634250
Read the following method skeleton and choose the best expression to fill in the blank on **line 2** so that the method will behave correctly:

```
/**
* Takes a string reference and counts the number of times
* the character 'A' or 'a' appears in the string object.
* @param aString   String reference to object containing chars.
* @precondition    aString is not null (you may assume this is true).
* @return          The number of times 'A' or 'a' appears in the string.
*/
public static int countAs(String aString)  // line 1
{
    int counter = __________;              // line 2
    int totalA = 0;                        // line 3
    while (counter < __________)           // line 4
    {
        if ( __________.equals("A") )      // line 5
        {
            totalA = totalA + __________;  // line 6
        }
        counter++;                         // line 7
    }
    return __________;                     // line 8
}
```

*a. 0

b. 1

c. aString.size()

d. aString.length

e. aString.length() - 1

f. "
g. "
h. "
i. "
j. "

General Feedback:

This method must examine each character in the given string to see if it is 'A' or 'a', and uses a loop to do so. The variable `counter` is being used as a loop index, and by imagining what will fill in other blanks, is also being used to keep track of the position of the current character in the string that is being examined. Since line 7 methodically increases `counter` on each loop iteration, it must be traversing left-to-right through the string, so zero is the best choice for its initial value.

633561
You are writing a depth-first search on a platform that doesn't support recursion. What data structure can help you complete your task?

*a. stack

b. queue

c. priority queue

d. hashtable

e. linked list

f. "
g. "
h. "
i. "
j. "

General Feedback:

As you visit nodes, if you push their neighbors on a stack, the neighbors will be the first popped off when you need to backtrack. This behavior is what makes a depth-first search depth-first.

632571
The worst-case time complexity of radix sort is:

a. O(1)

b. O(n)

*c. O(k n)

d. $O(n^2)$

e. none of the above

f. "
g. "
h. "
i. "
j. "

General Feedback:

n is the number of items to be sorted, as usual, and k is the number of digits in the largest number in the list. Radix sort processes the entire list (n numbers) once for each of the k digits in the largest number in the list, so the work done is proportional to n*k.