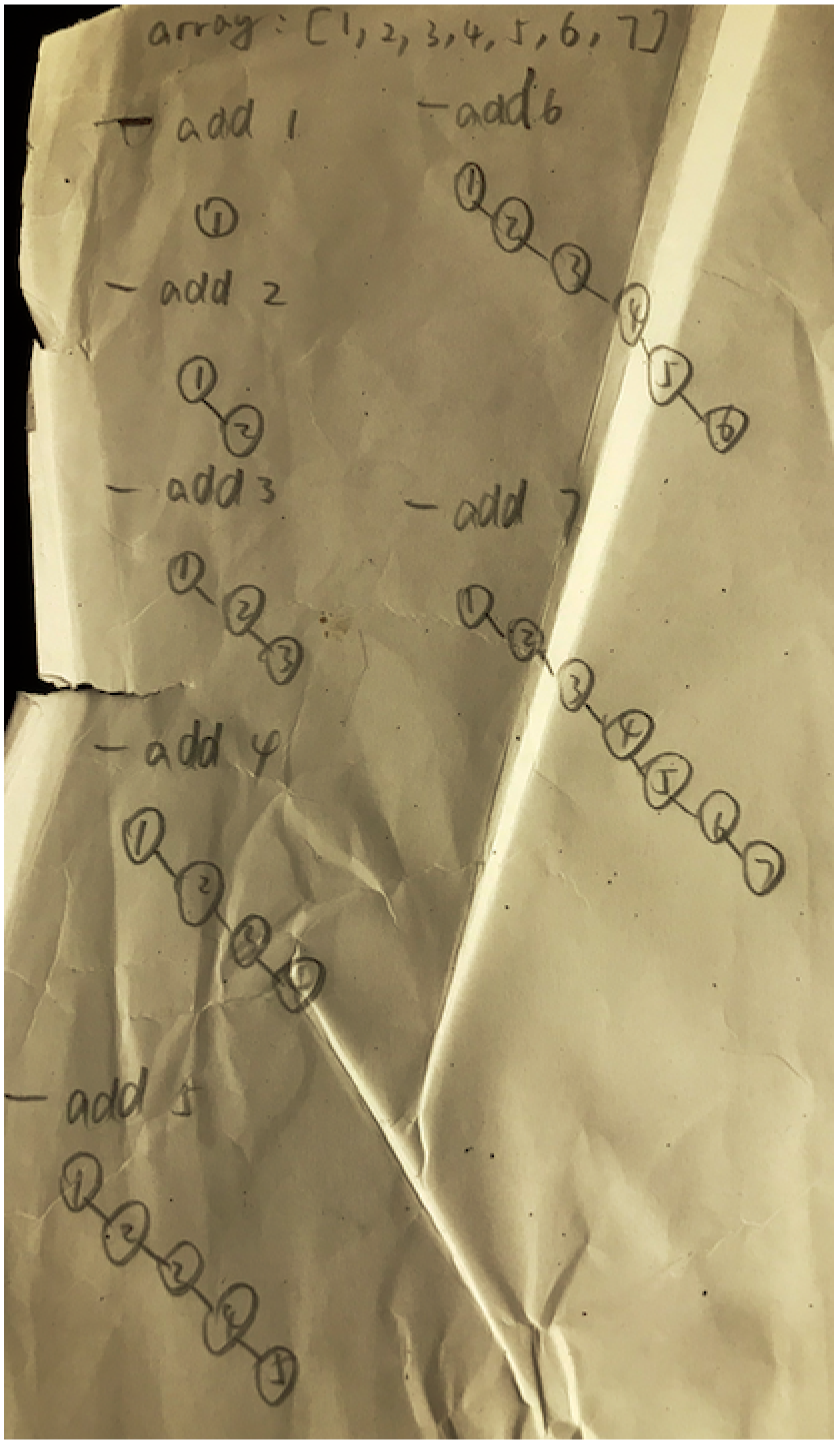
1. **What is the big-O search time for a balanced binary search tree? Give a logical argument for your response for all the questions below.**
    1. **Part 1: Assume that the binary tree is perfectly balanced and full.**

In a balanced binary search tree, search time is o(n), because binary search tree hold the property that left node < root node < right node, each time we compare the node value and target value, if target value < node value, we search left tree, else we search right tree, we eliminate half of the nodes in each operation, and height of binary search tree is log(n), so it takes at most O(log_n) time.

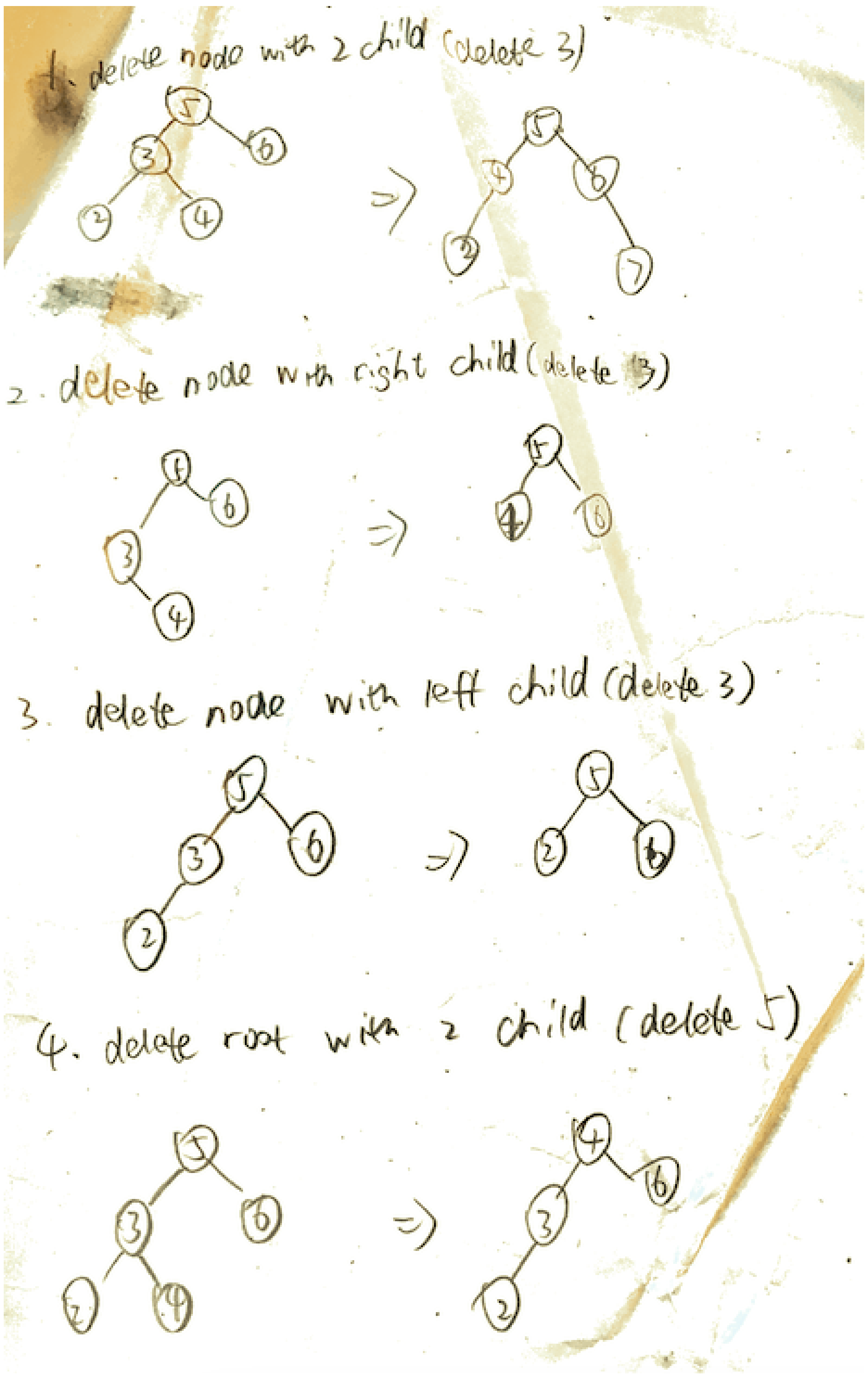
2. **Part 2: Assume that it is not.**

If the tree is not perfectly balanced and full, the search time is still O(log_n), because each search operation, we can still eliminate nearly half of the nodes using the property of BST.

2. **What data set would create an unbalanced binary search tree? Draw the steps of the creation of this binary search tree for an array of 7 numbers. (You create the array of numbers; show the creation of the binary tree step-by-step as these numbers are added to the tree. You will need to draw the binary tree as each number is added).**



3. **How do you delete an item from a binary search tree, while preserving the binary search tree structure? Include a picture that explains how this deletion looks before and after, and explain how you can be -sure- that what remains still conforms to the required binary search tree structure. Do not Google for this answer: please work through the steps on your own. Start with a picture of a binary search tree and go from there. Explain at least 4 different cases:**


1. delete node with 2 child (delete 3)
2. delete node with right child (delete 3)
3. delete node with left child (delete 3)
4. delete root with 2 child (delete 5)

1. **item you're deleting has 2 children,**

In case 1: Follow the example picture above, we want to delete 3 from the tree, we get the min value node in the right subtree of the node to be deleted, which is 4. Set the value of 3 to be 4, and remove 4 from the tree. If 4 has the one right child, we attach 4's right child to 3's right child. Since 4 is selected min value node of the right subtree, we only need to consider that if it has a right child, because if it has a left child we won't select 4 as min value node in the right subtree of the node to be deleted.

What remains still conforms to the required binary search tree structure because min value of node to be deleted right subtree is greater than all nodes in left subtree and its less than all other nodes in right subtree, so we can replace the node to be deleted as the node’s min value of right subtree and the BST property still holds.

2. **item you're deleting has a right child,**

In case 2: Follow the example picture above, if we want to delete 3 from the tree, it only has one right child. We replace the value of 3 as its right child 4, and delete right node 4 from the tree by traversing the tree. If 4 has a left or right child, we can attach its children to 3’s right child.

What remains still conforms to the required binary search tree structure because the node to be deleted’s right child is greater than the node to be deleted, and its less than the node to be deleted’s parent. So we can simple replace the value node to be deleted by its right child and preserve the tree data structure.

3. **item you're deleting has a left child,**

In case 3: Follow the example picture above, if we want to delete 3 from the tree, it only has one left child. We replace the value of 3 as its left child 2, and delete left node 2 from the tree by traversing the tree. If 4 has a left or right child, we can attach its children to 3’s left child.

What remains still conforms to the required binary search tree structure because the node to be deleted’s left child is less than the node to be deleted. So we can simple replace the value node to be deleted by its left child and preserve the tree data structure.

4. **item you're deleting is the root with 2 children.**

In case 4: Following the example picture above, we want to delete root, which is 5 from the tree, we get the min value node in the right subtree of the node to be deleted, which is 4. Set the value root to be 4, and remove 4 from the tree.

What remains still conforms to the required binary search tree structure because min value of node to be deleted’s right subtree is greater than all nodes in left subtree and its less than all other nodes in right subtree, so we can replace the node to be deleted as the node’s min value of right subtree and the BST property still holds.

4. **What is the recurrence relation for search in a binary search? Your answer should be in the form of T(n) = aT(n/b) + f(n). Clearly state the values for *a*, *b* and *f(n)*.**

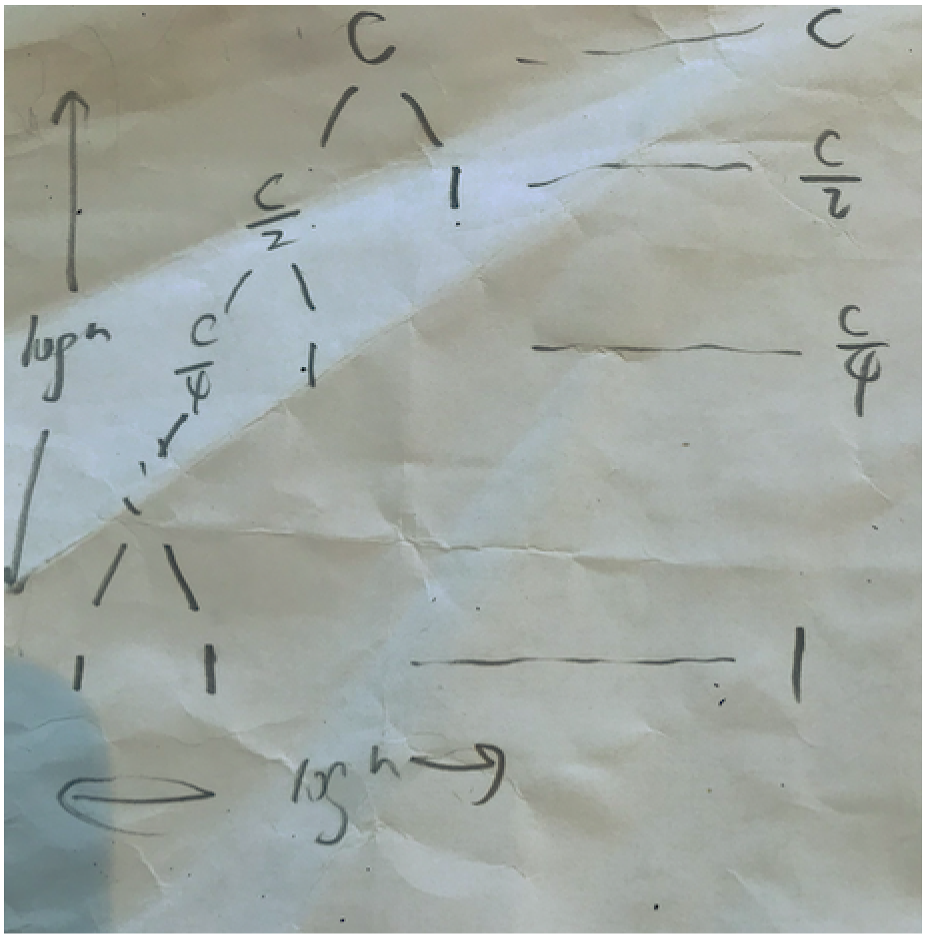
T(n) = T(n/2) + O(1), where a = 1 , b = 2, f(n) is O(1).

a = 1 because in binary search operation every time we eliminate half of the nodes, either smaller part or larger part, so we only have one subproblem to deal with.

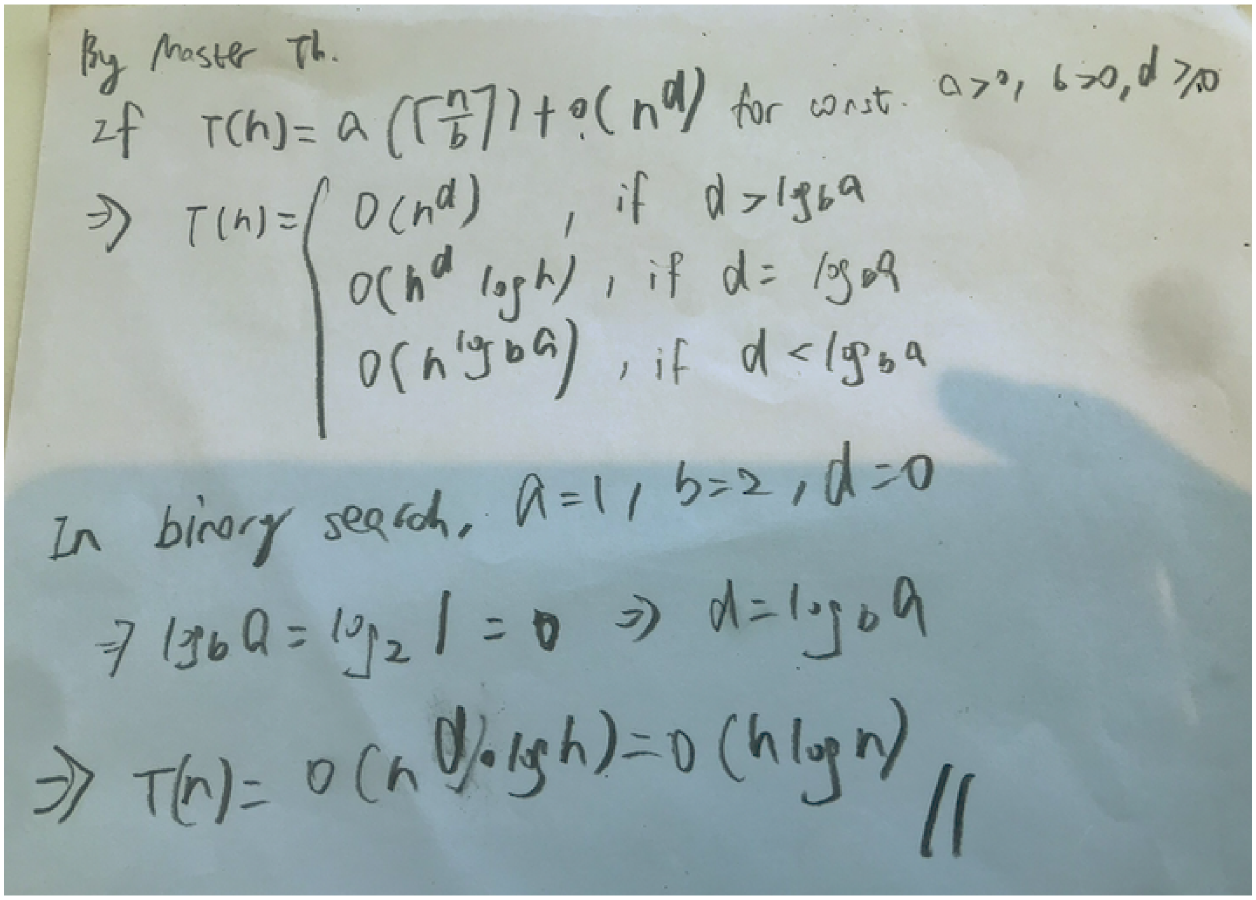
b = 2 is because in search operation each time we divide the problem into two parts.

f(n) is O(1) because what we do in binary search is to compare the target with middle point, and each comparsion' s cost is O(1)

5. **Draw the recurrence tree for question 4. Label the height and width of the tree, as well as the amount of work done at every level.**



6. **Confirm that your solution to #5 is correct by solving the recurrence for binary search using the *master theorem*. For full credit, clearly define the values of *a*, *b*, and *d* (or epsilon if you are using CLRS notation) and show all the steps of this method.**



By Master Th.

If $T(n) = a(\lceil \frac{n}{b} \rceil)) + O(n^d)$ for const. $a > 0, b > 0, d \geq 0$

$$\Rightarrow T(n) = \begin{cases} O(n^d), & \text{if } d > \log_b a \\ O(n^d \log n), & \text{if } d = \log_b a \\ O(n^{\log_b a}), & \text{if } d < \log_b a \end{cases}$$

In binary search, $a = 1, b = 2, d = 0$

$\Rightarrow \log_b a = \log_2 1 = 0 \Rightarrow d = \log_b a$

$\Rightarrow T(n) = O(n^0 \cdot \log n) = O(n \log n)$ //