NANYANG TECHNOLOGICAL UNIVERSITY

SEMESTER II EXAMINATION 2017-2018 SUGGESTED SOLUTION

MH1402 – Algorithms & Computing II

MAY 2018 TIME ALLOWED: 2 HOURS

INSTRUCTIONS TO CANDIDATES

1. This examination paper contains **SEVEN (7)** questions and comprises **FOUR (4)** printed pages.
2. Answer all questions. The marks for each question are indicated at the beginning of each question.
3. Answer each question beginning on a **FRESH** page of the answer book.
4. This **IS NOT** an **OPEN BOOK** exam.
5. Candidates may use calculators. However, they should write down systematically the steps in the workings.

Solutions provided by: Brandon Goh – bgoh008@e.ntu.edu.sg

Question 1.

(a) Show by mathematical induction that $T(n) = 2^{n+1} - 1$, given the following definition: **(6 marks)**

$$T(n) = \begin{cases} 1 & \text{if } n = 0, \\ T(n-1) + 2^n & \text{otherwise.} \end{cases}$$

(b) What is the complexity of the following piece of Python code in Big-Oh notation? **5 marks**

```
sum = 0
for i in range(n**2):
    for j in range(i):
        sum += j
```

Answer

(a) Take the LHS as the given equation and RHS as $T(n) = 2^{n+1} - 1$.

Base Case $n = 0$

(LHS) $T(0) = 1$
(RHS) $T(0) = 2^1 - 1 = 1$

LHS=RHS, $\therefore$ Base case is true.

Hypothesis $n = k$

Assume that $n = k$ is true, then

$$T(k) = T(k-1) + 2^k = 2^{k+1} - 1$$

Induction $n = k + 1$

To prove that $n = k + 1$ is true, LHS must be equal to RHS.

$$\begin{aligned}
\text{(RHS) } T(k+1) &= 2^{k+2} - 1 \\
\text{(LHS) } T(k+1) &= T(k) + 2^k \\
&= (2^{k+1} - 1) + 2^{k+1} \\
&= 2 \cdot 2^{k+1} - 1 \\
&= 2^{k+2} - 1 \\
&= \text{(RHS)}
\end{aligned}$$

From the above, $n = 0$ and $n = k$ are true
$\Rightarrow n = k + 1$ is also true. Then by mathematical induction, the equation is true for all $n \in \mathbb{N}_0$. □

(b) Break the question into 2 blocks:

Block 1:
`sum = 0` $\rightarrow$ 1 operation.

Block 2:
```
for i in range(n**2):
   for j in range(i):
      sum += j
```

It is important to note that `sum += j` constitutes as 1 operation per execution.

Start by considering small cases.

(1) When $n = 0$, there is no execution of the **outer** loop.

(2) When $n = 1$, $i = 0$ only $\Rightarrow$ there is no execution of **inner** loop.

(3) When $n = 2$, $i \in \{0, 1, 2, 3\}$

- `j in range(0)` $\Rightarrow$ inner loop is not executed.
- `j in range(1)` $\Rightarrow$ inner loop is executed **once**.
- `j in range(2)` $\Rightarrow$ inner loop is executed **twice**.
- `j in range(3)` $\Rightarrow$ inner loop is executed **thrice**.

Total executions: $\sum_{x=1}^{3} x = 6$

(c) When $n = 3$, $i \in \{0, 1, 2, 3, 4, 5, 6, 7, 8\}$

- `j in range(0)` $\Rightarrow$ inner loop is not executed.
- `j in range(1)` $\Rightarrow$ inner loop is executed **once**.
- `j in range(2)` $\Rightarrow$ inner loop is executed **twice**.
- $\vdots$
- `j in range(8)` $\Rightarrow$ inner loop is executed **eight** times.

Total executions: $\sum_{x=1}^{8} x = 36$

Using the inner loops, we determine that the number of executions can be simplified to the following:

$$\sum_{x=1}^{\max(i)} x$$

Now consider the outer loop, the number of executions of the inner loop is based on the value of n. Furthermore, i depends on n. The formula for the total number of executions can be written in the following form.

$$\sum_{j=1}^{n} \sum_{x=1}^{j^2-1} x$$

Solving the double summation yields the following the result.

$$\begin{aligned}\sum_{j=1}^{n} \sum_{x=1}^{j^2-1} x &= \sum_{j=1}^{n} \frac{(j^2-1)(j^2)}{2} \\ &= \frac{1}{20}(n-1)(n)(n+1)(n+2)(2n+1) \\ &= O(n^5)\end{aligned}$$

Note: You are **NOT** required to know the result from the double summation. Instead, you are expected to know that $\sum_{x=1}^{n} x^a = O(x^{a+1})$ where $a \in \mathbb{N}_0$.

The loop gives us a time complexity of $O(n^5)$, so the addition of the 1 operation from outside the loop ($O(1)$) will not change the time complexity of the algorithm. $\square$

Question 2. Given the following binary search tree

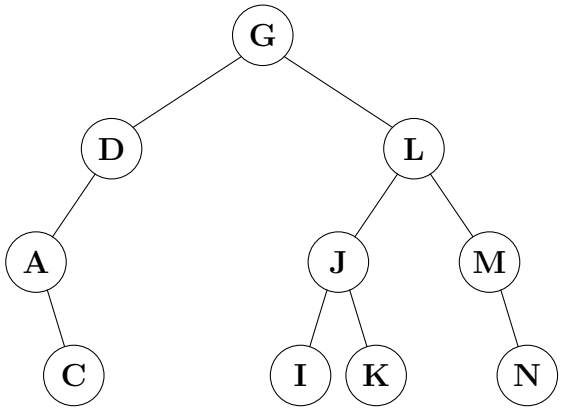


Figure 1: Binary Search Tree

(a) What is the sequence of nodes by Euler's Traversal? **(8 marks)**

(b) Draw the resultant tree after removing node "L". **(6 marks)**

Answer

(a) $G \to D \to A \to C \to A \to D \to G \to L \to J \to I \to J \to K \to J \to L \to M \to N \to M \to L \to G$ □

(b) Step 1: L has 2 child nodes, so the key must be swapped with its **successor**.

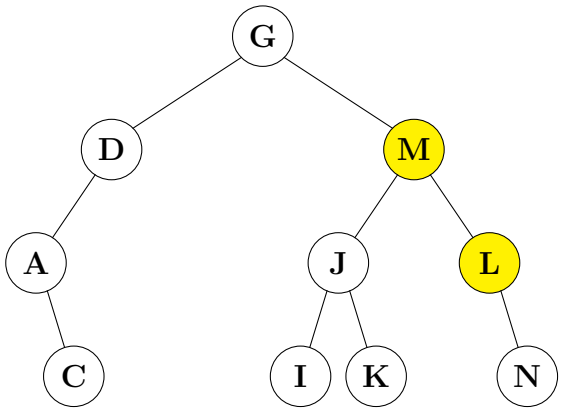


Figure 2: Binary Search Tree After Step 1

Step 2: L now has 1 child node, so swap the key with the child.

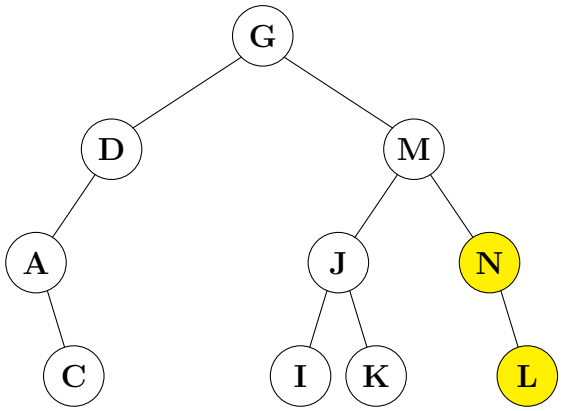


Figure 3: Binary Search Tree After Step 2

Step 3: L has no child nodes, so remove the node from the BST.

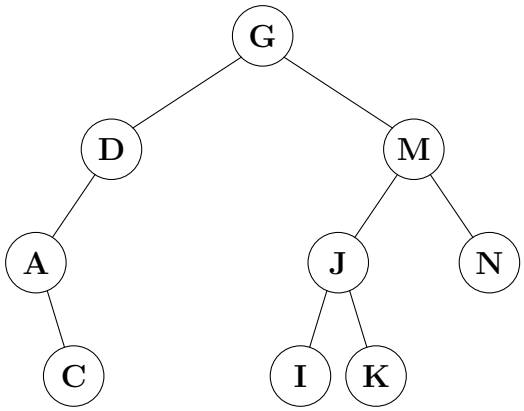


Figure 4: Final Binary Search Tree

□

Question 3. Given the node class implementation below:

```
class Node:
    def __init__(self, initElement):
```

```
        self.left = None                #left child
        self.right = None               #right child
        self.parent = None              #parent
        self.element = initElement
```

write in Python a *function* to print the element of all nodes of a binary tree in the order of depth, i.e., the node of depth 0 first (the root), followed by all the nodes of depth 1 (children of the root), then all the nodes of depth 2 (grandchildren of the root) ..., until all the nodes are printed. The function takes the root node of the binary tree as input. You may assume the Queue class is available for use, including functions like:

- Queue(): instantiate an empty queue,
- enqueue(Node x): enqueue node x,
- dequeue(): dequeue and returns the node removed from queue,
- isEmpty(): returns *True* if queue is empty, *False* otherwise.

Note the input binary tree may or may not be a proper binary tree.

(12 marks)

Answer

Since the question only requires us to print the nodes in order without any particular formatting, the code below satisfies the requirements.

```
def TreeOrder(root):
    A=Queue()                       #Create empty queue A
    A.enqueue(root)                 #Queue the root node
    while not A.isEmpty():          #The queue is not-empty,
                                    #so there are nodes that
                                    #still need to be processed
        P=A.dequeue()               #Remove node for processing
        print(P.element)            #Print element in the node
        if P.left!=None:
            A.enqueue(P.left)       #Queue left child node if exists
```

```
        if P.right!=None:
            A.enqueue(P.right)  #Queue right child node if exists
    return
```

□

Question 4. Consider the list L = [1, 4, 2, 8, 6, 3, 5].

(a) Apply merge-sort algorithm on the list L. Explain with a binary tree the successive recursive calls, and another binary tree the successive merge processes. Assume the left part takes half or one element less, i.e. $\lfloor n/2 \rfloor$ when dividing a sub-problem of size n. **(12 marks)**

(b) Apply quick-sort algorithm on the original list L. Explain with a binary tree the sort process and underline the pivots. Assume it is always the second element of current sub-problem selected as the pivot. **(12 marks)**

Answer

(a) First Binary Tree for Successive Recursive Calls

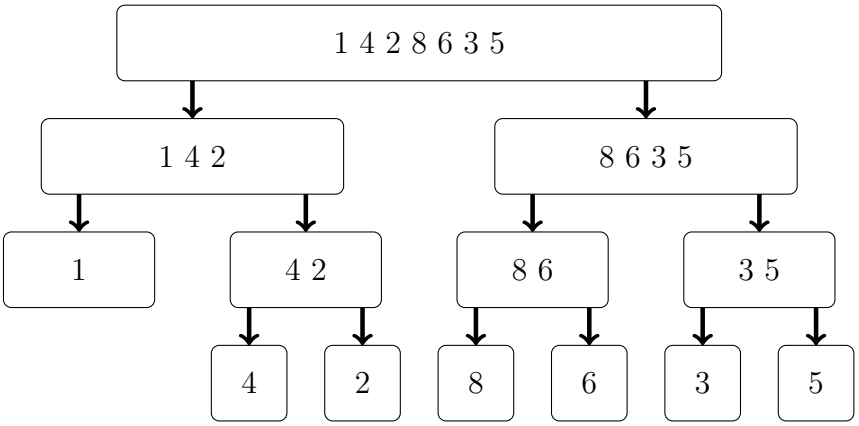


Figure 5: Binary Tree for Mergesort (Splitting)

Second Binary Tree for Sorting

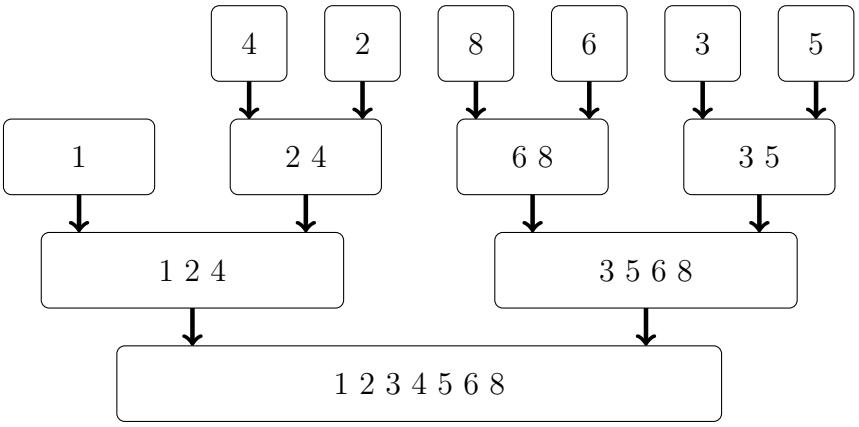


Figure 6: Binary Tree for Mergesort (Merging)

(b) Binary Tree for quicksort

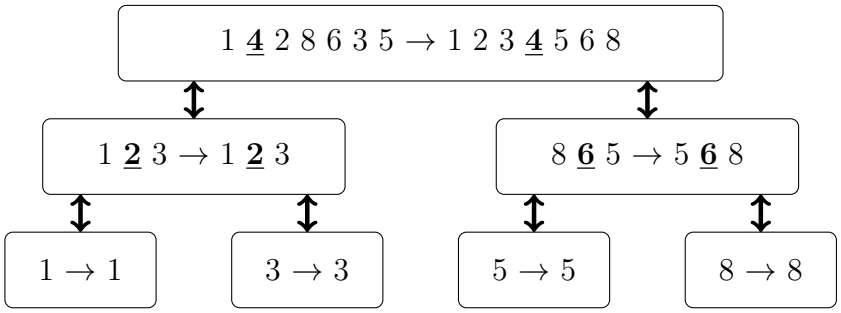


Figure 7: Binary Tree for Quicksort

Step (1)
Identify pivot

Step (2)
Split the values into the left and right subtrees respectively depending on whether the values are
less than (<) or greater than (>) the pivot (while preserving the order).

Step (3)
Arrays of length 1 are already sorted (trivial).

Step (4)
Combine after obtaining the results from the child nodes. □

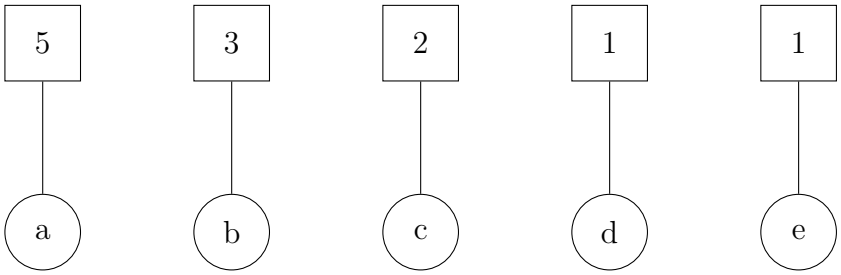
Question 5. Give the frequency table, Huffman tree, and the resulting table of code words of all characters for the string "aabaabccdbea". Note that subtree with more nodes is always placed on the right when joining two subtrees. **(15 marks)**

Answer

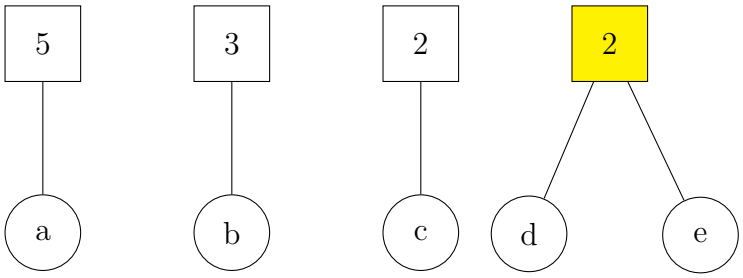
Frequency table:

a	b	c	d	e
5	3	2	1	1

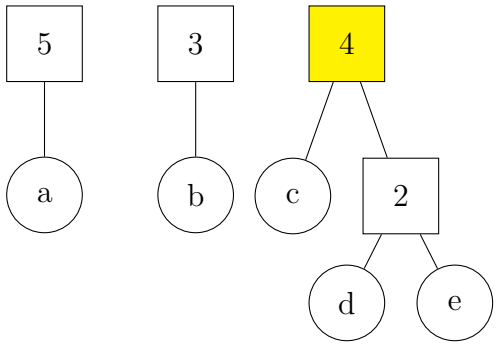
When drawn in its node form:



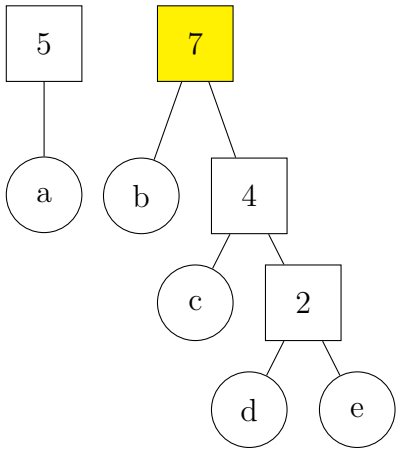
When computing the Huffman Tree, always combine the nodes of letters (or data) with the two lowest frequencies. In this case, d and e both have the lowest frequency of 1.



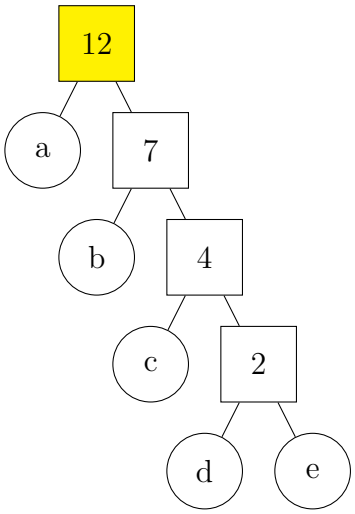
Now, c & $(d$ & $e)$ have the lowest frequencies. Therefore, we will combine these two branches.



Again, b & $(c$ & $(d$ & $e))$ have the lowest frequencies. Note that b has 3 nodes while $(c$ & $(d$ & $e))$ has 4 nodes. As such, b is on the left subtree.



Finally, a & $(b$ & $(c$ & $(d$ & $e)))$ are the last two roots of their respective subtrees. Both subtrees can be merged together, with a on the left subtree.



To determine the table of codewords, you must refer to the Huffman Tree. The codeword for each left edge is '0', while the right edge is '1'. For example, to get to the letter 'c', you must go down the edges in this order: right $\rightarrow$ right $\rightarrow$ left. Therefore, the codeword for $c \rightarrowtail 110$. Repeat for all characters and you will get the following codeword table.

a	b	c	d	e
0	10	110	1110	1111

□

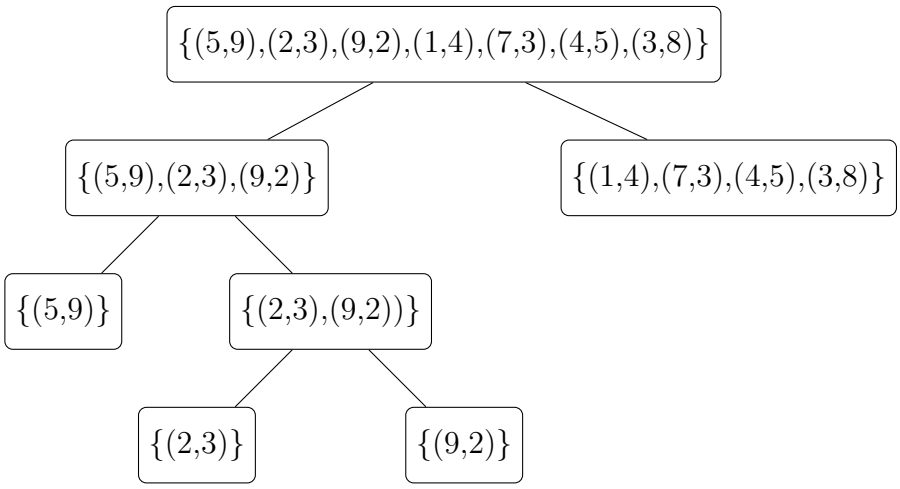
Question 6. What is the maximal set from the following set of points? Show your steps using a binary tree, that arises from the method of divide-and-conquer.

$$\{(5,9),(2,3),(9,2),(1,4),(7,3),(4,5),(3,8)\}$$

(12 marks)

Answer

Assume that the left subset will have $\lfloor n/2 \rfloor$ points.



END OF PAPER