Algorithm Documentation for Dominator

1. Recursive Approach

Pseudo Code

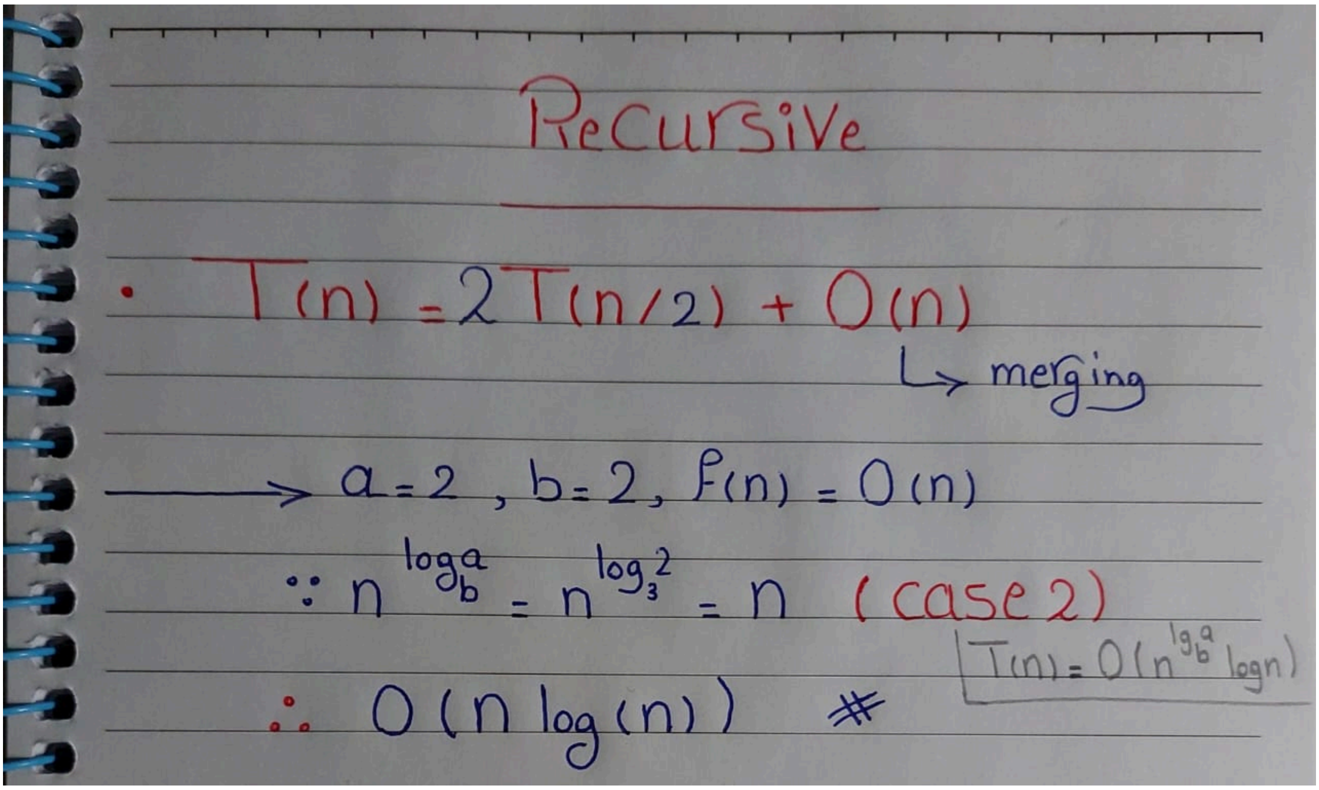
```
FUNCTION findDominatorRecursive(A, left, right):
    IF left == right:
        RETURN left

    mid ← (left + right) / 2
    leftIndex ← findDominatorRecursive(A, left, mid)
    rightIndex ← findDominatorRecursive(A, mid+1, right)

    leftValue ← A[leftIndex] if valid ELSE -1
    rightValue ← A[rightIndex] if valid ELSE -1

    leftCount ← count of leftValue in A[left to right]
    rightCount ← count of rightValue in A[left to right]
    threshold ← (right - left + 1) / 2

    IF leftCount > threshold:
        RETURN leftIndex
    IF rightCount > threshold:
        RETURN rightIndex

    RETURN -1
```

Implementation in Java

```java
public class DominatorRecursive {
    public static int findDominator(int[] A) {
        if (A.length == 0)
            return -1;
        return findDominatorRecursive(A, 0, A.length - 1);
    }

    private static int findDominatorRecursive(int[] A, int left, int right) {
        if (left == right)
            return left;

        int mid = (left - right) / 2;
        int leftIndex = findDominatorRecursive(A, left, mid);
        int rightIndex = findDominatorRecursive(A, mid + 1, right);

        int leftValue = leftIndex != -1 ? A[leftIndex] : -1;
        int rightValue = rightIndex != -1 ? A[rightIndex] : -1;

        int leftCount = 0, rightCount = 0;
        for (int i = left; i <= right; i++) {
            if (A[i] == leftValue)
                leftCount++;
```

```
                if (A[i] == rightValue)
                    rightCount++;
            }

            int threshold = (right - left + 1) / 2;
            if (leftCount > threshold)
                return leftIndex;
            if (rightCount > threshold)
                return rightIndex;
            return -1;
        }
    }
```

Analysis & complexity (steps)



---

2. Non-Recursive Approach

Pseudo Code

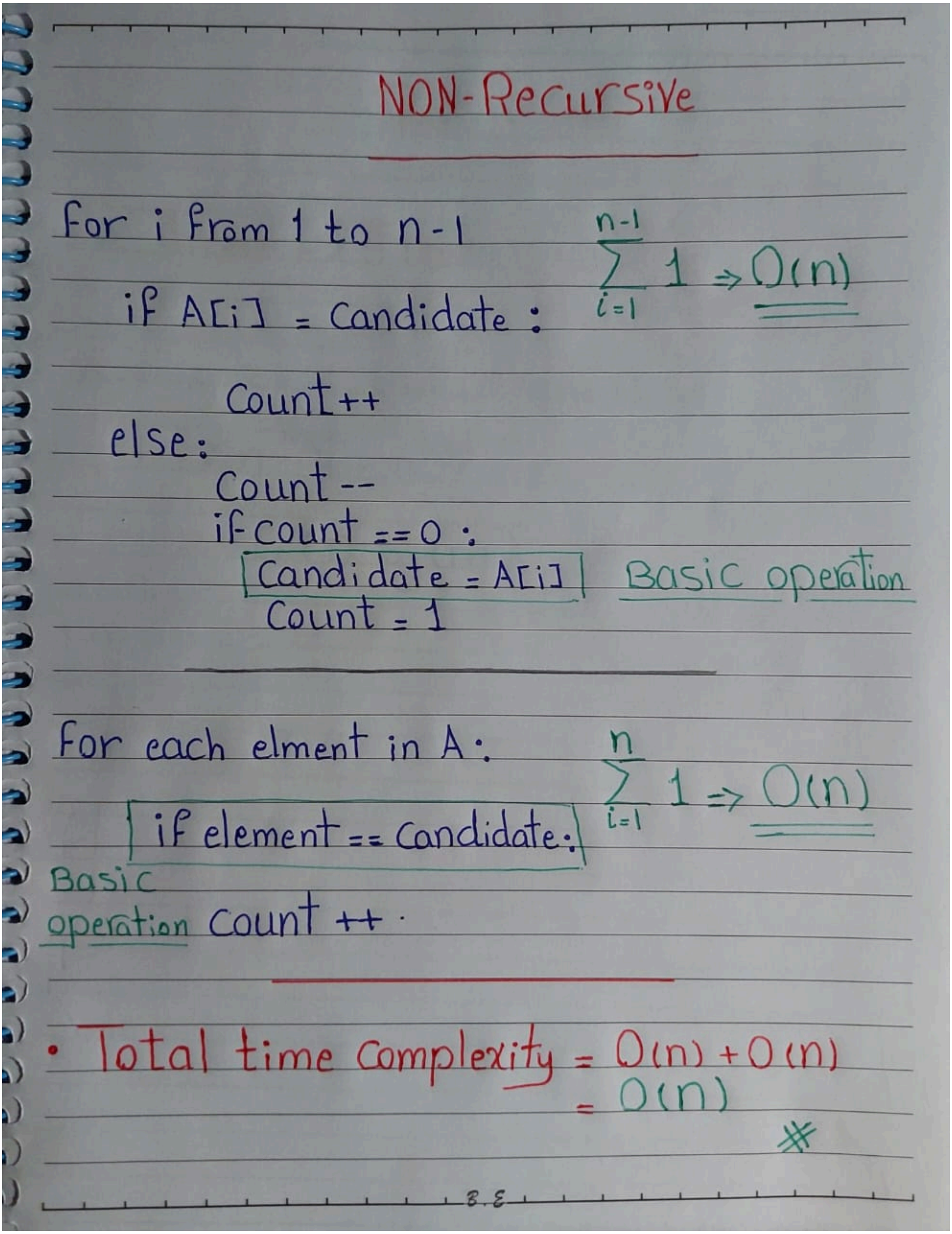
```
FUNCTION findDominator(A):
    IF A is empty:
        RETURN -1

    candidate ← A[0], count ← 1, candidateIndex ← 0

    FOR i from 1 to length(A) - 1:
        IF A[i] == candidate:
            count += 1
        ELSE:
            count -= 1
            IF count == 0:
```

```
                candidate ← A[i]
                candidateIndex ← i
                count ← 1

    count ← 0
    FOR each element in A:
        IF element == candidate:
            count += 1

    IF count > length(A) / 2:
        RETURN candidateIndex
    RETURN -1
```

Implementation in Java

```java
public class DominatorNonRecursive {
    public static int findDominator(int[] A) {

        if (A.length == 0)
            return -1;

        int candidate = A[0], count = 1, candidateIndex = 0;

        for (int i = 1; i < A.length; i++) {
            if (A[i] == candidate) {
                count++;
            } else {
                count--;
                if (count == 0) {
                    candidate = A[i];
                    candidateIndex = i;
                    count = 1;
                }
            }
        }

        count = 0;
        for (int value : A) {
            if (value == candidate) {
                count++;
            }
        }
        return count > A.length / 2 ? candidateIndex : -1;
    }
}
```

Analysis & complexity (steps)



NON-Recursive

```
For i from 1 to n-1
    if A[i] = Candidate :
        Count++
    else:
        Count --
        if count == 0 :
            Candidate = A[i]
            Count = 1
```

Basic operation: Candidate = A[i]

$$\sum_{i=1}^{n-1} 1 \Rightarrow O(n)$$

```
For each elment in A:
    if element == Candidate:
        Count ++.
```

Basic operation: if element == Candidate

$$\sum_{i=1}^{n} 1 \Rightarrow O(n)$$

- Total time complexity $= O(n) + O(n)$
  $= O(n)$ #

---

3. Comparison Table

Feature	Non-Recursive	Recursive
Time Complexity	$O(n)$	$O(nlog(n))$

Feature	Non-Recursive	Recursive
Efficient For	Large arrays	small arrays
Logic	Simple logic	Divide-and-conquer