HUFFMAN CODES

Huffman codes are used in data compression. Huffman greedy algorithm uses a table of the frequencies of occurrence of the characters to build up an optimal way of representing each character as a binary string.

Huffman's algorithm gives variable-length code.

```
HUFFMAN (C)
1.  n ← |C|          // n stores initial Queue size
2.  Q ← C            // initializes min-priority Queue (Q)
                        with the characters in C. Q is
                        implemented as a binary-min-heap.
3.  for i ← 1 to n-1
4.  {       do allocate a new node z
5.          left[z] ← x ← EXTRACT_MIN(Q)
6.          right[z] ← y ← EXTRACT_MIN(Q)
7.          f[z] ← f[x] + f[y]
8.          INSERT(Q, z)
9.  }   return ( EXTRACT_MIN(Q) )
```

Points -

(a) steps 3 to 8 do, algorithm creates a new node Z, and stores minimum frequency node as left child of Z, stores next minimum frequency node as right child of Z, the frequency of new node Z is the sum of the frequencies of x and y nodes

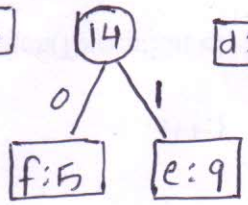
(b) after n-1 iteration only one node remains (root).

HUFFMAN CODES

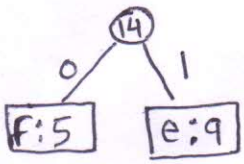
EXAMPLE

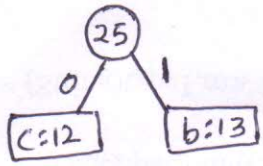
(a) f:5 | e:9 | c:12 | b:13 | d:16 | a:45

(b) c:12 | b:13 | 14 | d:16 | a:45

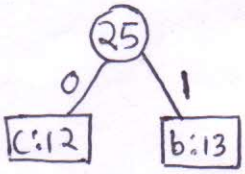


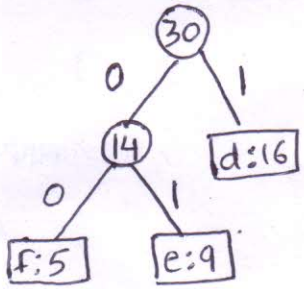
(c) 14 | d:16 | 25 | a:45



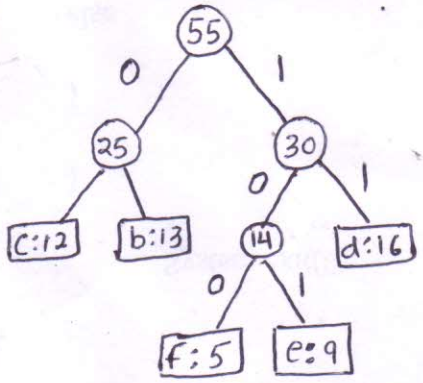


(d) 25 | 30 | a:45

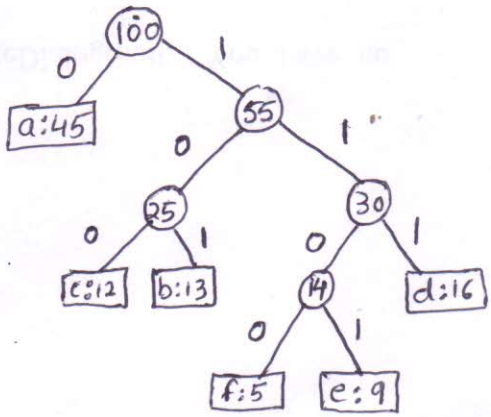




(e) a:45



(f)



Analysis of running time

Line 2 does initialization of Q, takes O(n) time using BUILD-MIN-HEAP Procedure.
for loop (Line 3 to 8) executes (n-1) times, since each heap operation requires time O(lgn) the loop takes O(nlgn).

Thus, $T(n) = O(n \lg n)$

BUILDING A MIN HEAP

The following algorithm build the MIN-HEAP.

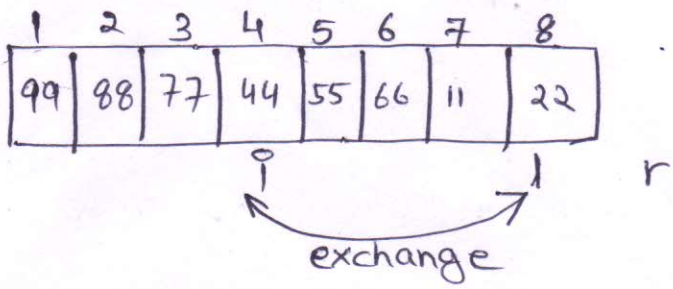
```
BUILD-MIN-HEAP (A)
1.  heap-size[A] ← length[A]
2.  for i ← ⌊length[A]/2⌋ downto 1
3       MIN-HEAPIFY (A, i)

MIN-HEAPIFY (A, i)
1.  l ← LEFT(i)          // returns 2i
2   r ← RIGHT(i)         // returns 2i+1
3   if l ≤ heap-size[A] and A[i] > A[l]
4   { smallest ← l }
5   else
6   { smallest ← i }
7   if r ≤ heap-size[A] and A[r] < A[smallest]
8   { smallest ← r }
9   if i ≠ smallest
10  { A[i] ↔ A[smallest]
11      MIN-HEAPIFY (A, smallest) }
```

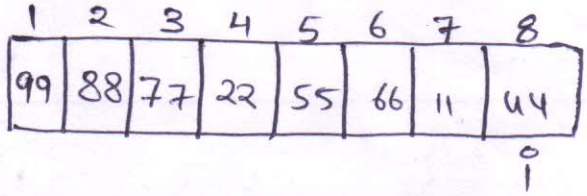
HUFFMAN CODES

BUILD-MIN-HEAP(A)

MIN-HEAPIFY(A,4)

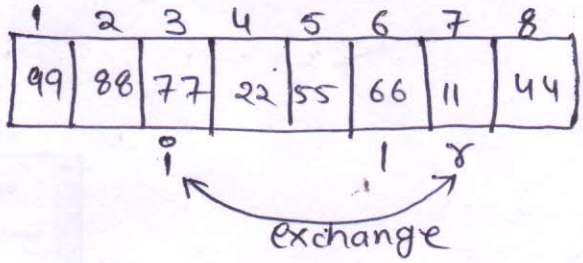


MIN-HEAPIFY(A,8)

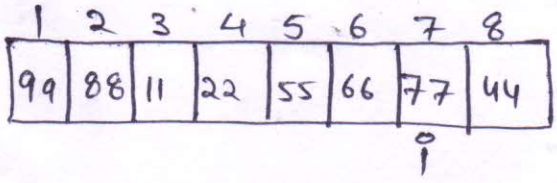


BUILD-MIN-HEAP(A)

MIN-HEAPIFY(A,3)

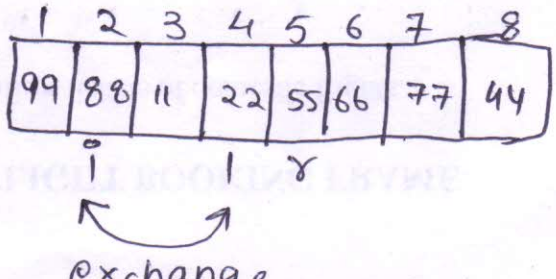


MIN-HEAPIFY(A,7)

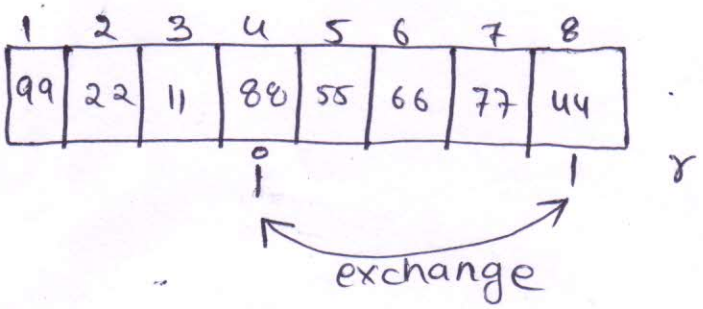


BUILD-MIN-HEAP(A)

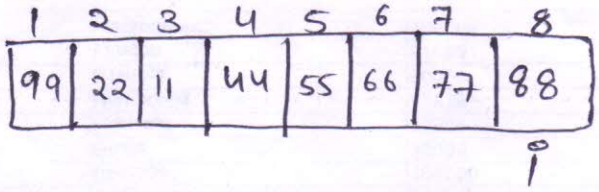
MIN-HEAPIFY(A,2)



MIN-HEAPIFY (A, 4)



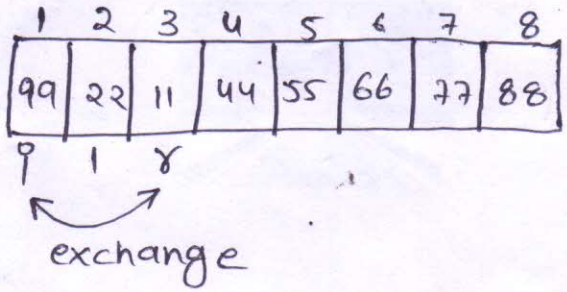
MIN-HEAPIFY (A, 8)



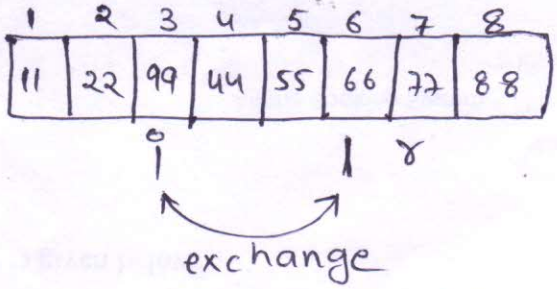
no l, r

BUILD-MIN-HEAP(A)

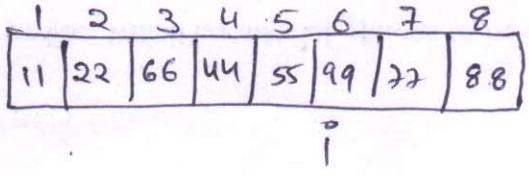
MIN-HEAPIFY(A, 1)



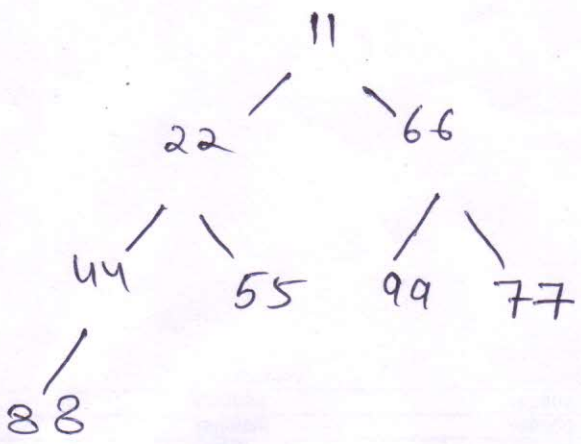
MIN- HEAPIFY (A, 3)



MIN- HEAPIFY(A, 6)



no l, r



Final Min-HEAP