3- Haffman

I used heapq which is a module provides an implementation of the heap queue algorithm(priority queue).

For encode function I stored values with frequencies in a dictionary because it takes O(1) to get the key:

```
freq_dict = dict(sorted(dict_items, key=lambda i: i[1]))
```

In merge function:

heapq.**heappush**(*heap*, *node*)

Push the value *in the node* onto the *heap*, maintaining the heap invariant.

heapq.**heappop**(*heap*)

Pop and return the smallest value from the *heap*, maintaining the heap invariant. If the heap is empty.

Decode function to visit the tree and assign 0 to left child and 1 to the left child.

Time Complexity

For encode function it takes O(n) -> n is the number of char in the data we encode

For merge function it takes O(1)

For decode function it takes O(logn) because we use min priority heap

Haffman Time complexity is O(nlogn)

Space Complexity

O(n) -> n is the number of data we encode.