Huffman Coding Algorithm

* Time Complexity of Algorithm

HuffmanTree(nodearr)

1.if (nodearr.First == nodearr.Tail) O(1)

2.top = nodearr.getTop O(1)

3.return O(1)

else

1. node = new HeapNode O(1)
2. newnode->left = nodearr.getmin() O(1)
3. newnode->right = nodearr.getmin() O(1)

7. newnode->setfrequency(newnode->left->getfrequency() + newnode->right->getfrequency()) O(1)

1. nodearr.add(newnode) O(logn)
2. HuffmanTree(nodearr) //This function call n time,so O(n)

The Time Complexity For Huffman Tree Build is:

There is only One step which takes O(logn) time other all steps take O(1) constant time and the function calls may go to N times.

So,

Time Complexity =O(n(1+1+1+1+1+1+1+logn))

Time Complexity =O(n(1+logn))

Time Complexity=**O(nlogn)**

Hence the Time Complexity for Huffman Coding Algorithm is **O(nlogn).**

* Correctness of Algorithm

We prove the correctness of Huffman’s algorithm by induction on the amount of symbols n within the alphabet.  
The base case, n = 2 is clear because the sole possibility (that isn't obviously suboptimal) may be a code where both code words are one bit long, which is what Huffman’s algorithm produces during this case.

Suppose that the algorithm produces an optimal tree for alphabets with n − 1 ≥ 2 symbols and their associated frequencies. We will prove that it produces an optimal tree for alphabets with n symbols and their associated frequencies.

Let C be an alphabet with n symbols, and f(n) be the frequency for each all the Characters. Let T be the tree produced by Huffman’s algorithm for all Characters. We must prove that H is optimal for this input.

As the Algorithm states that there are two siblings with minimum frequency in the Priority Que. Let`s make a new Node z which has a node of minimum frequency at left and second minimum at the right.

Intuitively, we are removing the x and y node from the priority Que and add a new node Z, whose frequency is the sum of the frequencies of x and y. Let T^ is the new Tree after Replacing the x and y from the Priority que and a new node Z in it.

T^=(T-{x,y})U {Z}

Now Note that T^ produced by huffman algorithm by the input of Character and frequencies. So By inductive Hypothesis,

T^ is Optimal for Q(having character and frequencies)

Now, let T be an optimal tree for Γ, f. Without loss of generality, wewillassume that x and y are siblings and are at maximum depth of T. (If not, wewillmove them in order that they'resiblings at the maximum depth of T without increasing the weighted average depth of the tree, by swapping them with symbols that are siblings at theutmostdepth.) Since T is optimal for Γ, f, so is H. So, Huffman’s algorithm produces optimal trees for alphabets with n symbols and their associated frequencies.

Now Let us see Some Observation Proofs for the Huffman coding Algorithm.

**Observation 1:**

Let T be any optimal prefix code tree, and let a and b be two children at the maximum depth of the tree. If {x, y} = {a, b} we are done. Otherwise, from the fact that x and y have the lowest frequencies, we may label the nodes such that f(b) ≤ f(c) and f(x) ≤ f(y). Now, since x and y have the two smallest frequencies it follows that f(x) ≤ f(b) and f(y) ≤ f(c). Because a and b are at the deepest level of the tree we know that dT (a) ≥ dT (x) and dT (b) ≥ dT (y). Thus, we have f(a)−f(x) ≥ 0 and dT (c)−dT (x) ≥ 0, and hence their product is nonnegative. Now, if we change the positions of x and a in the tree, giving a new tree T\*.

Now Our tree has been changed from the T to T\*,

By subtracting the old contributions of these nodes and adding in the new contributions we have

B(T \*) = B(T) − (old cost for b and x) + (new cost for b and x)

B(T\*)= B(T) − (f(x)dT (x) + f(b)dT (b)) + (f(x)dT (b) + f(b)dT (x))

B(T\* ) = B(T) + f(x)(dT (b) − dT (x)) − f(b)(dT (b) − dT (x))

B(T\*) = B(T) − (f(b) − f(x))(dT (b) − dT (x)) ≤ B(T)

Since T was an optimal tree, T\* is also an optimal tree. By a similar argument, we can switch y with b to get a new tree T\*\*. Again, the same thing implies that T\*\*,and prove that T\*\* is also optimal. The final tree T\*\* satisfies the statement of the claim.

The above claim applies to just one pair of nodes, those with the lowest frequencies. To show that the entire Huffman tree is optimal, we need to extend this argument. We will do it by induction. In order to reduce from n characters to n − 1, we will do the same reduction as Huffman algorithm does; namely we will merge characters x and y into a new meta-character z, whose frequency is the sum of the frequencies of x and y.

Thus the Observation has been Proved.

**Observation 2:**

Here, d denotes the depths of x and y in tree Tn. and z is at depth d-1 in T(n-1). Because z replaces x and y the cost of two tress satisfies.

B(Tn) = B(Tn−1) − (z’s cost in B(Tn−1)) + (x and y’s costs in B(Tn))

B(Tn) = B(Tn−1) − p(z)(d − 1) + (p(x)d + p(y)d)

B(Tn) = B(Tn−1) − p(z)(d − 1) + p(z)d

B(Tn) = B(Tn−1) + p(z).

Here, the cost of trees Tn and Tn-1 is different just because of the fixed term p(z), and it does not depend on tree’s structure. That’s why the subject to this replacement, minimizing the cost of Tn is equivalent to minimizing the cost of Tn-1. Hence proved.

**Observation 3:**

This proof is by induction on n, the number of characters. The base case (n = 1) is trivial so there is only one tree possible. If n ≥ 2, then By Claim 1, as we know that the two characters x and y of lowest probability are the siblings at the deepest level of an optimal tree. This Huffman’s algorithm replaces nodes by a character z whose probability is the sum of their probabilities. Then by induction, Huffman’s algorithm computes the optimum tree over the resulting alphabet of n − 1 symbols. Replacing z with nodes x and y results in a tree Tn whose cost is higher by the fixed amount p(z) = p(x) + p(y). Here Tn−1 is optimal, and the cost of replacement does not depend on the tree’s structure, So Tn is also optimal.