

# *Data Structures and Algorithms*

## **Huffman Encoding and Decoding**

**PLSD210(ii)**

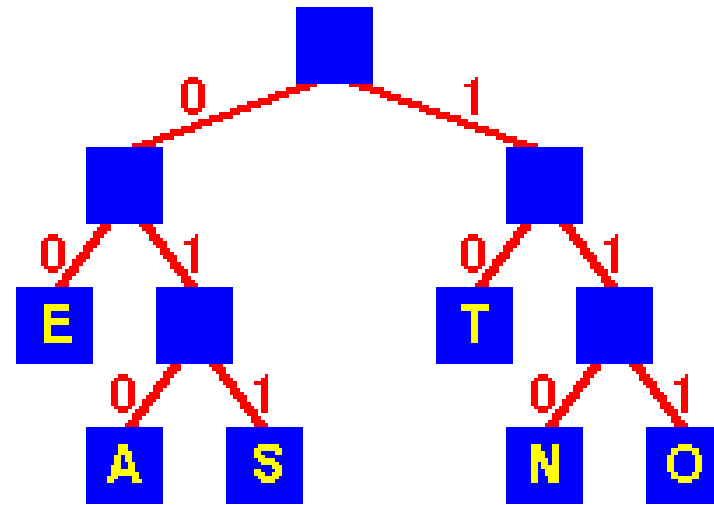
# Huffman Encoding

- **Compression**
  - Typically, in files and messages,
    - Each character requires 1 byte or 8 bits
    - Already wasting 1 bit for most purposes!
- **Question**
  - What's the smallest number of bits that can be used to store an arbitrary piece of text?
- **Idea**
  - Find the frequency of occurrence of each character
  - Encode Frequent characters      **short bit strings**
  - Rarer characters      **longer bit strings**

# Huffman Encoding

- **Encoding**

- Use a tree
- Encode by following tree to leaf
- eg
  - E is 00
  - S is 011
- Frequent characters  
E, T 2 bit encodings
- Others  
A, S, N, O 3 bit encodings



# Huffman Encoding

- **Encoding**
  - Use a tree
    - Inefficient in practice
  - Use a direct-addressed lookup table

## ? Finding the optimal encoding

- Smallest number of bits to represent arbitrary text

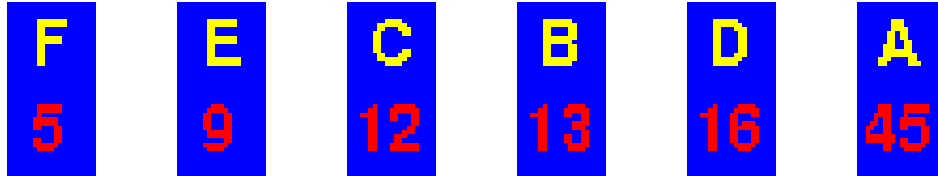
A	010
B	
:	
E	00
:	
N	110
:	
S	001
T	10

# Huffman Encoding

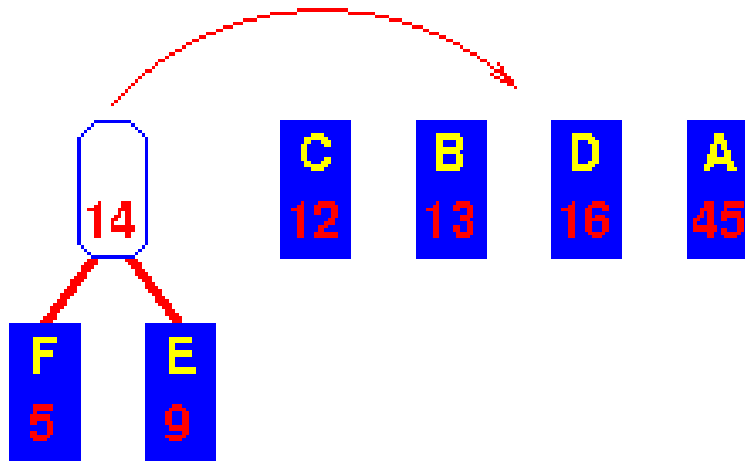
- **Divide and conquer**
  - **Decide on a root -  $n$  choices**
  - **Decide on roots for sub-trees -  $n$  choices**
  - **Repeat  $n$  times**
    - **$\blacktriangleright O(n!)$**
- **Greedy Approach**
  - **Sort characters by frequency**
  - **Form two lowest weight nodes into a sub-tree**
    - **Sub-tree weight = sum of weights of nodes**
  - **Move new tree to correct place**

# Huffman Encoding - Operation

Initial sequence  
Sorted by frequency



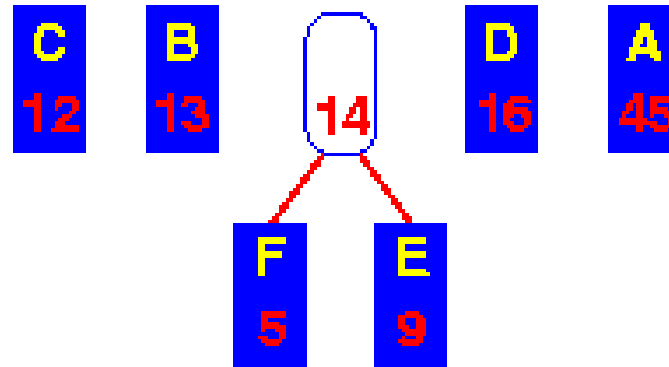
Combine lowest two  
into sub-tree



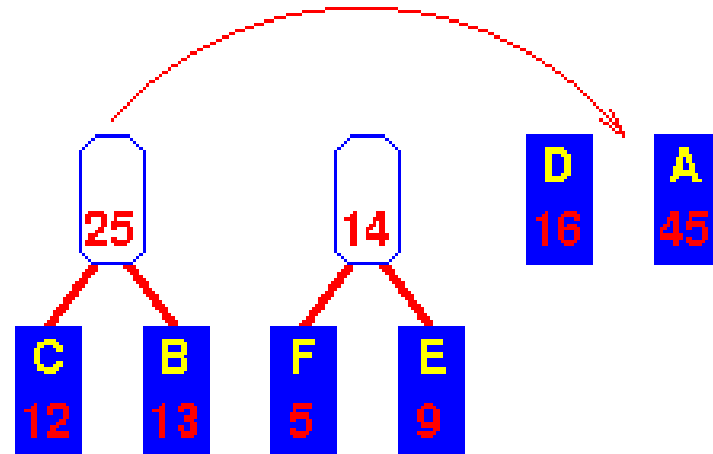
Move it to correct  
place

# Huffman Encoding - Operation

After shifting sub-tree to its correct place ...



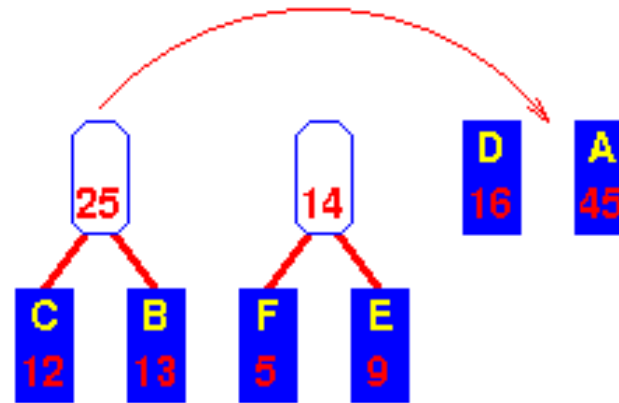
Combine next lowest pair



Move sub-tree to correct place

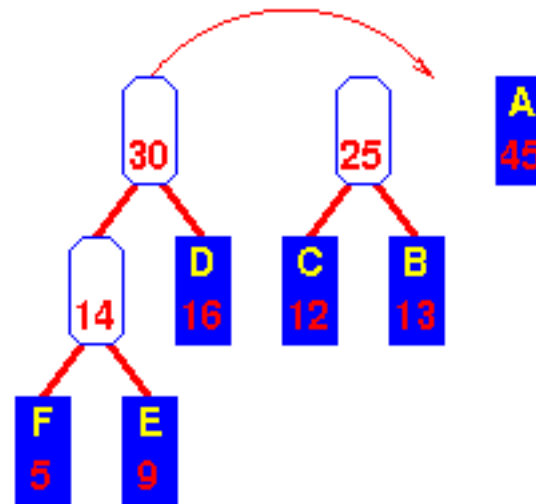
# Huffman Encoding - Operation

Move the new tree to the correct place ...

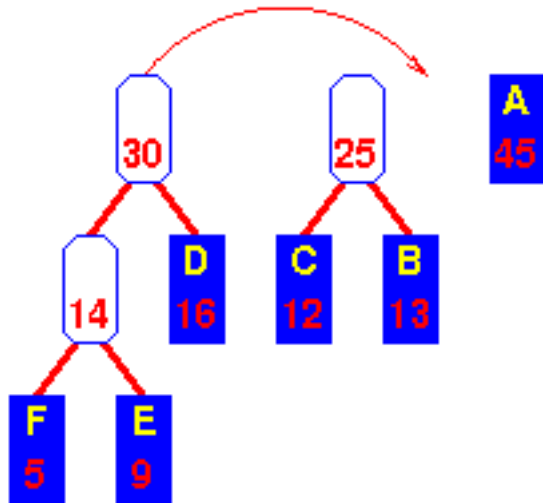


Now the lowest two are the "14" sub-tree and D

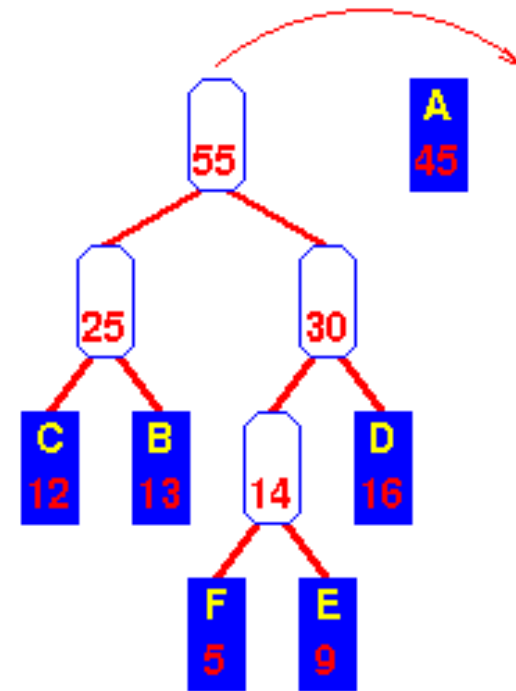
Combine and move to correct place



# Huffman Encoding - Operation



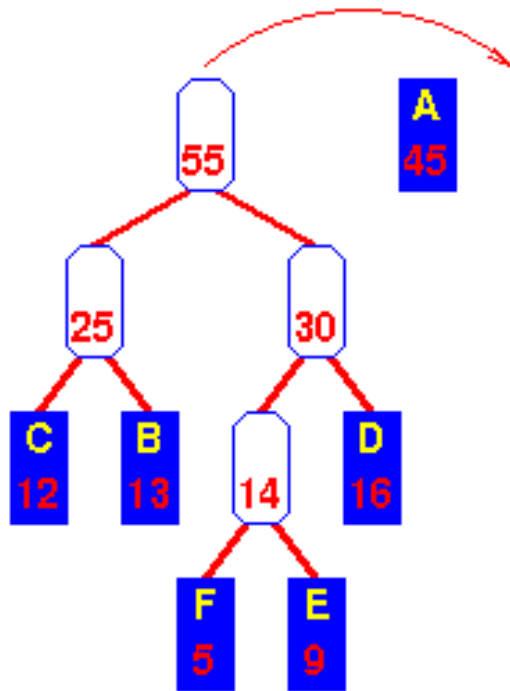
Move the new tree to the correct place ...



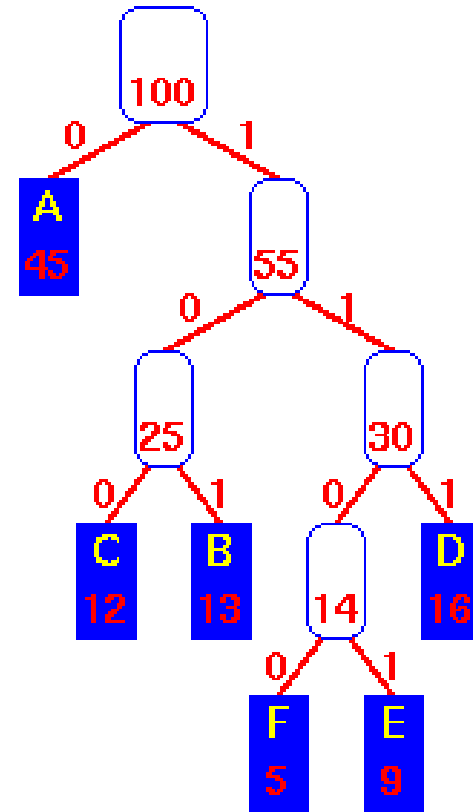
Now the lowest two are the the “25” and “30” trees

Combine and move to correct place

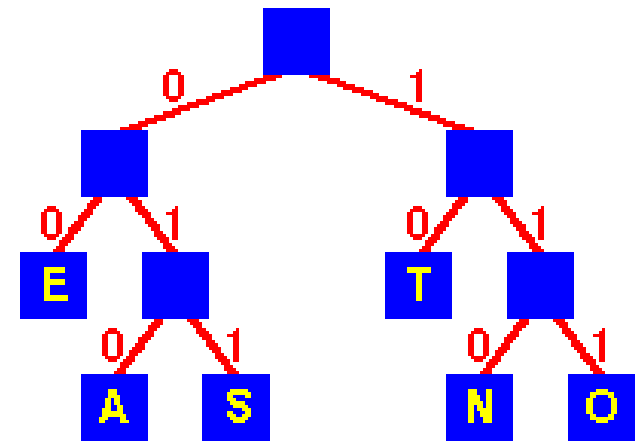
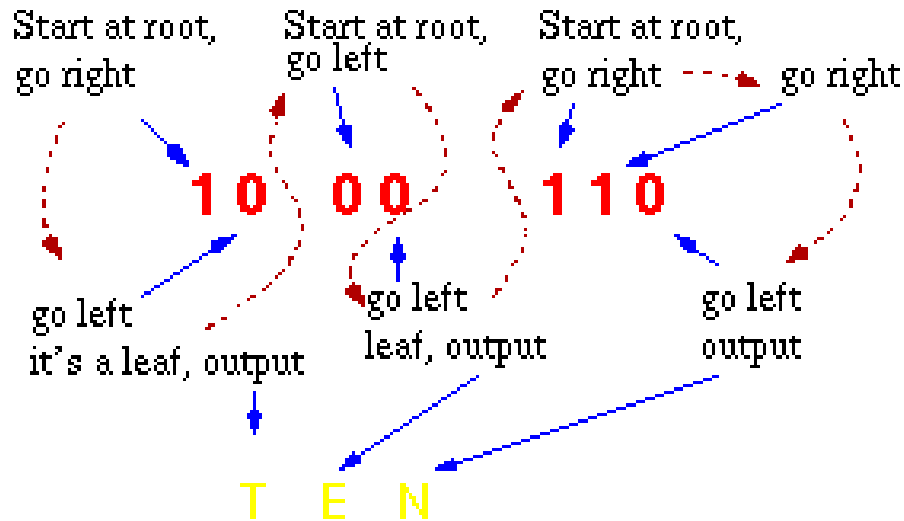
# Huffman Encoding - Operation



Combine  
last two trees



# Huffman Encoding - Decoding



# *Huffman Encoding - Time Complexity*

- **Sort keys**  $O(n \log n)$
- **Repeat  $n$  times**
  - **Form new sub-tree**  $O(1)$
  - **Move sub-tree**  $O(\log n)$   
(binary search)
  - **Total**  $O(n \log n)$
- **Overall**  $O(n \log n)$