COL 106 Lecture 15

Topics: Huffman Coding – Algorithms
Heap Sort

Announcement

Deadline for Assignment 1 regrade requests:

~~Sep 9~~ Sep 10, 23:00

Recap: Huffman Coding – Recursive Algorithm:

0. If $S = \{a\}$ (ie $|S| = 1$) return a tree with a single key a.
1. Let a_i, a_j be two least freq. letters. — $O(n)$
2. Construct a new instance with — $O(n)$

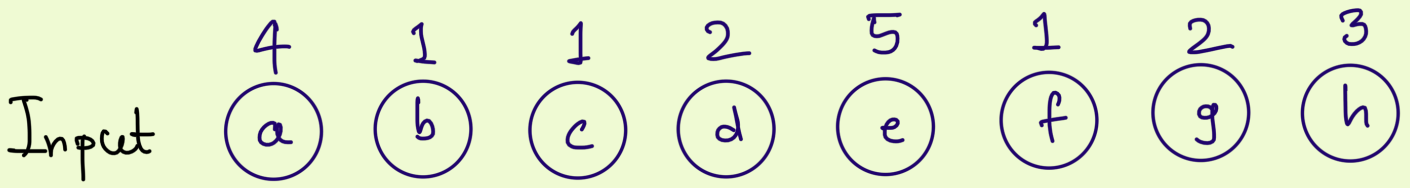
$S' = (S \setminus \{a_i, a_j\}) \cup \{u\}$, $p_k' = p_k$ for $k \neq i, j$

$p_u' = p_i + p_j$

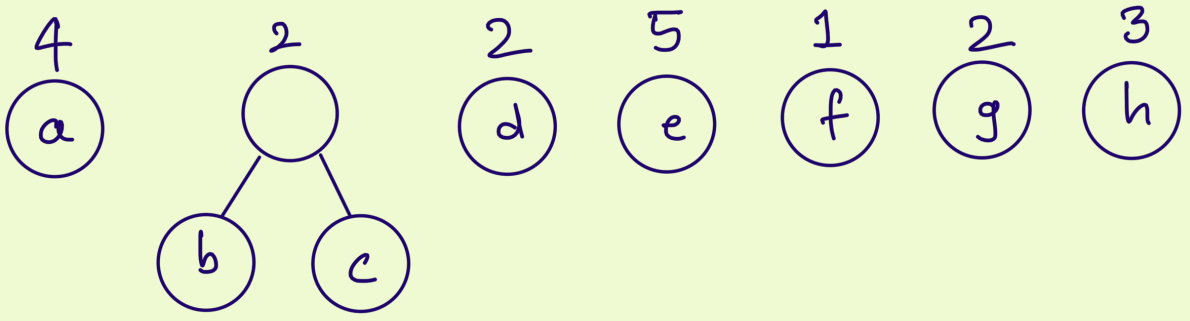
3. Recursively find the best tree, say T' for the new instance. (T' has u as a leaf) — $T(n-1)$
   — find u: $O(n)$
4. Attach a_i, a_j as children of u, and return this tree.

Running time: $T(n) = T(n-1) + O(n) \therefore T(n) = O(n^2)$

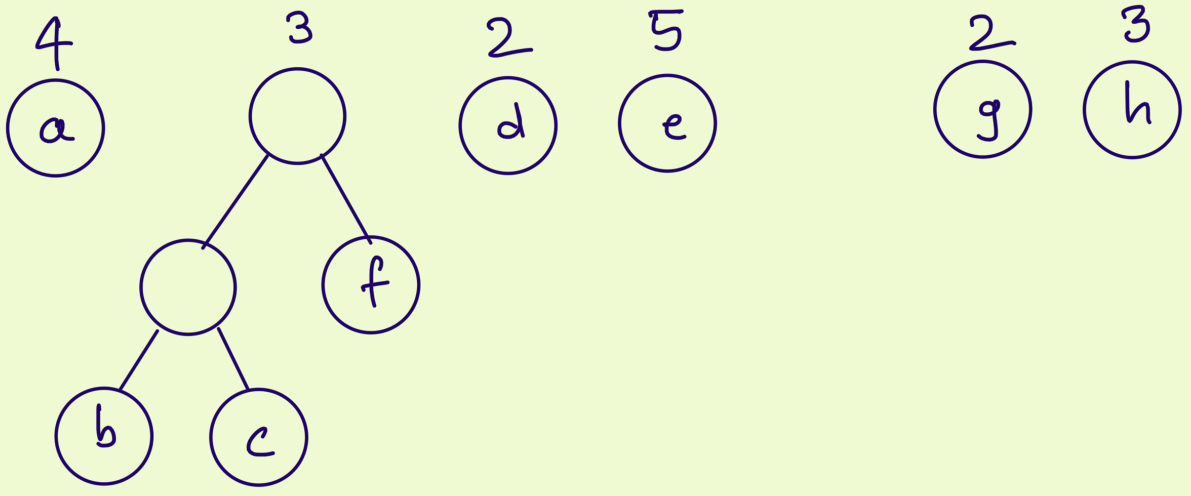
Example run



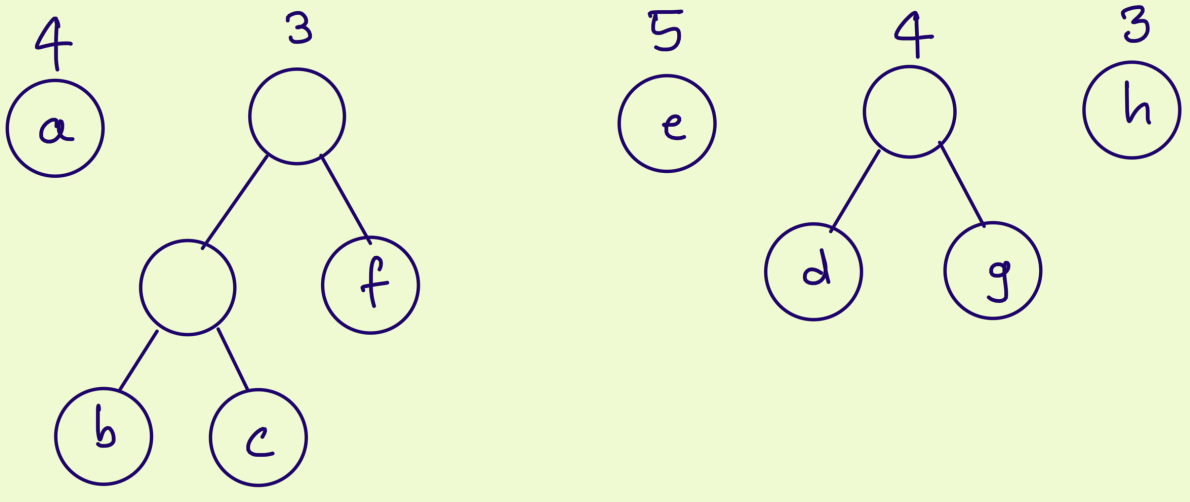
Step 1



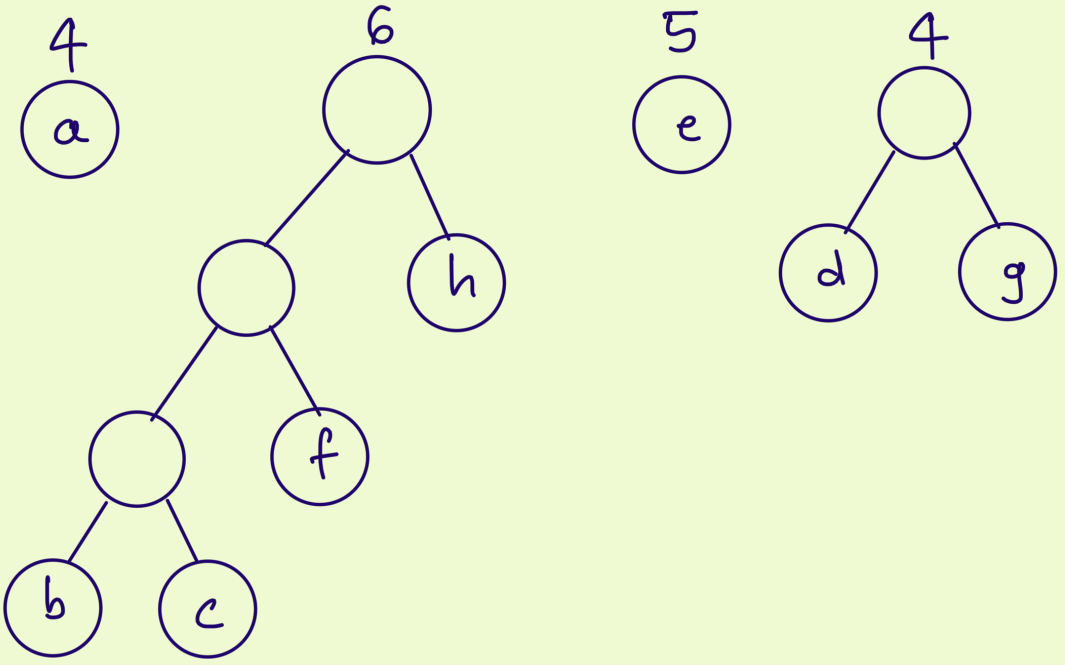
Step 2



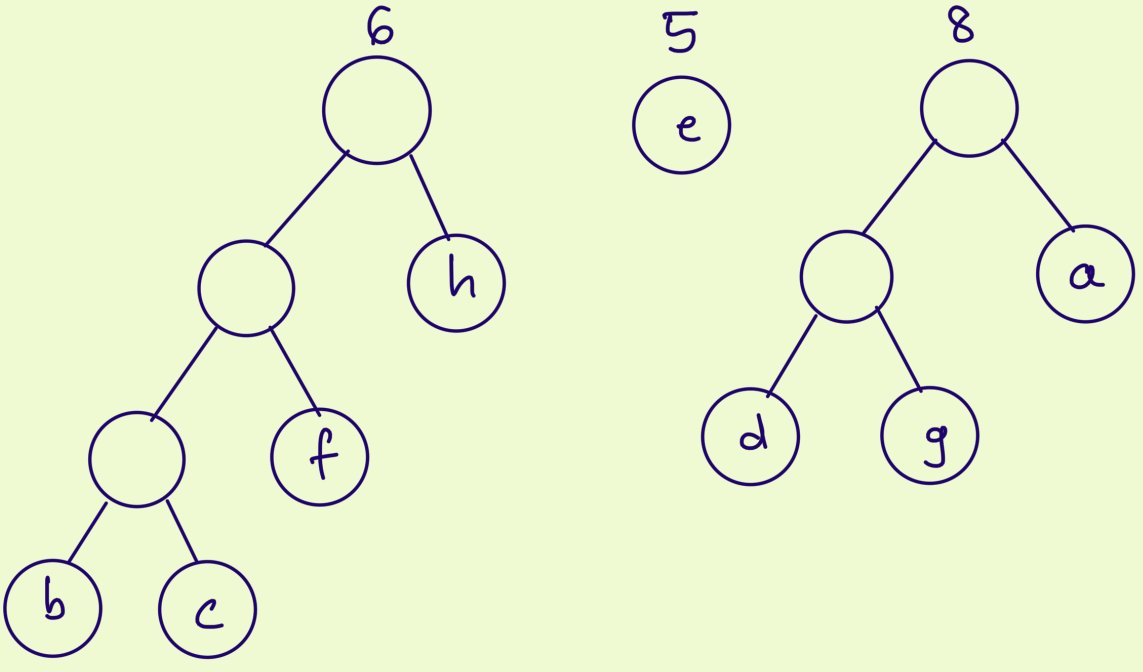
Step 3



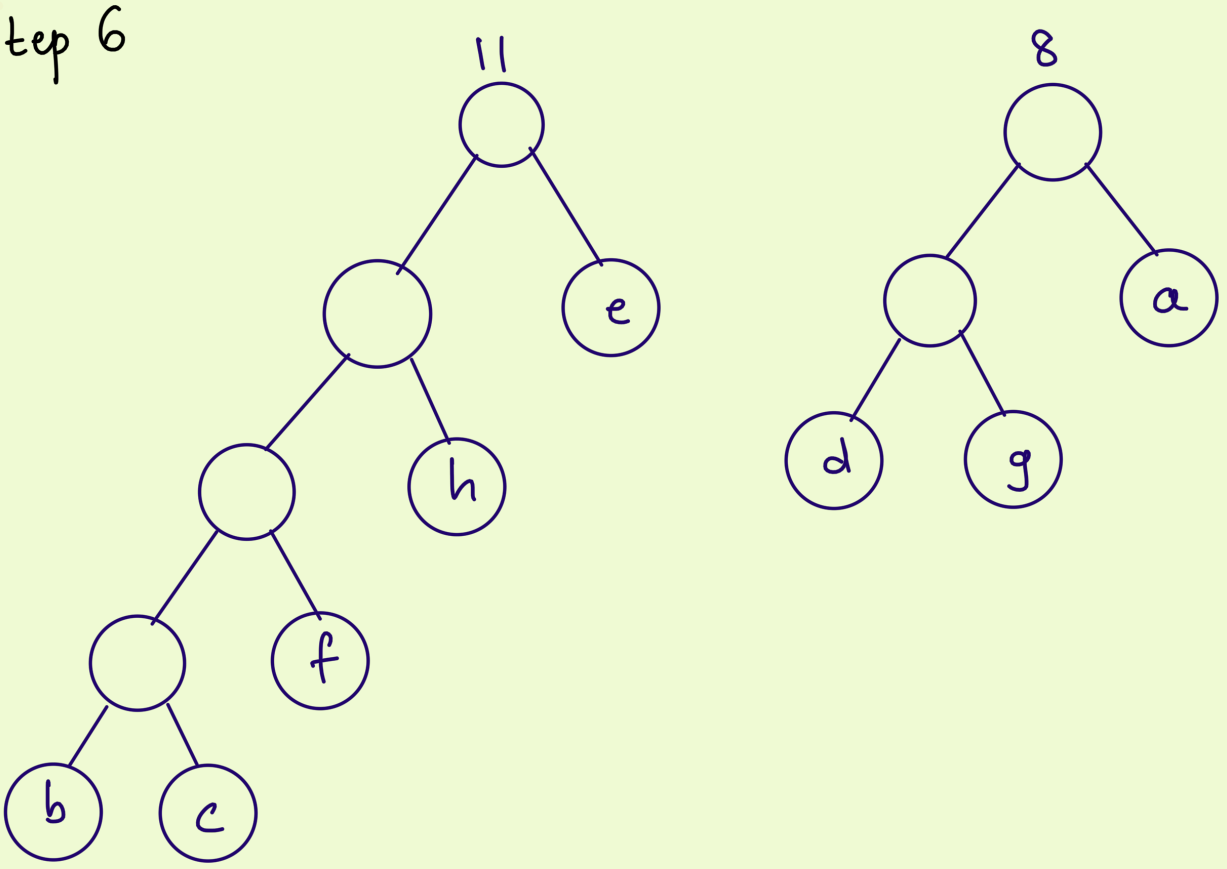
Step 4



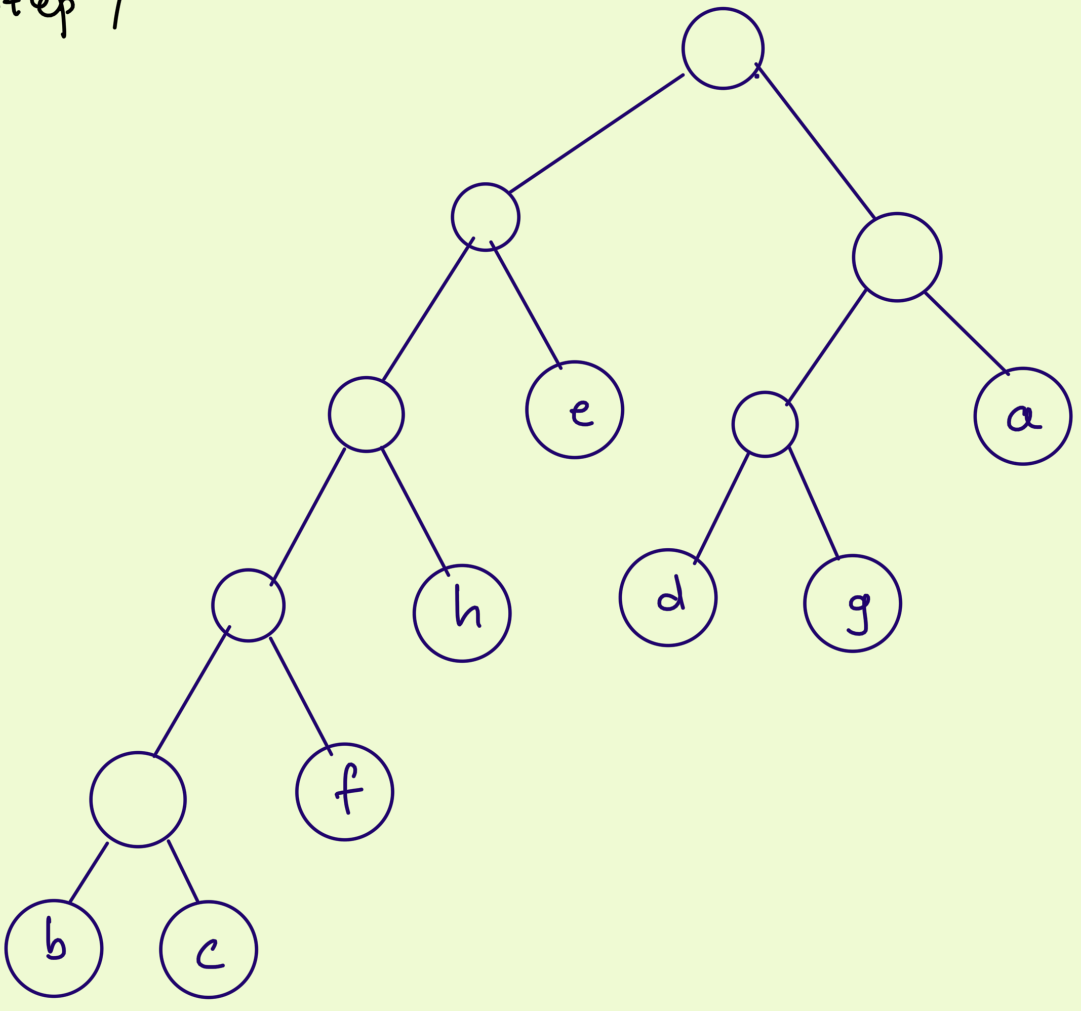
Step 5



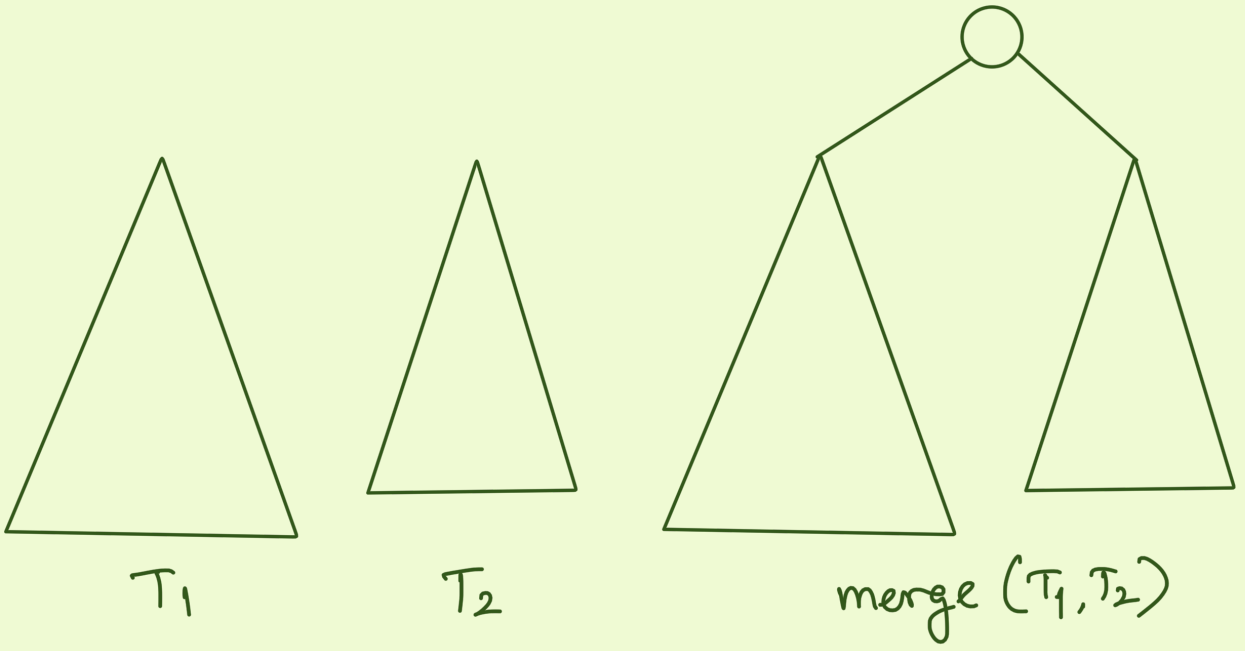
Step 6



Step 7



The "Merge" Operation



Equivalent iterative algorithm.

1. For each letter $a \in S$:
   Create tree T_a with single node a.
   $O(n)$ Add T_a to F.
   $p(T_a) \leftarrow p_a$.

2. While $|F| > 1$: → $O(n)$ iterations
   $T_1, T_2 \leftarrow$ two trees in F with least $p(\cdot)$, → $O(n)$
   $T' \leftarrow$ merge (T_1, T_2). — $O(1)$
   $p(T') = p(T_1) + p(T_2)$ — $O(1)$
   $F \leftarrow (F \setminus \{T_1, T_2\}) \cup \{T'\}$. → $O(n)$

Running Time : $O(n^2)$.

Faster Algorithm using Heaps

1. For each letter $a \in S$:
   Create tree T_a with single node a. — $O(n)$
   Set $p(T_a) = p_a$.
2. $H \leftarrow$ BuildHeap$(\{(T_a, p(T_a)) \mid a \in S\})$. — $O(n)$
3. While H contains > 1 elements: — $O(n)$ iterations
   $(T_1, p_1) \leftarrow H.$ ExtractMin(). — $O(\log n)$
   $(T_2, p_2) \leftarrow H.$ ExtractMin() — $O(\log n)$
   $T' \leftarrow$ merge (T_1, T_2). — $O(1)$
   H. Enqueue $((T', p_1 + p_2))$ — $O(\log n)$
4. Return H. ExtractMin(). — $O(1)$.

Running time: $O(n \log n)$.

Sorting Algorithm	Worst-case Time Complexity	Worst-Case Extra Space
Bubble, Selection, Insertion	$O(n^2)$	$O(1)$
Merge Sort	$O(n \log n)$	$O(n)$
Quicksort	$O(n^2)$?	?
	$O(n \log n)$ ↑ if median* is chosen as pivot	

* Median can be found in $O(n)$ time; this is out of scope for COL106.

Yet Another Sorting Algorithm : Heapsort.

```
HeapSort (l):
    H ← BuildHeap (l)
    While H not empty:
        Print / Output / Append to list / Yield    H.ExtractMin().
```

Running Time : $O(n \log n)$

Claim: HeapSort can be performed in $O(1)$ extra space (and $O(n \log n)$ time)

```
In-place Heapsort (l):
    Make l a max-heap in-place        Using better buildheap O(n)
    For i = size(l) − 1 to 0:
        ( Invariant : l[0..i] is a max heap; l[i+1..] contains
          the size(l) − i largest elements of l in inc. order.)
        Exchange l[0] and l[i]
        Make l[0, ..., i-1] a heap using HeapDown.
```