Session 18

Tuesday, 19 August 2025 3:11 PM

Question 189 Leet Code Rotate Array

```
void ArrayCopy(int a[], int b[], int size)
{
    for (int i = 0; i < size; i++)
    {
        b[i] = a[i];
    }
}
void rotate(int* a, int size, int n) {
    int b[size];
    // Copy Fuction
    ArrayCopy(a, b, size);
    n = (n % size);
    for (int i = 0; i < size; i++)
    {
        a[(i + n) % size] = b[i];
    }
}
```

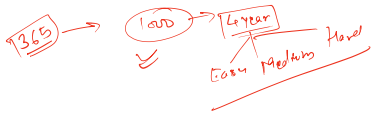
Question 33 : LeetCode

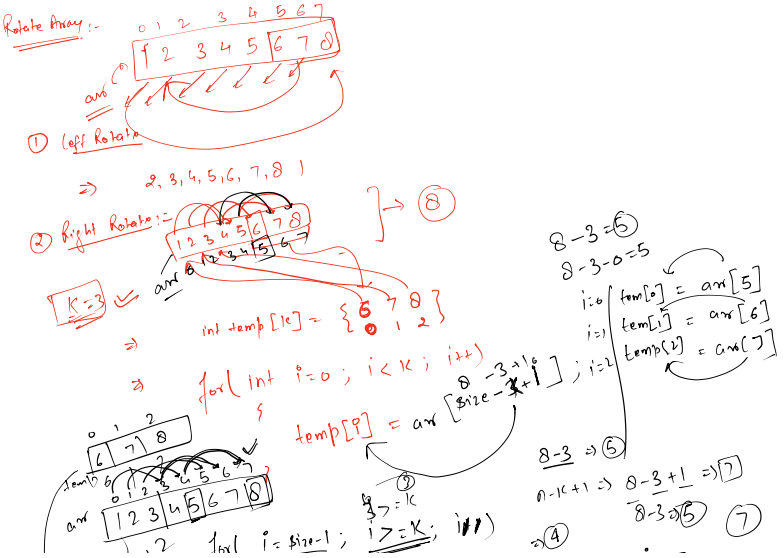
Search in rotated Sorted array

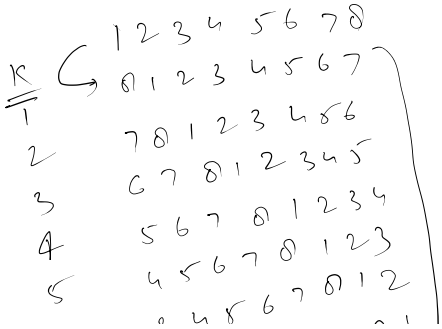
```
int search(int* arr, int numsSize, int k) {
    int low = 0, high = numsSize - 1;
          while (low <= high) {
              int mid = (low + high) / 2;
              // if mid points the target
              if (arr[mid] == k)
                  return mid;
              // if left part is sorted:
              if (arr[low] <= arr[mid]) {
                  if (arr[low] <= k && k <= arr[mid]) {
                      // element exists:
                      high = mid - 1;
                  } else {
                      // element does not exist:
                      low = mid + 1;
                  }
              } else { // if right part is sorted:
                  if (arr[mid] <= k && k <= arr[high]) {
                      // element exists:
                      low = mid + 1;
                  } else {
                      // element does not exist:
                      high = mid - 1;
                  }
              }
          }
          return -1;
}
```

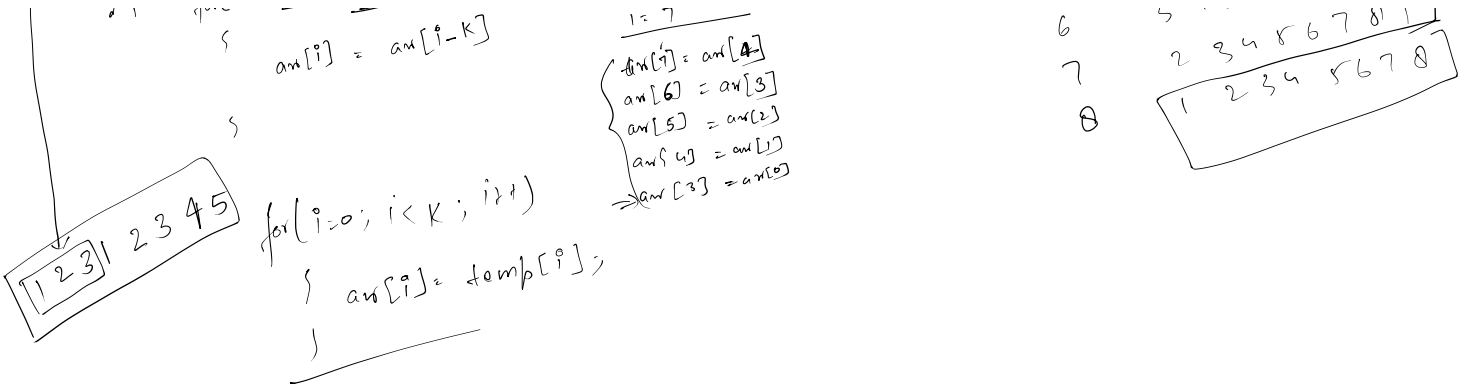
Question 81. Search in Rotated Sorted Array II

```
bool search(int* arr, int numsSize, int k) {
    int low = 0, high = numsSize-1;
    while (low <= high) {
        int mid = (low + high) / 2;

        if (arr[mid] == k) return true;
if (arr[low] == arr[mid]&&arr[mid]==arr[high]){
    low=low+1;
    high=high-1;
    continue;
}
        //if left part is sorted:
        if (arr[low] <= arr[mid]) {
            if (arr[low] <= k && k <= arr[mid]) {
                //element exists:
                high = mid - 1;
            }
            else {
                //element does not exist:
                low = mid + 1;
            }
        }
        else { //if right part is sorted:
            if (arr[mid] <= k && k <= arr[high]) {
                //element exists:
                low = mid + 1;
            }
            else {
                //element does not exist:
                high = mid - 1;
            }
        }
    }
    return false;

}
```

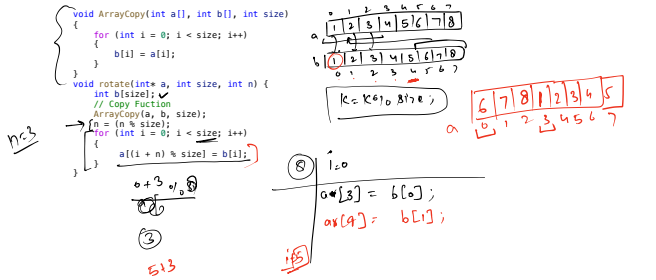


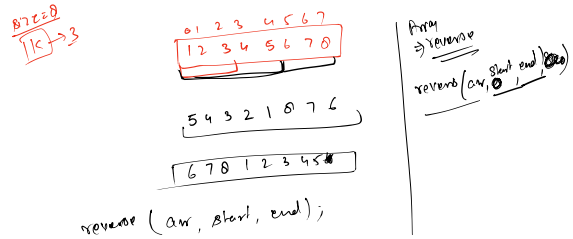


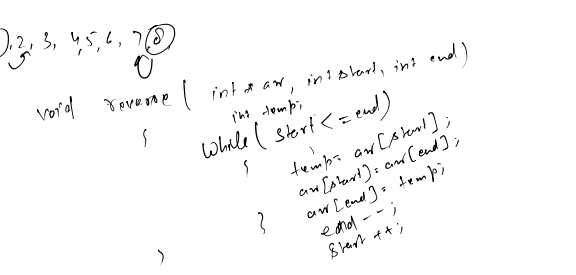




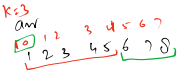
```
void ArrayCopy(int a[], int b[], int size)
{
    for (int i = 0; i < size; i++)
    {
        b[i] = a[i];
    }
}
void rotate(int* a, int size, int n) {
    int b[size];
    // Copy Fuction
    ArrayCopy(a, b, size);
    n = (n % size);
    for (int i = 0; i < size; i++)
    {
        a[(i + n) % size] = b[i];
    }
}
```

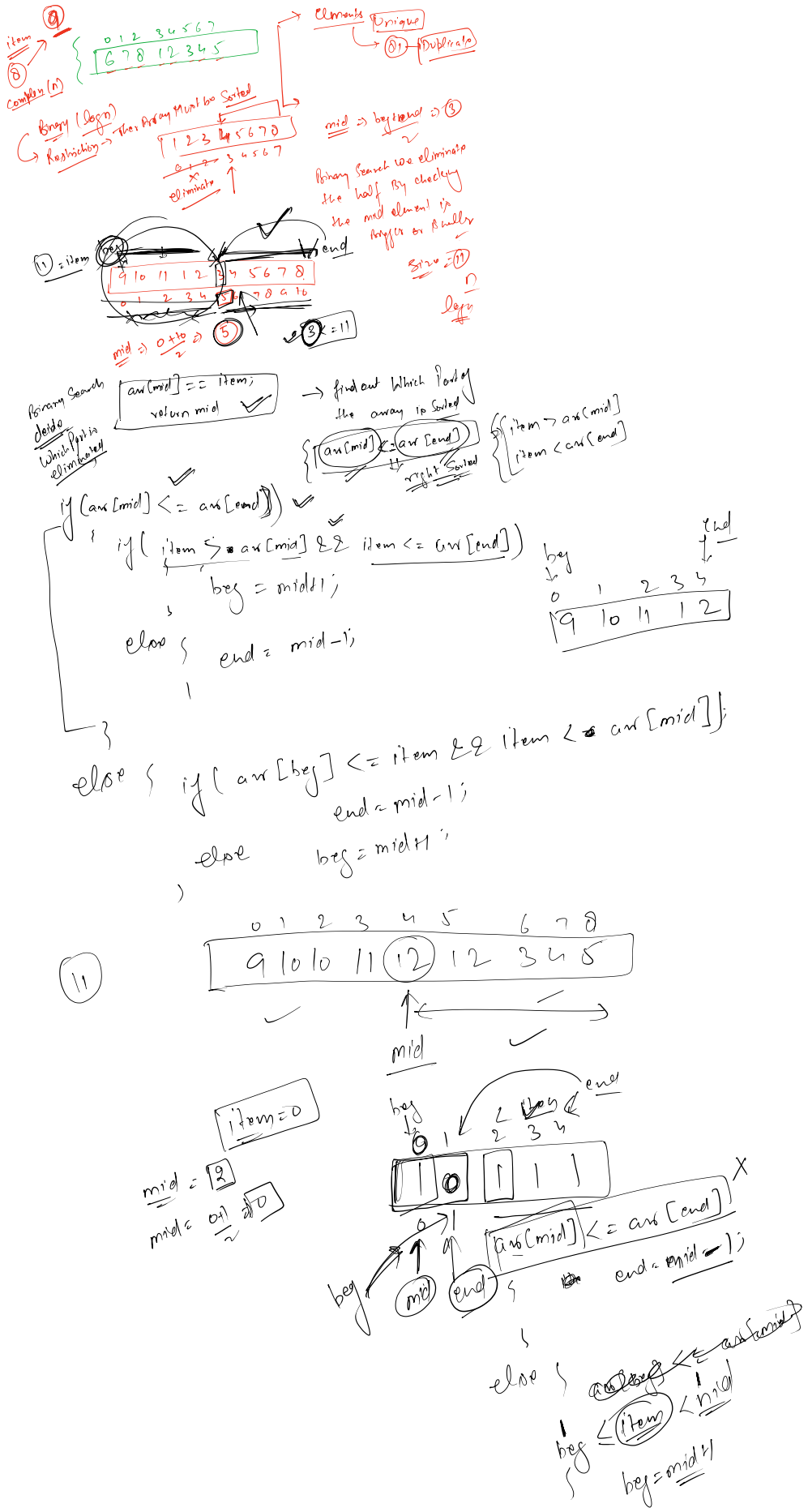







```
void reverse(int * nums , int start, int end){
    int temp;
    while(start < end){
        temp = nums[start];
        nums[start]=nums[end];
        nums[end]=temp;
        start++;
        end--;
    }
}
void rotate(int* nums, int numsSize, int k) {
    k=k%numsSize;
    reverse(nums, 0 , numsSize-k-1);
    reverse(nums, numsSize-k ,numsSize-1);
    reverse(nums , 0 , numsSize-1);
}
```




item
Complexity (n)
Binary (log n)
Restriction
Elements
Unique
Duplicate
mid
beg
end
Binary Search we eliminate the half by checking the mid element is bigger or smaller
Binary Search
arr[mid] == item;
return mid
find out which part of the array is sorted
arr[mid] <= arr[end]
right sorted
item > arr[mid]
item < arr[end]
if (arr[mid] <= arr[end])
if (item > arr[mid] && item <= arr[end])
beg = mid+1;
end = mid-1;
else if (arr[beg] <= item && item < arr[mid])
end = mid-1;
beg = mid+1;
item=0
mid = 2
mid = 0+1 = 0
arr[mid] <= arr[end]
end = mid-1;
else if
item < mid
beg = mid+1