**Problem Set 8, Part I**

**Problem 1: Hash tables**

**1-1) linear**

| | |
|---|---|
| 0 | ant |
| 1 | flea |
| 2 | bat |
| 3 | cat |
| 4 | goat |
| 5 | dog |
| 6 | bird |
| 7 | bison |

**Duck is overflowed**

**1-2) quadratic**

| | |
|---|---|
| 0 | |
| 1 | |
| 2 | |
| 3 | cat |
| 4 | goat |
| 5 | bird |
| 6 | bison |
| 7 | dog |

**ant, flea, bat, duck is overflow**

**1-3) double hashing**

| | |
|---|---|
| 0 | ant |
| 1 | bat |
| 2 | flea |
| 3 | cat |
| 4 | goat |
| 5 | bison |
| 6 | bird |
| 7 | dog |

**duck is overflow**

**1-4)** probe sequence:

"cat" = 3, 6

1) 3 x
2) (3 + (1 * 3)) % 8 = 6 - open

**1-5)** table after the insertion:

| | |
|---|---|
| 0 | |
| 1 | leopard |
| 2 | |
| 3 | fly |
| 4 | toad |
| 5 | |
| 6 | cat |
| 7 | terrier |

**Problem 2: Comparing data structures**

From all requirements, I think the Balanced Search Tree would be the best data structure choice here. Due to the main fact that in a Balanced Binary Search Tree, the items sorted would be placed next to each other, so we can easily scan for words with the same prefix of the item.

We can’t select the same prefixes using a Hash table because items with the same prefix will not only have radically different hashcodes but also have no correlation between them - this would be a disaster to do with the hash table/function. Hence, a Balanced Search Tree is ideal in this case.

In addition, we are unsure of the input size. We must increase the size of the database, hence Hash tables are only efficient when we know the input data size. 2-3 trees will be still balanced no matter how many more inputs will be stored.

For better efficiency, the worst-case run time for 2-3 trees will be log2 n since it is balanced three. This gives us the result that log2 (one million) = 19.9315685693, so it is less than 20 operations per retrieval.

**Problem 3: Complete trees and arrays**

**3-1)** the parent: 24 → (50 - 1) / 2 = 24
the left child: 101 → (50 * 2) + 1 = 101
the right child: 102 → (50 * 2) + 2 = 102

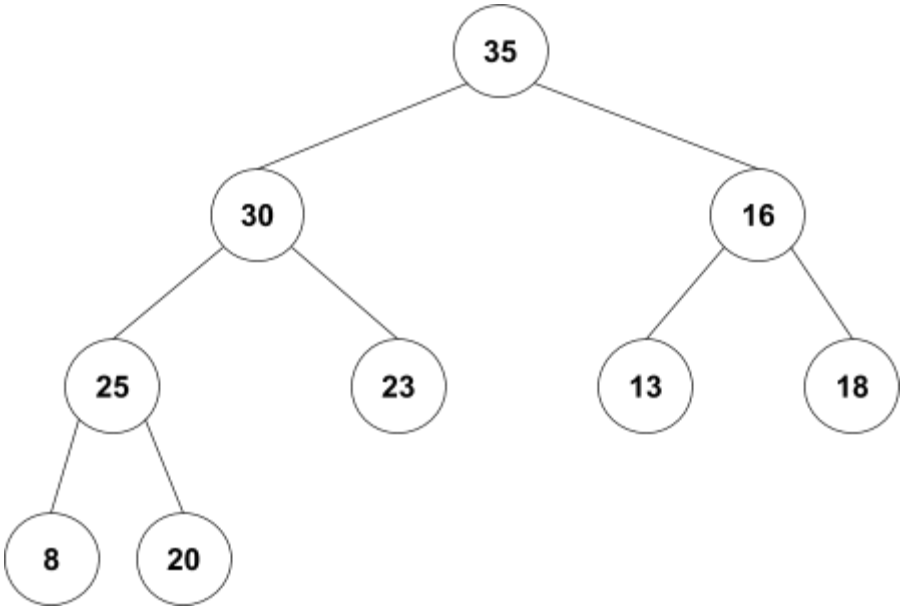
**3-2)** the height: 7
Because of log2 (200 + 1) = 7.65…., so the height will be 7. (2^ 7 = 128 and 2^ 8 = 256, 200 is in between so it will be 7

**3-3)** left child.
Because 2^7 = 128, so 200 - 128 = 73 which is an odd number, so the rightmost leaf node will be eventually placed in the left child of the last parent node.

**Problem 4: Heaps**

**4-1)**

**after one removal**




**after a second removal**

(copy your revised diagram from part 1 here, and edit it to show the result of the second removal)

**4-2)**