There are 2 questions for a total of 10 points.

---

1. (3 points) Apply the Master theorem and give the solution for the following recurrence relations in big-O notation. Explanation is not required.
   (a) $T(n) = 7 \cdot T(n/2) + O(n^2)$, $T(1) = O(1)$

   **Answer:** $T(n) \in O(n^{\log_2 7})$

   (b) $T(n) = 7 \cdot T(n/2) + O(n^3)$, $T(1) = O(1)$

   **Answer:** $T(n) \in O(n^3)$

   (c) $T(n) = 8 \cdot T(n/2) + O(n^3); T(1) = O(1)$

   **Answer:** $T(n) \in O(n^3 \log n)$

2. (7 points) You are given an array of bits, $b[1..n]$ (i.e., $b[i] \in \{0, 1\}$) where $n \geq 2$ is a power of 2 (i.e., $n = 2^k$ for some integer $k$), and $b[1] = b[n] = 1$. Give an algorithm that finds two consecutive bits that are the same. That is, it finds an index $1 \leq i \leq n - 1$ such that $b[i] = b[i + 1]$. Give a time analysis for your algorithm, and brief explanation for correctness.

   [Please note that a significant number of points will be for efficiency of your algorithm for this question.]

**Solution: Algorithm:**

```
function SOLVE(b[1..n], n = 2^k)
    if n = 2 then
        return 1
    end if
    if b[n/2] = b[n/2 + 1] then   ▷ now at least (in fact exactly) one of b[n/2] and b[n/2 + 1] is 1
        return n/2
    end if
    if b[n/2] = 1 then
        return SOLVE(b[1..n/2], n/2)
    else
        return n/2+ SOLVE(b[(n/2 + 1)..n], n/2)
    end if
end function
```

**Implementation note**

We implement this algorithm using a pointer to the first element and the size of the array, and hence we do not need to worry about making copies to the array (note that we always use contiguous subarrays, hence it is sufficient to use this representation).

**Runtime analysis**

$T(2) = O(1)$ since it is a constant time check in the beginning of the function.

$T(n)$ can be at most $O(1) + T(n/2)$, since we recurse only on the case where the two middle elements are not equal, and in both the cases, the array sizes are exactly half of $n$ (since $n$ is even and $n/2 - 1 + 1 = n/2$ and $n - (n/2 + 1) + 1 = n/2$). Arithmetic operations and pointer arithmetic is assumed to take $O(1)$ time.

Hence we have $T(n) \leq T(n/2) + O(1)$, and applying master theorem, we have $T(n) \in O(\log_2 n) = O(k)$

**Proof of correctness**

We induct on $k$ as defined in the problem, with the predicate

$P(k)$ = There is an index $i$ such that $a[i] = a[i+1]$ in any array of size $2^k$ which has $a[1] = a[2^k] = 1$ and one such occurrence is returned by the algorithm *solve* when applied on the array $a$

**Base case:** $k = 1$ Here note that since the first and the last elements of the array are consecutive, and both are 1, the answer is simply the index of the first element of the bit array, and it is returned by the algorithm, completing the base case.

**Inductive hypothesis:** $P(k-1)$ is true for some $k > 1, k \in \mathbb{N}$.

**Inductive step:**

If the middle two elements are the same, then we are trivially done. Let $n = 2^k$

If the left middle element $a[n/2]$ is 1, then the sub-array $a[1..n/2]$ satisfies the conditions in the inductive hypothesis, and hence there exists such an index in $a[1..n/2]$ that is returned by the call to *solve* by the inductive hypothesis. Hence such an index exists in $a[1..n]$ as well, and both indices are the same since $a[1..n/2]$ starts at the beginning of $a[1..n]$.

Now suppose it is not 1. Since both the middle elements are distinct, the right middle element $a[n/2+1]$ must be 1. Note that now the subarray $a[(n/2+1)..n]$ satisfies the conditions in the inductive hypothesis, and hence there must exist such an index in $a[(n/2+1)..n]$ that is returned by the call to *solve* done in the algorithm, by the inductive hypothesis. Note that since indices in $a[(n/2+1)..n]$ are offset by $n/2$ as compared to $a[1..n]$, the index in $a[1..n]$ corresponding to the answer in $a[(n/2+1)..n]$ would be $n/2$ plus the index in $a[(n/2+1)..n]$, which is precisely what the return after the call does.

Hence we have shown that in any case, there exists an index $i$ in an array $a$ satisfying the conditions of the inductive hypothesis, that satisfies $a[i] = a[i+1]$ and that $solve(a[1..n], n)$ returns that index. This shows that $P(k)$ is true, so it completes the inductive step, and we are done by the principle of mathematical induction.

Applying this result to the array $b[1..n]$ given to us, we can see that the function *solve* indeed returns an index with the properties as requested in the problem.