


**Exercise 7 (for grade)** ~ Wednesday, November 2, 2022 ~ CPSC 535.01 Fall 2022

Write one submission for your entire group, and write all group members' names on that submission. Turn in your submission before the end of class. The  symbol marks where you should write answers.

Recall that our recommended problem-solving process is:

1. **Understand** the problem definition. What is the input? What is the output?
2. **Baseline** algorithm for comparison
3. **Goal** setting: improve on the baseline how?
4. **Design** a more sophisticated algorithm
5. **Inspiration** (if necessary) from patterns, bottleneck in the baseline algorithm, other algorithms
6. **Analyze** your solution; goal met? Trade-offs?

Follow this process for each of the following computational problems. For each problem, your submission should include:

- a. State are the input variables and what are the output variables
- b. Pseudocode for your baseline algorithm, that needs to include the data type and an explanation for any variable other than input and output variables
- a. The  $\Theta$ -notation time complexity of your baseline algorithm, with justification.

and if you manage to create an improved algorithm:

- c. Answer the question: how is your improved algorithm different from your baseline; what did you change to make it faster?
- d. Pseudocode for your improved algorithm, that needs to include the data type and an explanation for any variable other than input and output variables
- a. The  $\Theta$ -notation time complexity of your improved algorithm, with justification.

Today's problems are:

1.

Compatible intervals: Given 10 open intervals  $(a_1, b_1), (a_2, b_2), \dots, (a_{10}, b_{10})$  on the real line, each representing start and end times of some activity requiring the same resource, the task is to find the largest number of these intervals so that no two of them overlap.

a[]	1	6	3	8	11	8	4	5	11	14
b[]	2	7	9	10	12	12	7	8	13	15

Show the schedule for each of the three greedy algorithms based on:

- earliest start first.
- shortest duration first.
- earliest finish first.

For each of the three algorithms, state whether the algorithm yields an optimal solution or not.

2.

Given the following set of frequencies:

a:13      b:21    c:35    d:38    e:74    f:112    g:189

- Write the fixed-length code for each character
- Write the Huffman code for each character. Show your work on computing Huffman code.

3.

If the symbols of the alphabet have their frequencies in increasing order then prove that the algorithm shown in class that builds Huffman tree has linear time complexity. An example is Exercise 2, above.

## Names

Write the names of all group members below.

 VaishnaviBacha

Niharika Velidhi

Rosa cho

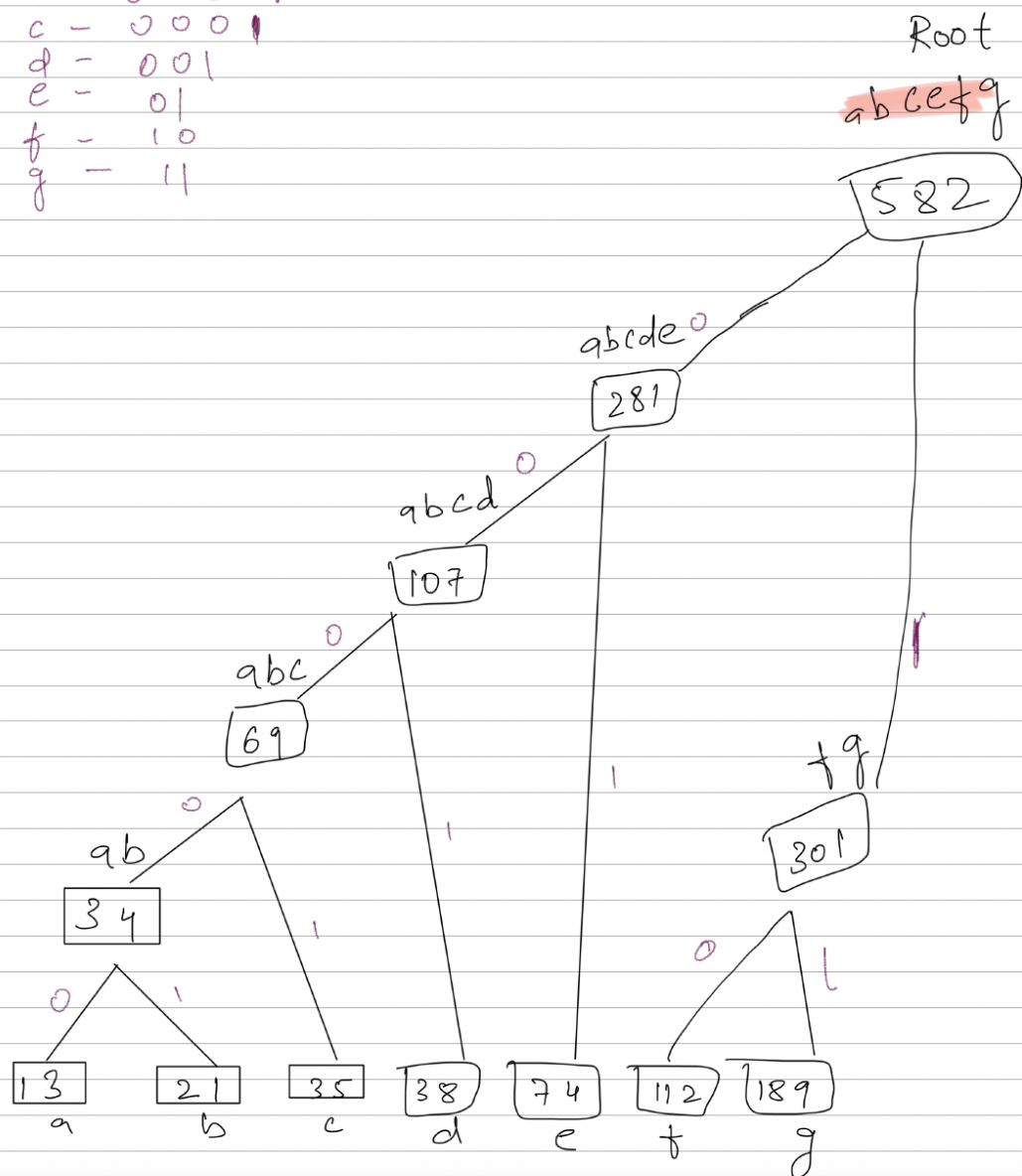
## Exercise 1: Solve and provide answer



## Exercise 2: Solve and provide answer

	a	b	c	d	e	f	g
frequency	13	21	35	38	74	112	189
fixed length	000	001	010	011	100	101	111
variable-length	00000	00001	0001	001	01	10	11

a - 00000  
 b - 00001  
 c - 0001  
 d - 001  
 e - 01  
 f - 10  
 g - 11





### Exercise 3: Solve and provide answer

✖ The Huffman coding for the alphabet with their frequencies in increasing order would be:

Input: alphabet (a, b, c,...), increased frequency (integers)

Output: Huffman Tree

1. character	frequency
a	1
b	2
c	3
d	4
e	5
↓	
<u>Σ</u>	26

We can already see a linear relationship between each character & frequency.  
a  
freq = n

#### 2. Pseudocode

Q.build(c) //We build a min-priority queue for frequencies

```
for int i = 1 to n-1:      //We look through each value in the queue Q and find the minimum value0
    x = Q.findMin()        //this function already runs at O(n) runtime
    y = Q.findMin()
    z.setLeft(x)           //create a leaf node and then insert it into the queue
    z.setRight(x)
    z.set(x.f() + y.f())    //combine the two smallest nodes to get a new one
    Q.insert(z)            //add new node to the queue
return Q.findMin()         //return min-priority queue
```

3. To find the smallest value, we get a(n=1), then we join, add up the values. Even though the order is already sorted, we are faced with the following dilemma. Since the value may not be the smallest, we join the next smallest and continue to do so in order to maintain the order of 1 of n values.

Therefore, the runtime is expected to be  $O(n)$ .