1. True, True

2. No, Yes

   (a) This is false.
   Let G has vertices v_1, v_2, v_3, v_4 with edges between each pair of vertices, and with the weight on the edge from v_i to v_j equal to i+j. Then every tree has a bottleneck edge of weight at least 5, so the tree consisting of a path through vertices v_3, v_2, v_1, v_4 is a minimum bottleneck tree. It is not a minimum spanning tree, however, since its total weight is greater than that of the tree with edges from v_1 to every other vertex.

   (b) It is true.
   Suppose that T is a minimum spanning tree of G, and T' is a spanning tree with a lighter bottleneck edge. Thus, T contains an edge e that is heavier than every every edge in T'. So, if we add e to T', it forms a cycle C on which it is the heaviest edge (since all other edges in C belong to T'). By the cut property, e does not belong to any minimum spanning tree, contradicting the fact that it is in T and T is a minimum spanning

3. Sort the edges from the max to min weight. Run Kruskal Algorithm

4. Huffman code (CLRS-16.3, KL-4.8)

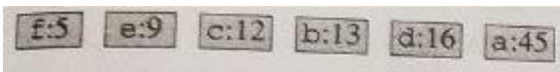
(i)

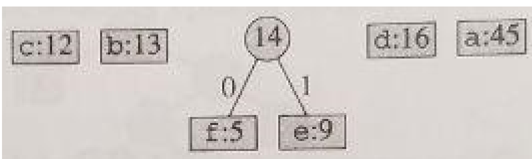
Consider the table ***C*** of characters and their frequencies.

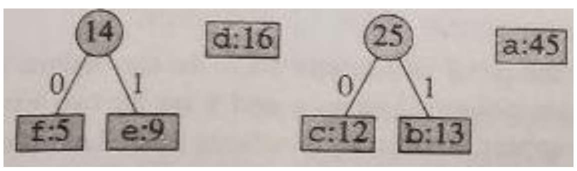
Example [CLRS-16.3]:

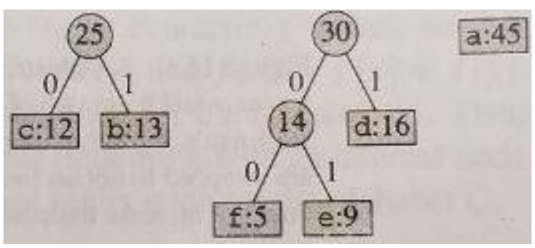
c	a	b	c	d	e	f
F[c]	45	13	12	16	9	5

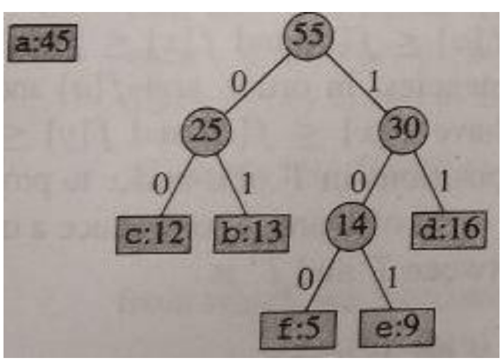
Using a priority queue, create a binary tree that merges the nodes with least weight (frequency) and insert them back into the tree

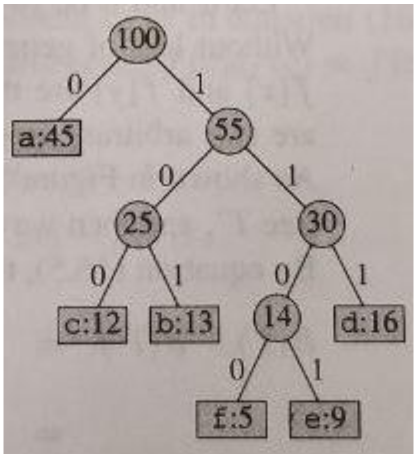












Huffman(C):

Q = a priority queue

for $c \in C$ **do**: *insert*(*Q*, *new_node(*$c, f[c]$*)*)

for *i = 1* **to** *|C|-1* **do**

z = new_node()

left[z] = ExtractMin(Q)

right[z] = ExtractMin(Q)

f[z]=f[left[z]]+f[right[z]]

insert(Q,z)

return *ExtractMin(Q)*

(ii) [CLRS-16.3] The following lemmas show the proof:

Lemma 1) Let *x* and *y* be the least freq. characters. There exists an optimal coding where the codes of x and y have the same length and only differ in the last bit.

Lemma 2) Let *x* and *y* be the least freq. characters in *C*. Let $C' = C - \{x, y\} \cup \{z\}$, where $f[z] = f[x] + f[y]$. Let *T'* be the optimal tree for *C'*. Adding the leaf nodes *[x]* and *[y]* to the node *[z]* in *T'* gives the optimal tree for *C*.

(iii)

D: document

T: Huffman tree

```
Decode(D,T):
  i=0, output=""
  while(i<|D|) do
     (c,i)=NextChar(D,T,i)
     add c to the end of output
  return output

NextChar(D,T,i):
  N=T.root()
  while (N is not a leaf node) do
     if (D[i]==0): N=N.left()
     else: N=N.right()
     i+=1
  return (N.char(), i+1)
```

5. **Dynamic MST**

Let T be the current MST and $e = (u, v)$the edge whose weight is updated to w. Consider the following cases:

- e belongs to the MST and its weight has reduced: no updated is needed.
- e does not belongs to the MST and its weight has increased: no updated is needed.
- e does not belongs to the MST and its weight has decreased: update the tree, if e is not the max-edge in the cycle it introduces in T. $[O(n)]$
- e belongs to the MST and its weight has reduced: Remove e from T. This will generate a cut. Find the edge with least weight in the cut-set and add it to the tree. $[O(m)]$

updateMST(u,v,w):

let T be the current MST

if $e = (u, v)$ belong to $T.edges$:

| $w' = e.weight$

| $e.weight = w$

| **if** $(w' < w)$:

| remove e from T

| $C_1 = connectedComponent(u)$

| $C_2 = T.nodes - C_1$

| CS = Cutset(C_1, C_2)

| Add $\min(CS)$ to T

else: //(u, v) does not belong to $T.edges$

| $w' = e.weight$

| if $(w' < w)$ then return

| Start BFS from u and find the path P to v

| e_r = the edge with max weight in P

| if $(e_r.weight > w)$:

| remove e_r from T

| add (u, v, w) to T

$[O(n + m)]$