Given array of length N reverse the whole array

Reverse is nothing but a combination of multiple swaps

arr[8] = {10, 20, 30, 40, 50, 60, 70, 80}

we need to swap each of the items from beginning of the array with the end of the array.

swap (10, 80)
swap (20, 70)
swap (30, 60)
swap (40, 50)

we can use 2 pointers - one at start and one at ending keep incrementing start & decrementing end point until they cross over.

Pseudocode - T.C - O(n/2) = O(n) S.C - O(1)

```
public static void main(String[] args)
    int n = arr.length;
    int sp = 0;
    int ep = n-1;
    while (sp < ep){
        reverse(arr, sp, ep);
        sp++;
        ep--;
    }
    printArray(arr);
}

private static void
    reverse(int[] arr, int sp,
            int ep) {
    int temp = arr[sp];
    arr[sp] = arr[ep];
    arr[ep] = temp;
}

private static void printArray(int[]
                               arr){
    for(int i=0; i<n; i++){
        print(arr[i]);
    }
}
```

# Reverse part of an array

Given N array elements and [s, e] reverse the array from [s,e].

[3, 7]

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| arr[10] = { | -3 | 4 | 2 | 8 | 3 | 9 | 6 | 2 | 8 | 10 } |

⇓

arr[10] = { -3 4 2 2 6 9 3 8 8 10 }

Point two pointers on 3rd & 7th index and reverse.

```
public static void reverse(int[] nums, int s, int e) {
    int fptr = s;
    int eptr = e;
    while(fptr < eptr){
        int temp = fptr;
        fptr = eptr;
        eptr = temp;
        fptr++;
        eptr--;
    }
}
```

```
reverse(3, 7);
reverse(4, 6);
```

T.C - O(n)

S.C - O(1)

## Rotate the array

Given an array of N elements, rotate from last to first by K times

arr[7] = { 3 -2 1 4 6 9 8 }  K = 3

(0 1 2 3 4 5 6)

↓ 1st rotation

= { 8 3 -2 1 4 6 9 }

↓ 2nd rotation

= { 9 8 3 -2 1 4 6 }

↓ 3rd rotation

arr[7] = { 6 9 8 3 -2 1 4 } → desired output

arr[7] = { 3 -2 1 4 6 9 8 }  K = 3

(0 1 2 3 4 5 6)

↓ to bring the elements from last to first, we basically reverse the array

{ 8 9 6 | 4 1 -2 3 }

reverse first K elements ↓

{ 6 9 8 | 4 1 -2 3 }

↓ reverse the elements after K

{ 6 9 8 3 -2 1 4 } → desired output achieved

pseudocode

```
public static void reverse(int[] nums, int s, int e){
    int fptr = s;
    int eptr = e;
    while(fptr < eptr){
        int temp = nums[fptr];
        nums[fptr] = nums[eptr];
        nums[eptr] = temp;

        fptr++;
        eptr--;
    }
}
public static void main(String[] args){

    //step 1 - reverse whole array
    reverse(nums, 0, n-1);
    //step 2 - reverse first k elements
    reverse(nums, 0, k-1);
    //step3 - reverse elements after k
    reverse(nums, k, n-1);
}
```

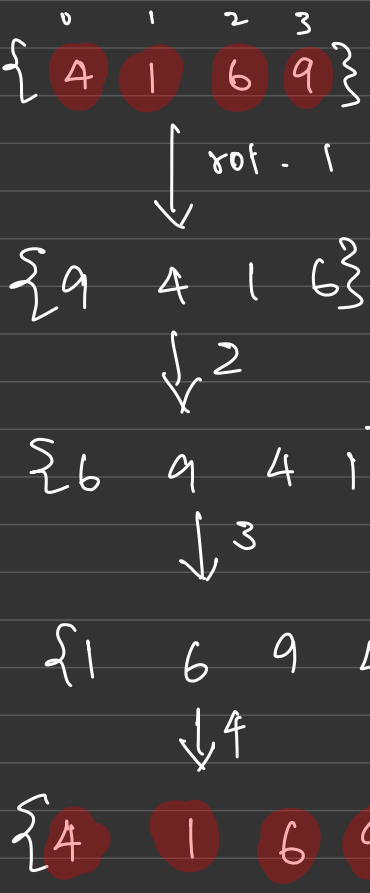
T.C = O(3*N) = O(N)

S.C = O(1)

## What if k is greater than N?

Ex:- k = 10

arr[4] = { 4 1 6 9 }




→ Basically, we get same array after rotating array length (4 in this case) times. or multiples of array length.

for an array of length 4, the number of access iterations equals to the highest multiple of 4 (8 in this case) because 10th iteration is same as 2nd iteration.

4 } same array
8

9

10 → same as 2nd iteration.

| array length | k |
|---|---|
| 5 | 50 — same array |
| 5 | 45 — same array |
| 5 | 48 — only need 3 rotations |

we can say, K = K % n

K = 48 % 5

K = 3

```
public static void main (String[] args) {
    int K = K% n;
    //step 1 - reverse whole array
    reverse (nums, 0, n-1);
    //step 2 - reverse first k elements
    reverse (nums, 0, k-1);
    //step3 - reverse elements after k
    reverse (nums, k, n-1);
}
```

# ArrayList / Dynamic Arrays

```
List<Integer> list = new ArrayList<>();
```

**Add:** T.C - O(1)

```
list.add(10);
list.add(20);
list.add(30);
```

**Access Index** - TC - O(1)

```
S.O.P(list.get(1)); // 20
```

**update the value**

```
list.set(0, 100);
```

T.C - O(1)

**No of elements**

```
list.size();
```

T.C - O(1)

**Remove index**

`list.remove(idx);` - the moment we remove from array list, the elements to the right of the removed element would be moved towards left..

T.C to remove last idx - O(1)

T.C to remove mid idx - O(N)

T.C to remove beginning idx - O(N)