4. Consider the following algorithm, which is a divide-and-conquer sorting algorithm. First, derive a recurrence relation for the algorithm. Then solve the recurrence using the Master Theorem.

```
// Sort the array A, but only the range starting at index i and ending
// at index j.
void LLDSort(Array A, int i, int j)
    if A[i] > A[j]
        swap(A[i], A[j])
    if i < j − 1
        // t will be 1/3 the size of the array segment
        // (rounded down)
        t ← floor( (j − i+1)/3 )
        LDDSort(A, i, j − t)
        LDDSort(A, i + t, j)
        LDDSort(A, i, j − t)
```

We assume that to make one comparison, the time taken is c, and that to make one swap between two elements, the time taken is c. Since i is the beginning of the array segment and j is the end of the array segment, the second condition $i < j - 1$ is true whenever, by algebra

$$i < j - 1$$
$$1 < j - i.$$

Hence, if the array segment's input size is $n = j - i + 1$, if we add 1 to both sides,

$$2 < j - i + 1$$
$$2 < n.$$

Then in order for the second condition to be true, n must not be 2. That is, recursion ends when $n \leq 2$. Hence, $T(n) = 3c = \Theta(1)$ if $n \leq 2$.

When the algorithm recurses, we let $t = n/3 = (j - i + 1)/3$. Then, the recurrence LLDSort$(A, i, j - t)$ has input size

$$
\begin{aligned}
(j-t)-i+1 &= (j-n/3)-i+1 \\
&= (j-(j-i+1)/3)-i+1 \\
&= j-j/3-i/3-1/3-i+1 \\
&= 2j/3-2i/3+2/3 \\
&= (2/3)(j-i+1) \\
&= (2/3)n.
\end{aligned}
$$

Likewise, the recurrence LLDSort$(A, i+t, j)$ has input size

$$
\begin{aligned}
j-(i+t)+1 &= j-(i+n/3)+1 \\
&= j-(i+(j-i+1)/3)+1 \\
&= j-i-j/3+i/3-1/3+1 \\
&= 2j/3-2i/3+2/3 \\
&= 2/3(j-i+1) \\
&= (2/3)n.
\end{aligned}
$$

Therefore, whenever $n > 2$, the recurrence always has input size $(2/3)n$. Since the algorithm sorts without splitting or merging, we do not need to add a constant to the end. Then, for $n > 2$, $T(n) = T(2n/3) + T(2n/3) + T(2n/3)$.

In general, we get the recurrence

$$
T(n) = \begin{cases} \Theta(1) & \text{if } n \le 2, \\ 3T(2n/3) & \text{otherwise.} \end{cases} \tag{1}
$$

We will now solve recurrence (1) using the Master Theorem.

Proof. We assume that $a = 3, b = 3/2, f(n) = 0$, and $\log_b a = \log_{3/2} 3 \approx 2.71$. Note that $f(n) = O(1)$ and that if we let $\epsilon = 2$, then $O(n^{\log_b a-\epsilon}) = O(n^{\log_{3/2} 3-2}) = O(n^{\log_{3/2} 1}) = O(n^0) = O(1)$. Since $f(n) = O(n^{\log_b a-\epsilon})$, then $T(n) = \Theta(n^{\log_b a}) = \Theta(n^{\log_{3/2} 3})$. Hence, we have proven that $T(n) = \Theta(n^{\log_{3/2} 3})$ by the Master Theorem. ■