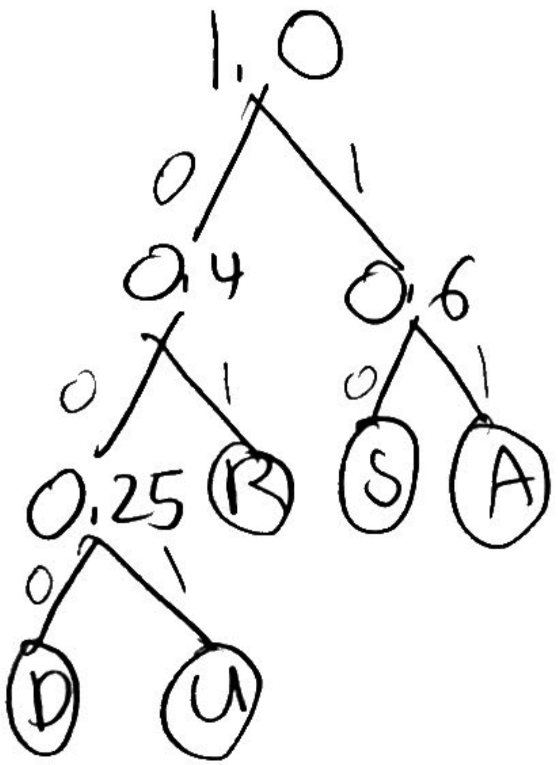
1a) Final Huffman Tree

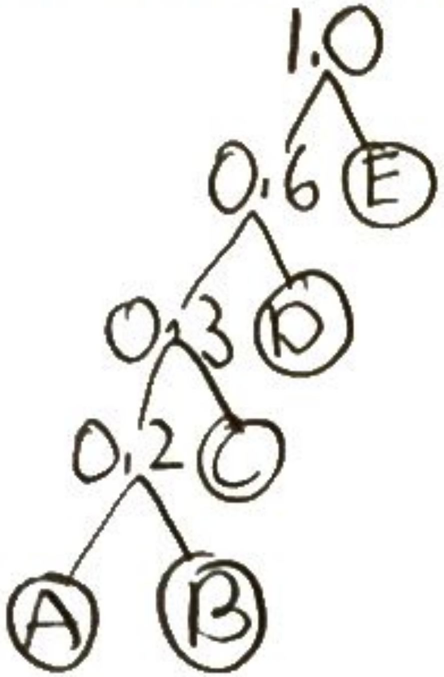


All Images

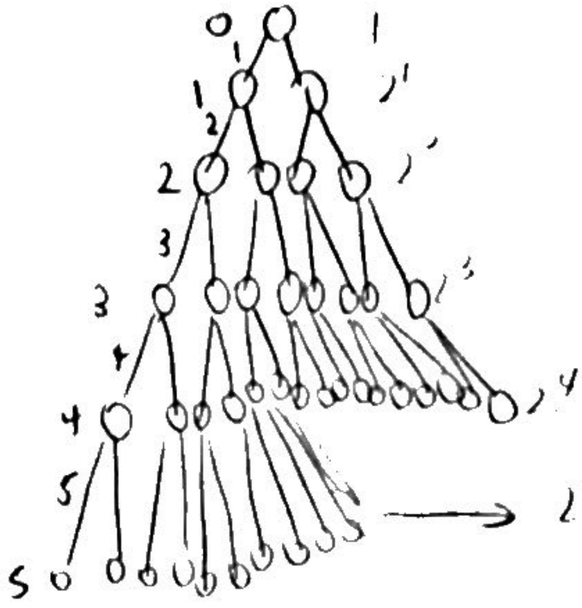
1b) Probabilities

(A,0.1) (B,0.1) (C,0.1) (D,0.3) (E,0.4)

Huffman Tree



1c)

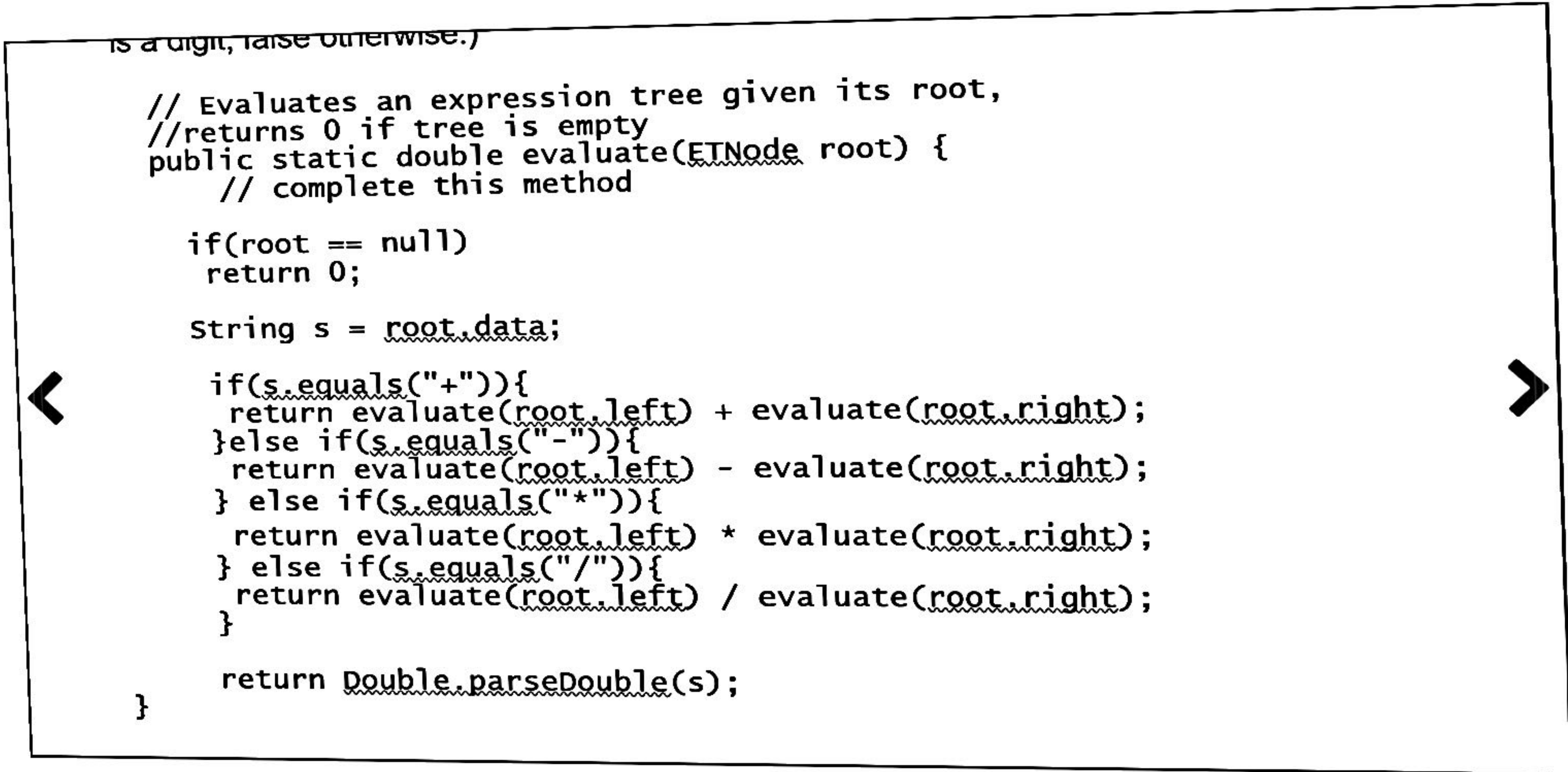


$\longrightarrow 2^5 = 32$ characters

(equal # of comparisons)

Since all of the charcters are coded in 5 bits as there are 5 comparisons the ratio is 5:8.

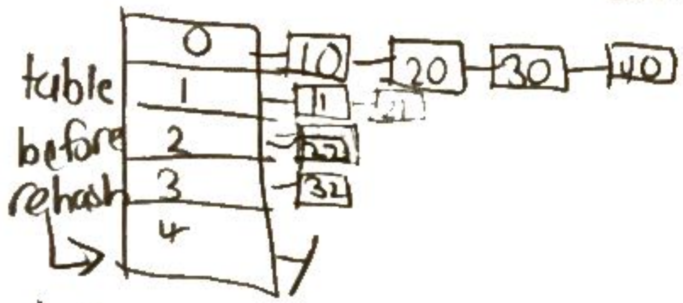
$\boxed{5:8}$



is a digit, false otherwise.)

```
// Evaluates an expression tree given its root,
//returns 0 if tree is empty
public static double evaluate(ETNode root) {
    // complete this method

  if(root == null)
   return 0;

  String s = root.data;

   if(s.equals("+")){
    return evaluate(root.left) + evaluate(root.right);
   }else if(s.equals("-")){
    return evaluate(root.left) - evaluate(root.right);
   } else if(s.equals("*")){
    return evaluate(root.left) * evaluate(root.right);
   } else if(s.equals("/")){
    return evaluate(root.left) / evaluate(root.right);
   }

   return Double.parseDouble(s);
}
```

3a) - average # of comparisons < 2 ✓
- worst case # of comparisons 4 ✓

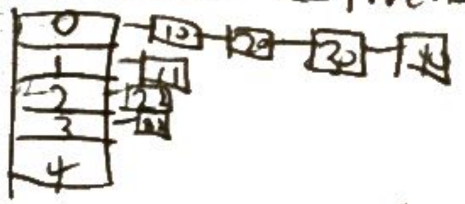


$$\frac{1+2+3+4+1+1+1}{7} < 2$$

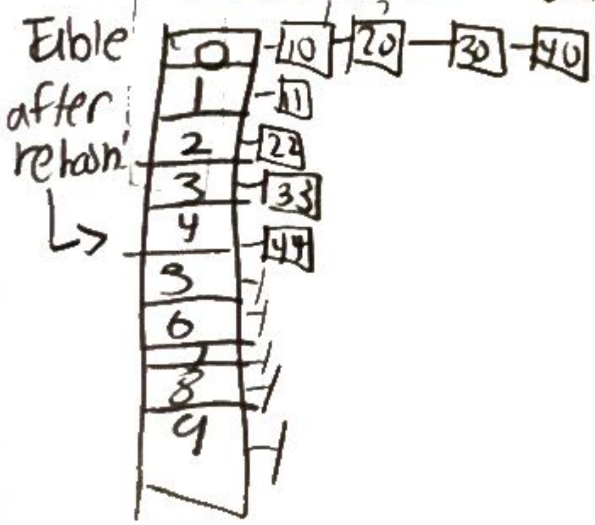
b) load factor is 1.5

$\frac{7}{5} < 1.5$ ✓

- Hash table stays the same as the load factor < threhold



c) after one more element is inserted it rehashes as load factor > threshold



8 numbers in this smallest data set
10, 20, 30, 40, 11, 22, 33, 44

Average number of comparisons for success

$$\frac{1+2+3+4+1+1+1+1}{8} = 1.75$$

1.75 comparisons number of success

Describe an algorithm for how you would do this (no Java code, write steps in plain English). Your algorithm should not look at more documents in the hash table than is absolutely necessary to find those with frequencies >= `f` for the given words.

Step 1: Locate where the word is stored in the hashtable
Step 2: if word is not found go on to Step 4
Step 3: Check if the first doc found has a frequency greater than f. If it does then check if already contains that doc. If it does then do not add it and move on to step 4 or else add it.

Step 4: If there are words then move on to step1 with that word but if there are not then this program will end.

3c) [4 pts]

This part pertains to the algorithm you came up with in Q3b above.
Assuming that a total of **n** document names appear in the hash table, and there are **k** document matches (duplicates included, of which **u** are unique) over all the 50 words combined, derive the big O running time of your algorithm.

For each big O component of the running time, specify whether it is worst case or expected case. You don't have to add up the components to a single big O.
Show your work - just big O answers without derivation will get no credit.
Also, your big O derivation will not get credit if your algorithm in 3b) is incorrect.

Locate the word – 50 times (expected) since we have to find the word regardless if it does contain it or not

Checking if the word frequency is greater – 50 times (worst case) since if hashset contains matches for all of the 50 words

Adding it to the hash set – U times (expected) since we will always end up adding all of the unique matches