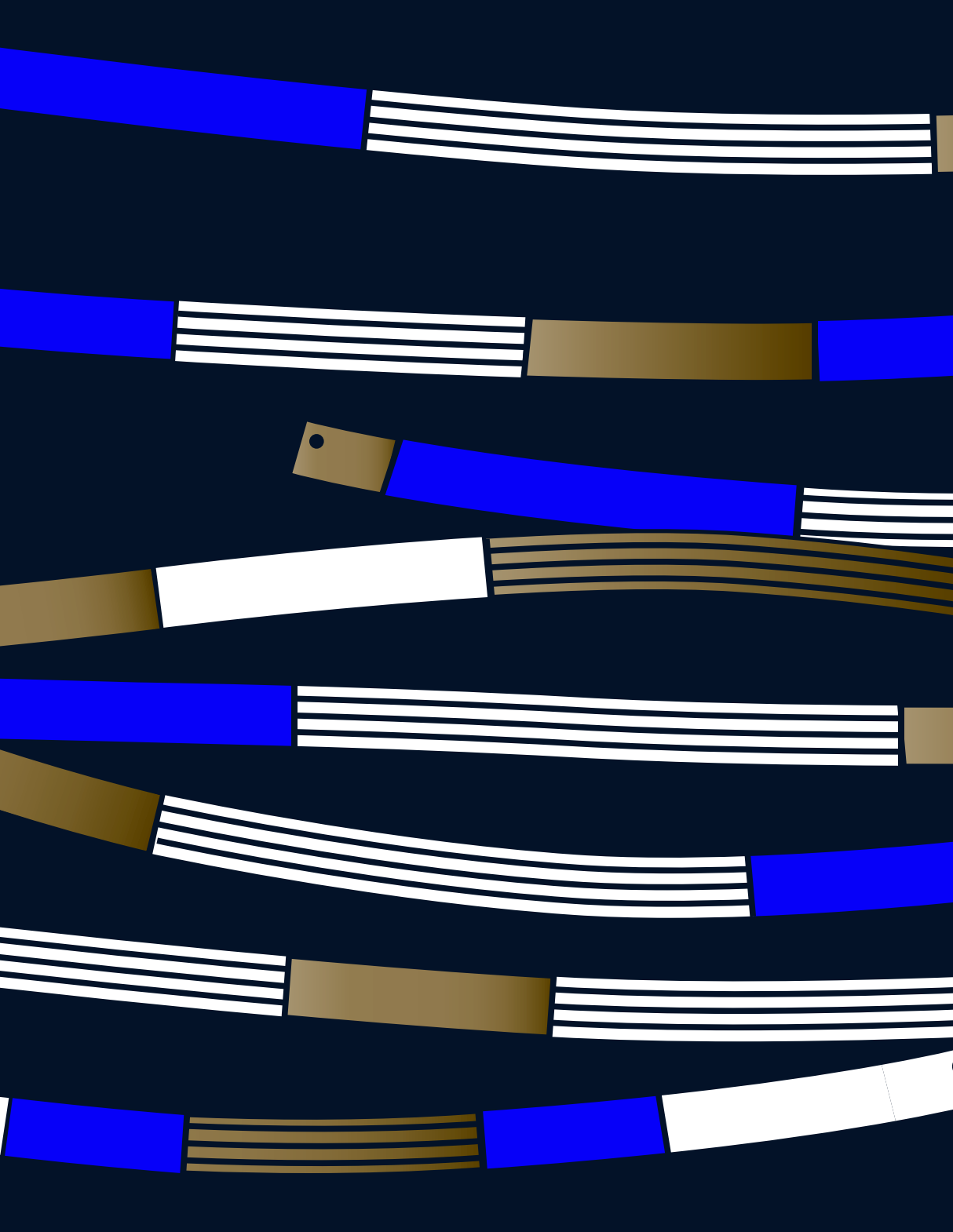


(1) Linear Search in Array

Let's say we have an Array

{1, 3, 5, 7, 8, 9, 0} (Not Sorted)

and we have target = 7

So we have to findout the number 7 in the Array if it exist then return the index of that target

if not exist then just return that Value not exist in the Array.

```
int index = -1
for (int i=0 ; i < nums.length ; i++) {
    if (nums[i] == target) {
        index = i
    }       return i;

    return index ;
```

(2) Largest in Array

To findout the largest Element in Array, we have to initialize a max named Variable with the value of Integer.MIN_VALUE;

So After every iforation, we have to Compare number with the max Variable if max is less than the Array value, then assign the value to it.

Code

```
int max = Integer.MIN_VALUE;
for (int i=0; i<nums.length; i++){
    if (max < nums[i]){
        max = nums[i];
    }
}
return max;
```

③ Binary Search (Devide and Conquer)

Here instead of Searching a target number from start to end, we have to devide the Array into 2 parts and Seach for the element, so that the time Complexity should be less as Compared to the linear Search.

→ For that we have to initialize a Variable called MID that always check the mid value of an Array.

So,

Code

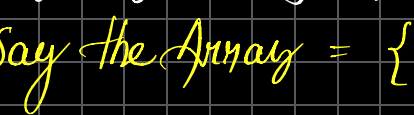
```
// function definition {
    int Beg = 0;
    int end = numbers.length - 1
    mid = (Beg + end) / 2;
    While (Beg <= end) {
        if (numbers[mid] == target){
            return mid;
        }
        mid = (Beg + end) / 2;
        if (numbers[mid] > target){
            end = mid - 1;
        }
        else if (numbers[mid] < target){
            Beg = mid + 1;
        }
    }
```

```
    return mid;
}
```

(9) reverse an Array

→ to reverse an Array, we have to perform the Kadane's Algorithm which is traverse the Array from begining and from end at a time.

let's say the Array = { 1, 7, 5, 3, 9 }



Code:

```
// function definition {
int start = 0, end = nums.length - 1;
while (start < end) {
    nums[start] = nums[start] ^ nums[end];
    nums[end] = nums[start] ^ nums[end];
    nums[start] = nums[start] ^ nums[end];
    start++;
    end--;
}
}
```

⑤ Pairs in Array

for example: array = {1, 7, 5, 3}

pairs are: (1,7), (1,5), (1,3),
(7,5), (7,3),
(5,3)

Simple right, we need two for loop Outer one starts from 0 to end, and inner loop starts from i+1 to end.

Code:

```
for (int i=0; i < nums.length ; i++) {
    for (int j = i+1 ; nums.length ; i++) {
        System.out.print("(" + i + "," + j + ")");
    }
    System.out.println();
}
```

⑥ Print SubArrays.

Here given parameters are the Array itself,
beginning index of Subarray and
ending index of Subarray

for example = array = {1, 5, 3, 5, 9, 1, 4}

beginning index = 3

ending index = 6

So we have to return a subarray that contain {5, 9, 1, 4}

Simple right, we have to initialize an Array of size of the SubArray i.e = (ending - begining) + 1

So iterate over the Array from beginning index to ending index and assign the value to the SubArray with the corresponding index number.

Code

```
int index = 0
int[] SubArray = new int[(beg + end) + 1];
for (int i = beg ; i <= end ; i++){
    SubArray[index] = nums[i];
    index++;
}
```

Print each Sub Arrays.

```
int totalSubArray = 0;
for (int i=0 ; i < nums.length ; i++){
    for (int j=i; j< nums.length ; j++){
        for (int k = i ; k <= j ; k++) {
            System.out.print(k + " ");
        }
        totalSubArray++;
        System.out.println( );
    }
    System.out.println( );
}
System.out.println("total no of SubArrays
                    : " + totalSubArray);
```