Reversing a string:

Approach 1 (the idea is to break the string in single characters and then start concatenating from the last element):

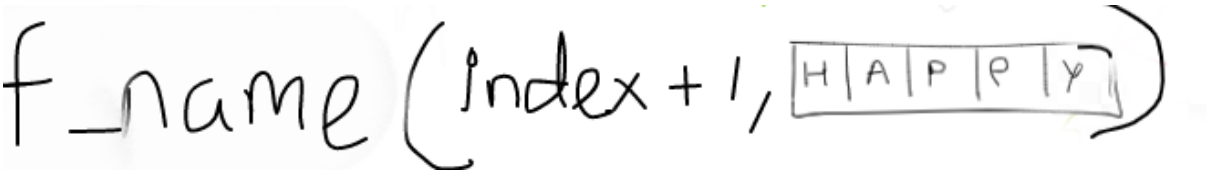
We need to convert the given string in character array, as I told you guys, such functions can be simply looked up and called, no need to redefine them.

so we do the same here using String.toCharArray().

Now we need to think how we will traverse the array recursively,

one way to that is we move from start to end, by incrementing the index, once we reach at end we can now start concatenation of the characters to string.

So our recursive call must look like



Now only thing is the base case, that simply should be the ending index +1,

here 4+1=5, then we return;

and before function ends we concatenate that character to a defined string.

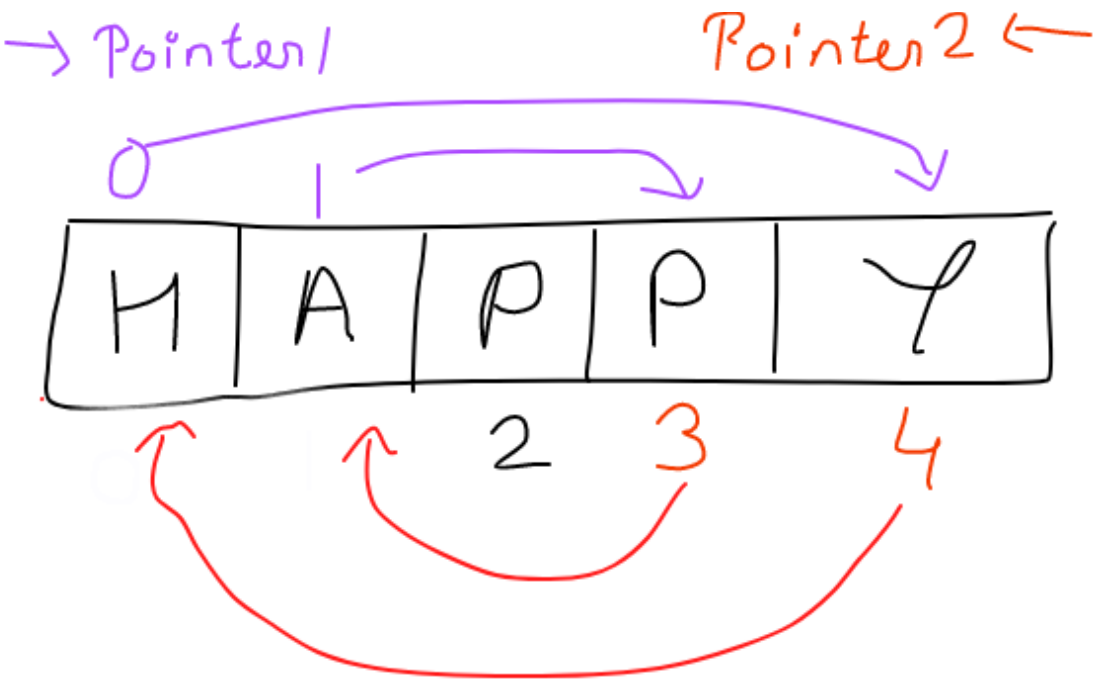
New_string = new_string + array[index];

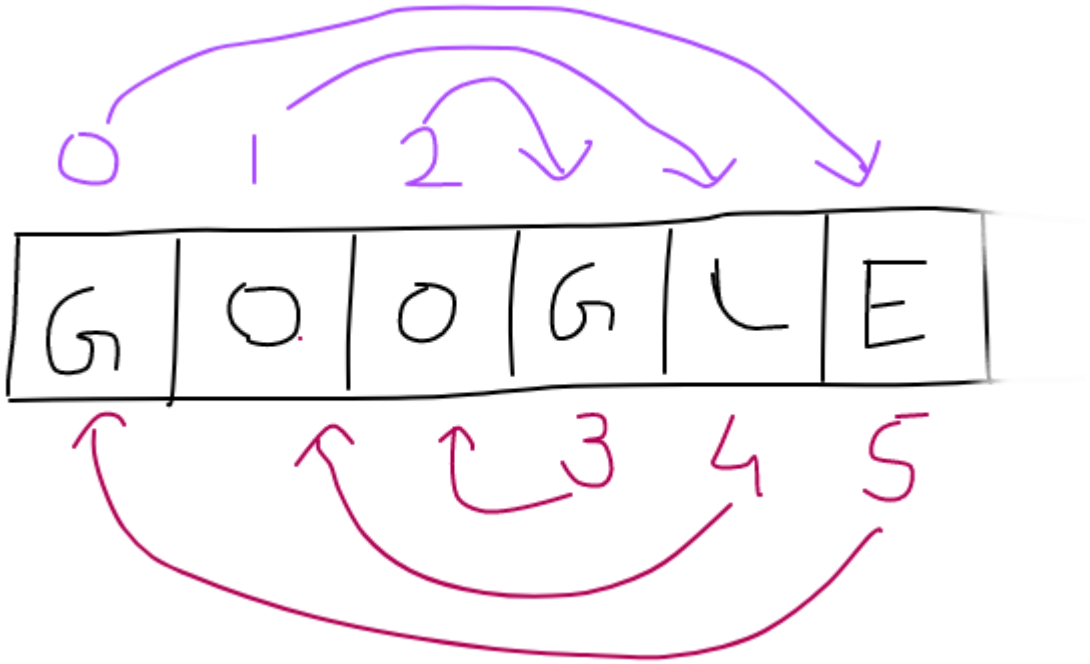
This should do the job, in O(n) time.

Approach 2 (how can we use the array to our benefit in solving this problem… we could traverse it from both ends, you could even think of having 2 movements through array indexes at any given point of time, this is called two pointer approach)

Conversion part is required here too,

Next we need to think of the recursive call,





It can be observed that pointer1 is incremented and pointer2 is decremented.

So this becomes the obvious part of our recursive call, and remaining is the array part.

f_name(pointer1++, pointer2--, array)

Now one thing to remember here is that, we will be swapping the values in pointer1 and pointer2, so we need to that before the recursive call, the usual way of swapping two elements by using third temporary one. Once done we move ahead (in program we done the same but incremented

and decremented pointer before the recursive call, either would work)

Now the stopping condition (base case), from the first example “HAPPY”, we see both pointer1 and pointer2 will arrived at same point (2 =>“P”), that should be our first hint to stop.

When pointer1 == pointer2 return;

Next example “GOOGLE”, here as it has even number of characters, pointers won’t be at same point.

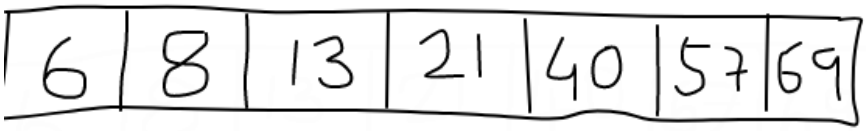
But surely, pointer1 and pointer2 will pass through each other, that is the time we should stop and return;

So when pointer2<pointer1, this ensures they have crossed, we will return;

Observe that we can have n number of base cases to stop the program.

Binary Search:

We will follow recursive approach to this,



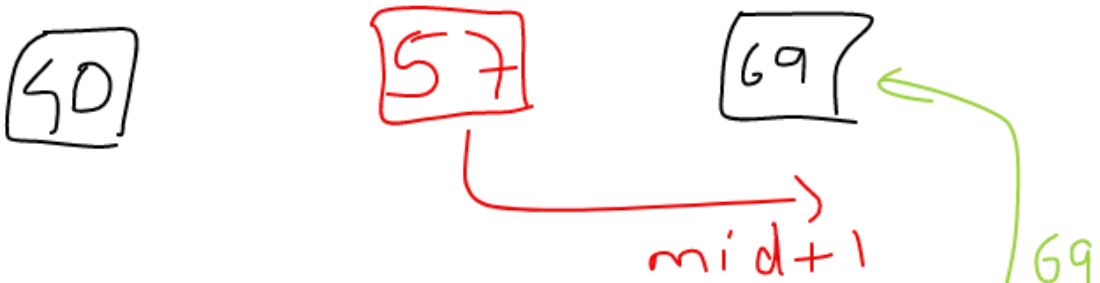
search = 69

start = 0

end = 6

$$mid = \frac{0+6}{2} = 3$$

$$mid = \frac{4+6}{2} = 5$$



$$mid = \frac{6+6}{2} = 6$$

Now when thinking about recursive function and how to move it,

Here we need select a part of array, on every iteration and the only way to do it, is knowing start and end of the array elements.

So this automatically becomes our recursive call, and also array needs to be passed.

Now we compare search element with midpoint of these elements, use formula :

mid = (start+end)/2;

For the base case, same as two pointer approach if end< start then return;

else we compare mid and search if matched return the index; if search element is greater we know the left half becomes useless, left half and current mid should be ignored, to that we simply adjust are start to new startpoint = mid+1; we go through similar process for right half, just adjusting end this time, endpoint = mid-1;

this way mid value helps to adjust start and end of the array elements we need to check.

As only (n/2) – 1, recursive calls are possible, so we check exactly half the elements or if odd number of elements than half+1;

For runtime use this fact, out of N array elements, the division occurs with factor of base 2,

$N/2^0 + 2^1 + 2^2 + 2^3 + 2^4……$

So, $N = 2 ^ k$

Taking log on both sides,

Log N = 1

Runtime becomes : O(log N).

Ice cream parlor problem:

Given a menu of ice cream prices, we need to spend our total money on two different flavors of them.

Eg: menu = [7,3,2,1,5,9], money = 10; return [0,1] as [7+3]=10, this is one of the possibilities, we need to return any of such cases indices.

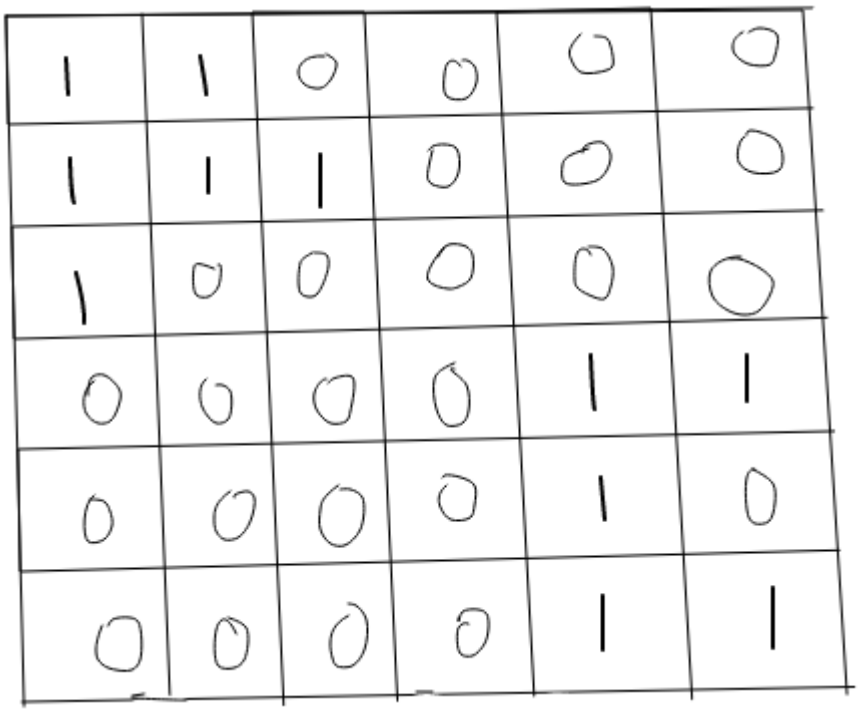
To solve this we first sort the array, then try to pick our two indices.

We use the idea of making it a “counting problem”, where if we subtract individual element in array from the total money we have, on each iteration we can make the rest of the array(right side) as searching problem, to find the complement i.e money – arrayElement,

If match is found we return the respective indices, else we move on.

If no match is found we return null.

Finding number of islands



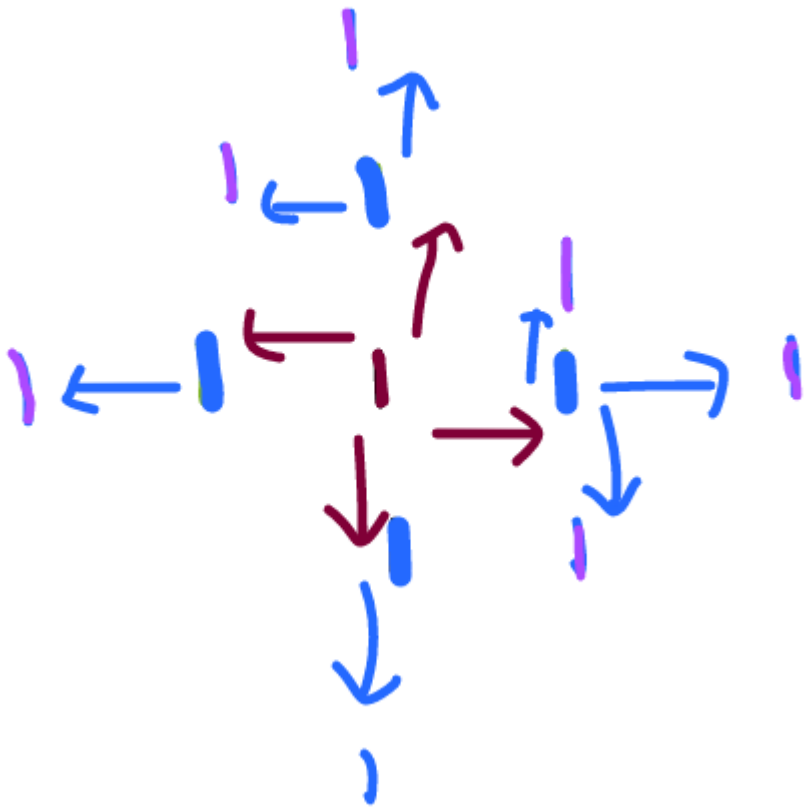
Given a following matrix we need to determine, the number of islands, where island is defined as single element 1 or maximum number of 1's connected vertically and horizontally.

So in our case clearly 2 islands can be observed.

Approach, we will be using tree traversal algorithm, called breadth first search (BFS).

Now in BFS we traverse all the children directly connected to parent, and this pattern continues until last child is reached.

Remember we can move up, down, left, right to find the connected 1’s. So now if we visualize it,



We will traverse similarly through entire matrix, but recursively, so last child would traverse through its parent , but since we can avoid here overlapping paths, while breaking through recursion we will make sure to set the parent already traverse to 0.

In this way will be avoiding same parent element problem, in short less recursive calls.

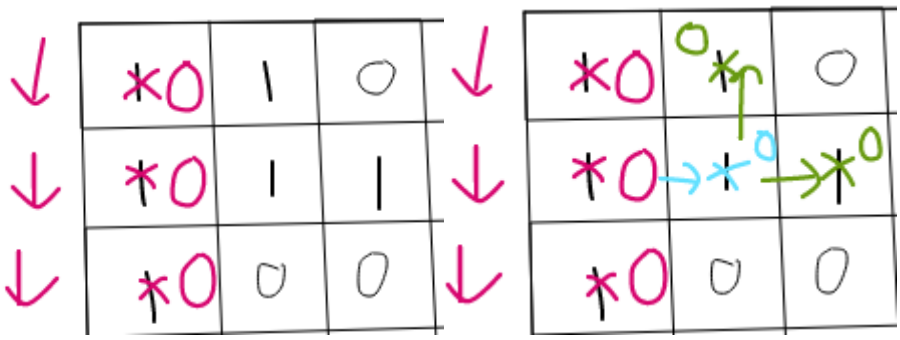
This approach has similarity to two pointer approach, where we move left, right, so 2 different recursive calls.

Here we move up and down too (as it's a matrix), so recursive calls become 4.

Therefore, the recursive function will consist of this four kinds of traversal.

Also for the base case we simply check all four boundaries of matrix, start, end of row and column. Then we need to check if element is 0, so that becomes part of base case too.

Finally before calling we need to set the matrix element where value is 1 to 0, to avoid repetition.



Here follow the color code to break the ordering of recursion (pink, blue, green).