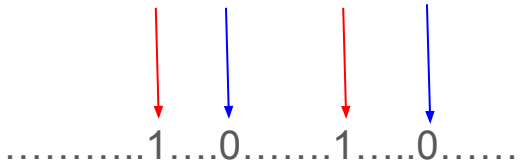
Motivation for the Algorithm

Consider the last swap that was made in making the string a palindrome

The red arrow point to the location pair that was swapped to make the string a
Palindrome. The other 0 and 1 are the corresponding location for this location.
If a bit is at location n then the corresponding location will be length-n.
So swapping the red arrows led to a palindrome. This one swap fixed two different location pairs

After swapping:



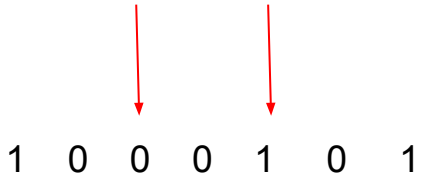
Algorithm steps

Bit differs, count =1

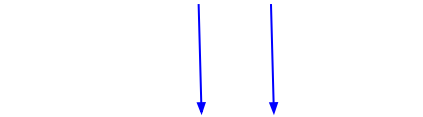
Bit differs, count = 2

Swap the blue positions

After swapping



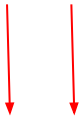
Bit differs, count =1



String length is odd and there is an extra 0
Put 1 in center, swap the blue location

1 0 0 1 0 0 1

Alternate case where 1's are more in the end



1 0 1 1 0 0 1

1 0 1 0 1 0 1

Final Algorithm

Some observations:

- If the number of location differences are even there is a solution

- If the string is of odd length there is always a solution
- If the string length is even length and the number of location differences Is odd there is no solution

Pseudocode:

```
Int countDiffLocations=0;

Int swap=0;
for(char c in binString){
    if(c!=bit at corresponding location)
       countDiffLocations++;
}
If counDiffLocations is odd and string length is even
     Return -1; // There is no solution
If countDiffLocations is odd and string length is odd
     swap=(countDiffLocations+1)/2  // There will be one extra swap to fix the middle string
If countDiffLocations is even
   Swap = countDiffLocations/2
```