omega of f(n), this means that there exists constants c > 0 and n_0 >= 0 such that |f(n)| >= c|g(n)| for all n >= n_0.

```
t = 0
for i=0 to n:
        for j=0 to n:
                for k=0 to n:
                        t = t + 1
```

b) We find the upper bound:

$$\sum_{i=0}^{n} \frac{i}{3^i} \le \sum_{i=0}^{n} \frac{2^i}{3^i} \le \sum_{i=0}^{\infty} \left(\frac{2}{3}\right)^i = 3 \in O(1)$$

We find the lower bound:

$$\sum_{i=0}^{n} \frac{i}{3^i} \ge \sum_{i=0}^{n} \frac{1}{3^i} = \frac{1}{2}\left(3 - \frac{1}{3^n}\right) \in \Omega(1)$$

The order of magnitude is \Theta(1).

c) There exists c1, c2, n1, n2 such that f(x) <= c1h(x) for n >= n1 and c2g(x) <= h(x) for n >= n2. Hence we have

af(x) + bf(x) <= (c1a + c2b)h(x) for max(n1, n2)
Let c3 = c1a + c2b. Then we have found a constant c3 where af(x) + bf(x) <= c3h(x) for max(n1, n2).

d) We have that 2^n <= 4^n for n >= 0. Hence we have that 2^n is O(4^n).

Suppose that 4^n is O(2^n). Then there exists constants c and n0 such that 4^n <= c2^n for n >= n0. Then for n >= n0, we have 2^n <= c. This is a contradiction, since there is no constant where this is true for all n greater than some constant n0. Hence 4^n is not O(2^n).

2. A) O(log(log n))

B) O(n)

C) O(log n)

D) O(log n)

E) 2^(h+1) - 1

F) Expected = O(log n)
Worst = O(n)

G) Expected = O(log n)
Worst = O(log n)

3. a) Let T(n) be the runtime of foo. Then we have that T(1) = 1, and

$$T(n) = 2T(n/2) + \frac{n}{\frac{1}{n}} = 2T(n/2) + n^2$$

There are two recursive calls, each splitting the array in half. These take T(n/2) time. The last while loop starts at x=n, and each step decrements x by 1/n until it reaches 0, hence it takes n/(1/n)=n^2 time.

b) We prove that T(n) <= 2n^2-n by induction

$$T(n) = 9T(n/3) + n \le 9\left(\frac{2n^2}{9} - \frac{n}{3}\right) + n$$

$$= 2n^2 - 2n \le 2n^2 - n$$

4. a) i. 5432169

ii. 3621459

b) The optimal ordering of the elements is to order them in non-increasing order of p(e). We prove the contrapositive: if we don't put the items in this order, then there is a different ordering with a smaller expected number of comparisons. Assume we have keys k0,...kn-1, with index(ki) = i, and assume the order of keys is not non-increasing. This means we have p(ki-1) < p(k) for some index 0 < i < n. We define a new order by exchanging ki and ki-1. If we subtract the expected cost of the two orderings, we get p(ki-1) - p(k) < 0, hence the new order is more optimal.

5. a) [8, 5, 12, 3, 6, -, 13, 1, -, -, -, -, -, -, -]

b) The array-based structure uses 15*4 = 60 bytes.
The linked structure uses 7*4+7*2*4 = 84 bytes.

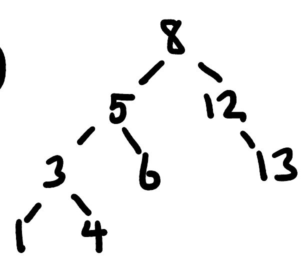
The array-based structure uses less space, since it doesn't need to store references.

c) It doesn't matter since they are the same asymptotically.

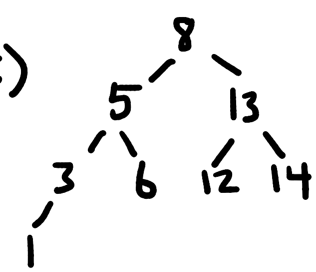
6. a) Condition code = balance factor?

1	3	5	6	8	12	13
0	-1	-1	0	-1	1	0

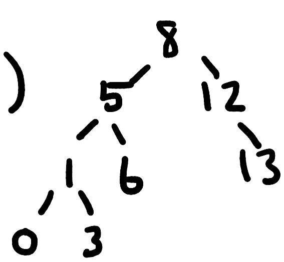
b)



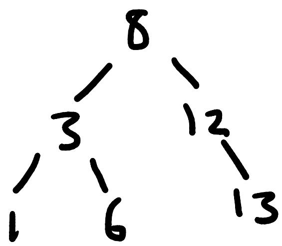
c)



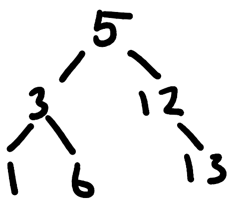
d)



e)



f)



7. We did not do this lol

8. a) It takes O(1 + n/m) time, since it takes O(1) time to find the key, and O(average bucket size) time to find the key in the array. The average bucket size is n/m.

b) We did not do coalesced chaining lol