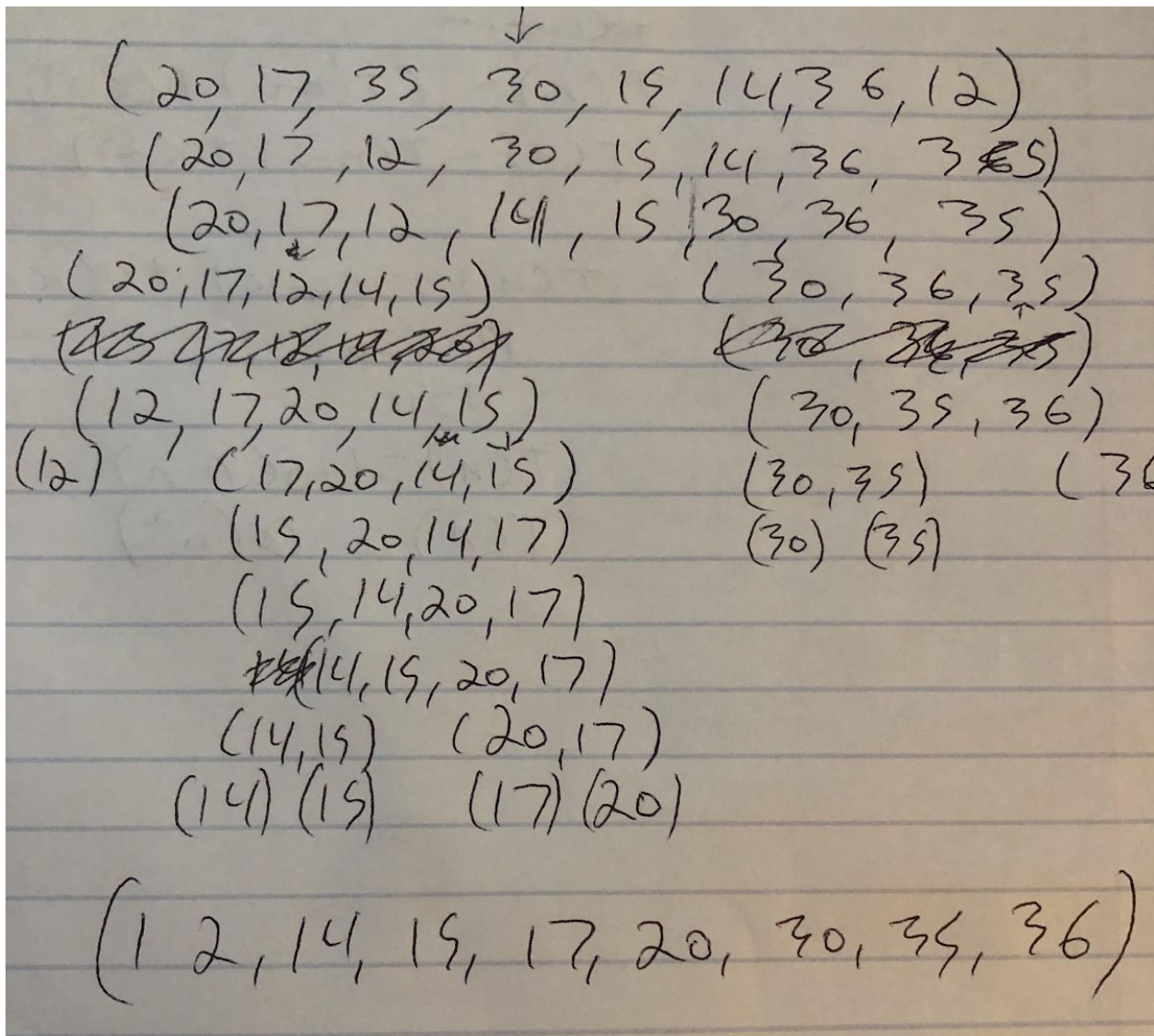


1.



2.

```
def recursive_insertion_sort(array, length):
    if length <= 1:
        return
    recursive_insertion_sort(array, length-1)
    last_element = array[length-1]
    x = length-2
    while (x >= 0 and array[x] > last_element):
        array[x+1] = array[x]
        x = x-1
    array[x+1] = last_element
```

$$T(n) = T(n-1) + O(n)$$

$$T(0)=1$$

$$T(n) = T(n-1) + O(n) \dots (1)$$

Handwritten derivation of the recurrence relation  $T(n) = T(n-1) + O(n)$ :

$$\begin{aligned}
 T(n) &= T(n-1) + O(n) \\
 T(0) &= 1 \\
 T(n) &= T(n-1) + O(n) \dots (1) \\
 T(n-1) &= T(n-2) + O(n) \\
 \cancel{T(n-1)} &= \\
 T(n) &= (T(n-2) + O(n)) + O(n) \\
 T(n) &= T(n-2) + O(n) + O(n) \\
 &\vdots \\
 T(n) &= T(n-k) + O(kn) \dots (2) \\
 n-k &= 0 \\
 n &= k \\
 T(n) &= 1 + O(n \cdot n) \\
 T(n) &= O(n^2)
 \end{aligned}$$

3.

```

modified_quick_sort(array, low, high) {
    if (low < high) {
        pivot = Algo(array)
        Swap(array, pivot, last) // Swap function from textbook
        parted = partition(array, low, high, pivot) // partition function from textbook
        modified_quick_sort(array, low, parted-1)
        modified_quick_sort(array, parted + 1, high)
    }
}

```

Using the Partition() Program from the textbook, we know that it's already  $O(n)$  time. We are also using the  $AlgO()$  function, which also takes  $O(n)$  time. Both these combine take  $O(n)$  time.

$$\begin{aligned} \text{Recurrence Relation } 2T(n) &= 2T\left(\frac{n}{2}\right) + O(n) \\ T(n) &= 2T\left(\frac{n}{2}\right) + O(n) \text{ when } n > 1 \\ &= 1C \text{ when } n = 1 \end{aligned}$$

$$\text{Assume } n = 2^k$$

$$\Rightarrow k_0 = \log_2 n$$

$$\begin{aligned} T(n) &= 2T\left(\frac{n}{2}\right) + Cn \\ &= 2\left[2T\left(\frac{n}{2^2}\right) + 2Cn\right] + 2Cn \\ &= 2^2\left[2T\left(\frac{n}{2^3}\right) + 3Cn\right] + 4Cn \\ &= 2^3\left[2T\left(\frac{n}{2^4}\right) + 4Cn\right] + 8Cn \\ &\vdots \end{aligned}$$

$$\begin{aligned} &= 2^{k_0}T\left(\frac{n}{2^{k_0}}\right) + k_0Cn \\ \Rightarrow T(n) &= n + C + k_0Cn \\ &= n.C + C.n.\log_2 n \\ &= O(n \log_2 n) \end{aligned}$$

4.

1. Find middle element of array
2. If this middle element is equal to  $i$ , then done
3. If not,
  - a. then do the same for the subarray on the left
  - b. Do the same for the sub array on the right
5. If after going through all of them, none are found, then it does not exist.