JUMP TO SOLUTION

```
   Input: str: a string of length n
   Input: n: the length of str
   Output: the first index i such that str[i] = a and str[i + 1] = b, or −1
           if str doesn't contain the substring "ab"
1  Algorithm: abSearch
2  if n < 2 then
3  |   return −1
4  else if n = 2 then
5  |   if str = "ab" then
6  |   |   return 1
7  |   else
8  |   |   return −1
9  |   end
10 else
11 |   midpt = ⌊n/2⌋
12 |   left = abSearch(str[1..midpt])
13 |   center = abSearch(str[midpt..midpt + 1])
14 |   right = abSearch(str[midpt + 1..n])
15 |   if left ≠ −1 then
16 |   |   return left
17 |   else if center ≠ −1 then
18 |   |   return midpt
19 |   else if right ≠ −1 then
20 |   |   return right + midpt
21 |   else
22 |   |   return −1
23 |   end
24 end
```

1. What would be the base case(s) when proving that abSearch is correct? Prove that abSearch is correct for the base case(s).

2. Prove that abSearch returns -1 if str does not contain "ab" as a substring. You may omit the base case for this proof (done in problem 1).

   Hint: you will need to think carefully about the different cases of the if statement in lines 15–23 to show that abSearch always returns -1 when str doesn't contain "ab."

SOLUTION

1. FIRST BASE CASE (n<2): If n=1 or n=0 it is a string with fewer than 2 elements and as such the if statement in line 2 returns true and abSearch returns -1 trivially since a string smaller than size 2 can't hold \`ab\`.

   SECOND BASE CASE (n=2): If n=2 abSearch returns either 1 or -1. It will return 1 if input ==\`ab\`, otherwise it will return -1 per line 8.

2. Suppose that for all inputs of length n from 3 up to k where k > 2 abSearch correctly returns -1 if that input does not contain substring \`ab\`.

   Consider n = k+1.

   midpt = [(k+1) / 2]
   left = abSearch (str[1...midpt])
   center = abSearch(str[midpt...midpt+1])
   right = abSearch(str[midpt+1...n])

   For each recursive call from left, center, and right there are three possible cases:

Case 1	Recursive call input length is less than 2, returning -1 per first base case
Case 2	Recursive call input length is exactly 2, and will return -1 if input != \`ab\` per the second base case
Case 3	Recursive call input length is greater than 2. In this case a new midpt will be chosen and recursive calls will be made again. We know that the length of any recursive input will be at most [(k+1) / 2] and we also know that this length will be less than or equal to k due to k being greater than 2 and the floor operator rounding k+1's midpt down to the nearest integer. Since we know the string will be less than or equal to k we can use our supposition for n = k and we know that this will correctly return -1 if substring \`ab\` is not present per the Inductive Hypothesis

   If substring \`ab\` isn't present then each recursive call returns -1 and the if statements on lines 15, 17, and 19 aren't met, so abSearch returns -1 per the else condition on line 21.

   Therefore, by induction, abSearch correctly returns -1 if str does not contain \`ab\` as a substring.