Two Pointers

— Pattern —

Handwritten Notes

by Dipti Verma

learned from CTO Bhaiya

Two Pointer

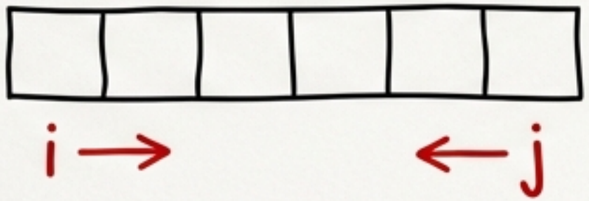
Day-1

A technique where two variables (pointers) traverse a data structure simultaneously from different positions.

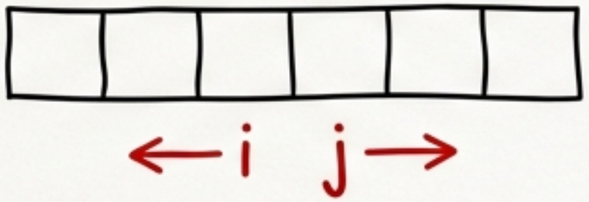
Used in linear data structures → — array
— String
— Linked List

Characteristics: Two pointers either move towards each other, away from each other, or in the same direction.

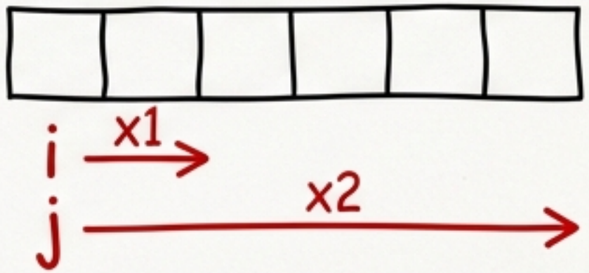
Towards



Away



Same / Fast-Slow

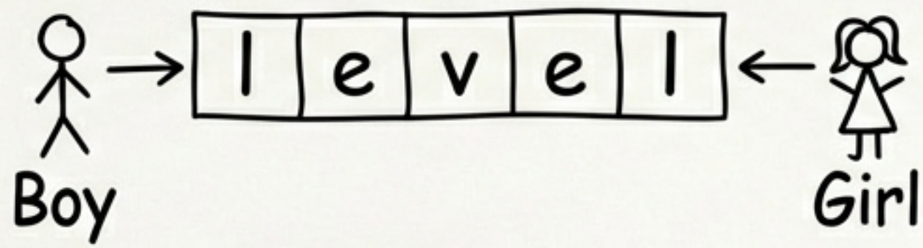


*Efficiently solving problems involving:

→ Comparisons
→ Searching
→ Partitioning

(PCS) shortcut to learn.

Example: Palindrome



Boy sees level > same = Palindrome ✓
Girl sees level

Brute Force:

S = "level"
s.rev = s.reverse()

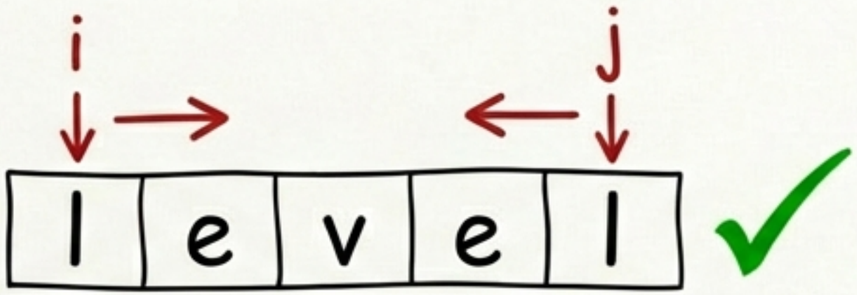
```
S = "level"
s.rev = s.reverse()

for i = 0
   while i < length:
      if s[i] != s.rev[i]:
         print("No")
         return
      i = i + 1
   print("Yes")
```

{ TC = O(n)
SC = O(n) because we are using s.rev. }

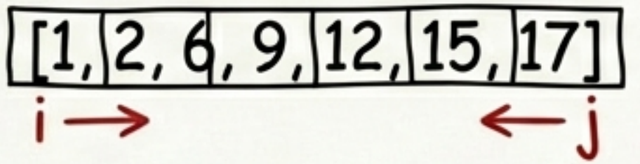
Optimization

```
s = "level"
i = 0
j = s.length - 1
while i < j:
    if s[i] != s[j]:
        print("No")
        return
    i++
    j--
print("Yes")
```



Why use two pointers?

→ No unnecessary iterations
- eg. Target sum in sorted array



Brute Force = ~~$O(n^2)$~~
Two Pointer = $O(n)$ ✓

→ Early stopping as soon as mismatch/condition fails
- eg. Palindrome → l e v e l ←

→ Works in-place → no need to create new arrays or copies
- eg. Palindrome

→ Improves both speed and space efficiency
- Speed: $O(n^2)$ → $O(n)$
- Space: $O(n)$ → $O(1)$

When to use it?

→ Comparing elements from both ends (palindrome, pair sum) ✍️
→ Finding pairs with specific properties (e.g. sum = target) 🔁
→ Removing or merging elements (remove duplicates, merge sorted arrays) 🔧
→ Detecting loops or middle elements (linked list cycle, middle node) ← #Fast-slow

Template

① 1 Input and opposite side pointers

```
a = [1, 4, 3, 2, 9]  left = 0   right = a.length - 1
while left < right:
    if Condition:   left = left + 1   else: right = right - 1
return ans
```

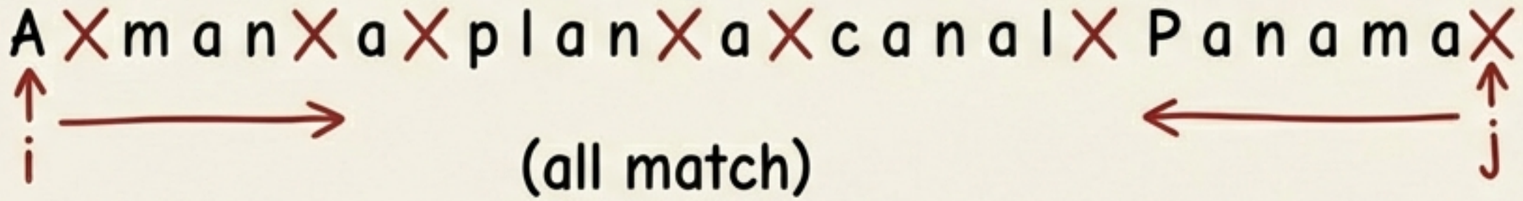
② 2 Inputs and exhaust both pointers (side note: merge array, in sorted array)

```
a = [1,4,3,2,9], b = [9,7,6,2,0,8]  left = 0, right = 0
while left < a.length && right < b.length:
   if condition:  left = left + 1  else: right = right + 1
// (handle remaining elements)
while left < a.length: ...
while right < b.length: ...
```

Day-2 Two Pointers Questions

○ Leetcode Valid Palindrome (125)

Example 1: s = "A man, a plan, a canal: Panama"



Example 2: s = ap a

a p a

condition break

i p → j

Code:

```
i = 0, j = s.length() - 1;
while (i < j) {
   char l = s.charAt(i);    char r = s.charAt(j);
   if (!Character.isLetterOrDigit(l)) {   i++;  continue; }
   if (!Character.isLetterOrDigit(r)) {   j--;  continue; }
→  if (Character.toLowerCase(l) != Character.toLowerCase(r)) {  return false;
   i++;  j--;
}
return true;
```

Reverse String leetcode (344)

Example: s = ["h", "e", "l", "l", "o"] → ["o", "l", "l", "e", "h"]


```
i=0; j=s.length-1;
while(i<j){
      char temp = s[i];
      s[i] = s[j];
      s[j] = temp;
      i++; j--;
}
```

TC = O(n), SC = O(1)

Squares of a Sorted Array leetcode (977)

(non-decreasing order)*

Example: nums = [-4, -1, 0, 3, 10]

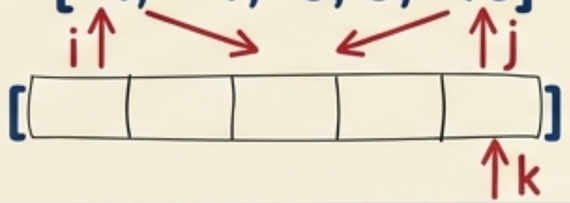
Naive Approach: square → [16, 1, 0, 9, 100]

↓sort [0, 1, 9, 16, 100] → TC = O(n log n)

Optimized Logic Intro

Now, improve T.C

nums = [-4, -1, 0, 3, 10]

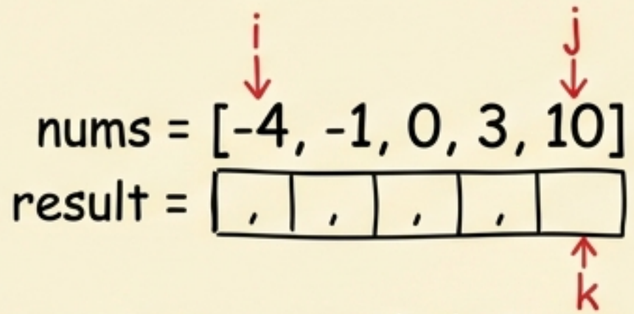


Squares of a Sorted Array (Cont'd) - Optimal Solution

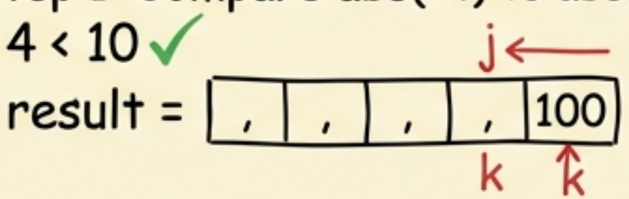
```
int result[] = new int[nums.length];
int i=0, j=nums.length-1, k=result.length-1;
while(i <= j){
    int l = Math.abs(nums[i]);
    int r = Math.abs(nums[j]);
    if(l < r){
        result[k] = (int)Math.pow(r, 2);
        j--;
    } else {
        result[k] = (int)Math.pow(l, 2);
        i++;
    }
    k--;
}
return result;
```

TC = O(n)

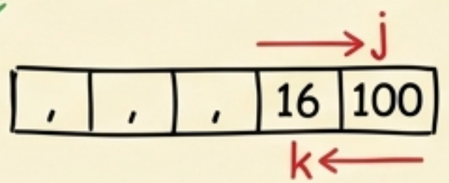
SC = O(1) (bcz result ko return kar rahe hai)



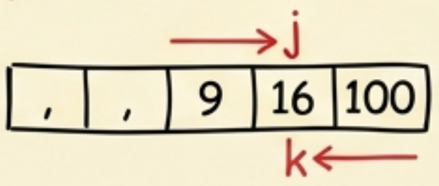
Step 1: Compare abs(-4) vs abs(10)
4 < 10 ✓



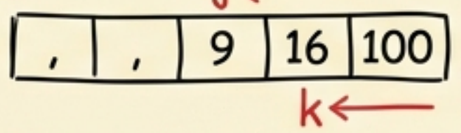
Step 1: Compare abs(-4) vs abs(3) ✓
4 < 10 ✓



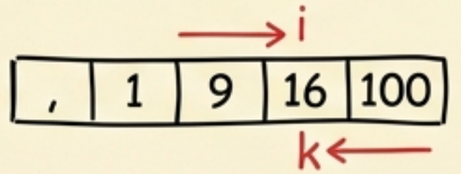
Step 2: Compare abs(-1) vs abs(3) ✓
4 > 3 ✓



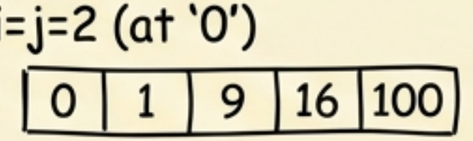
Step 3: Compare abs(-1) vs abs(3) ✓
1 < 3 ✓



Step 4: Compare abs(-1) vs abs(0) ✓
1 > 0 ✓



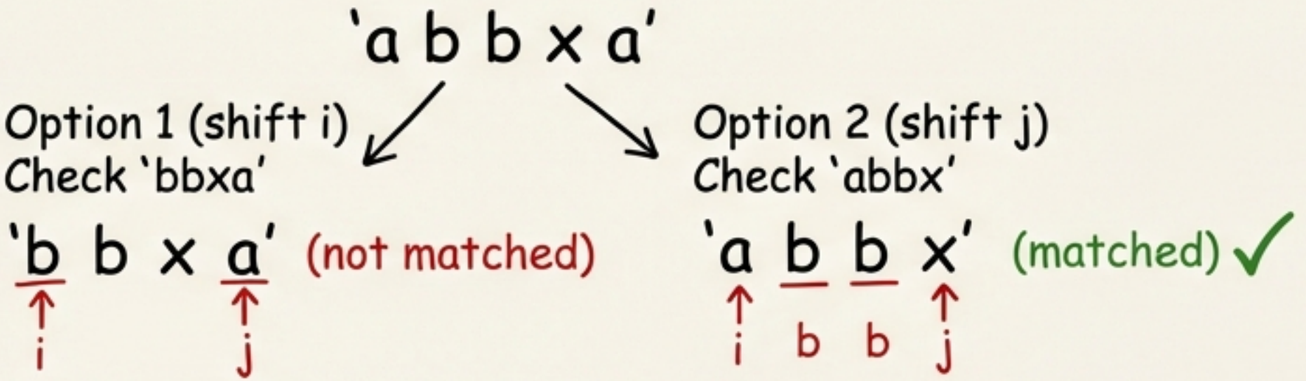
Step 5: `i=j=2 (at '0')



Valid Palindrome II Leetcode (680)

Problem Concept:"Super power: delete at most 1 character & check palindrome or not."

Example: s = 'abbxa'



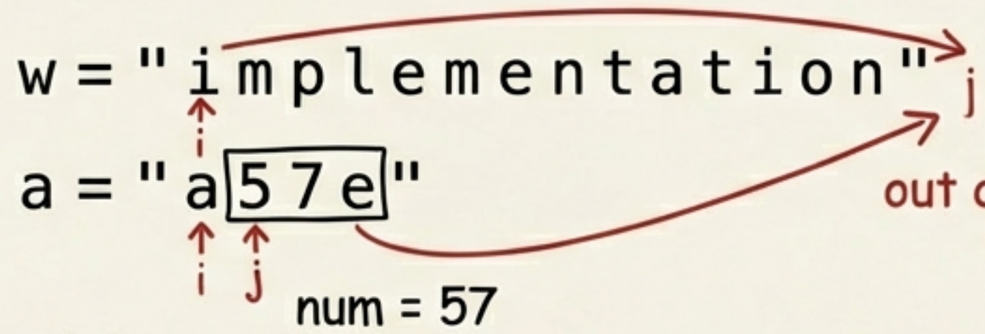

```java
public boolean palindromeHelper(int i, int j, String s){
    while (i < j){
        if (s.charAt(i) != s.charAt(j)) return false;
        i++; j--;
    }
    return true;
}
public boolean validPalindrome(String s){
    int i=0, j=s.length()-1;
    while(i < j){
        char left = s.charAt(i), right = s.charAt(j);
        if (left != right){
            return palindromeHelper(i+1, j, s) || palindromeHelper(i, j-1, s);
        }
        i++; j--;
    }
    return true;
}
```

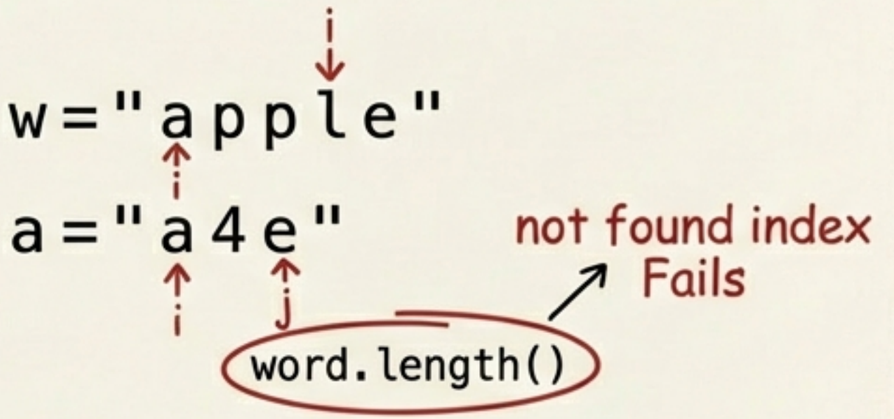
TC=O(n), SC=O(1)

Valid Word Abbreviation (Nextcode)

Eg 1: word = "apple", abbr = "a3e" → true ✓

Eg 2: w = "implementation", a = "i12n" → true ✓






```java
i=0, j=0;
while (i < word.length() && j < abbr.length()) {
    char w_c = word.charAt(i), a_c = abbr.charAt(j);
    if (Character.isDigit(a_c)) {
        if (a_c == '0') return false;
        int curr = 0;
        while (j < abbr.length() && Character.isDigit(abbr.charAt(j))) {
            curr = curr * 10 + (abbr.charAt(j) - '0');
            j++;
        }
        i = i + curr;
    } else {
        if (w_c != a_c) return false;
        i++; j++;
    }
    return i == word.length() && j == abbr.length();
}
```

TC=O(m+n), SC=O(1)

Day-3 Two Pointer Pattern

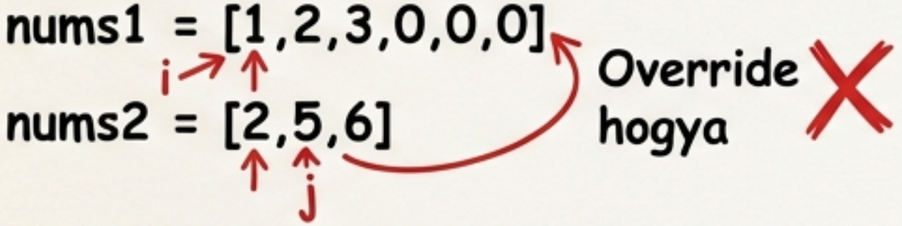
Merge Sorted Array Leetcode (88)

Example: nums1 = [1,2,3,0,0,0], m=3 nums2=[2,5,6], n=3 → o/p: [1,2,2,3,5,6]

Approach 1 (Brute force) Merge nums2 into nums1 and then sort. → But TC=O((n+m)log(n+m))

Approach 2: Optimal (Visual Walkthrough)

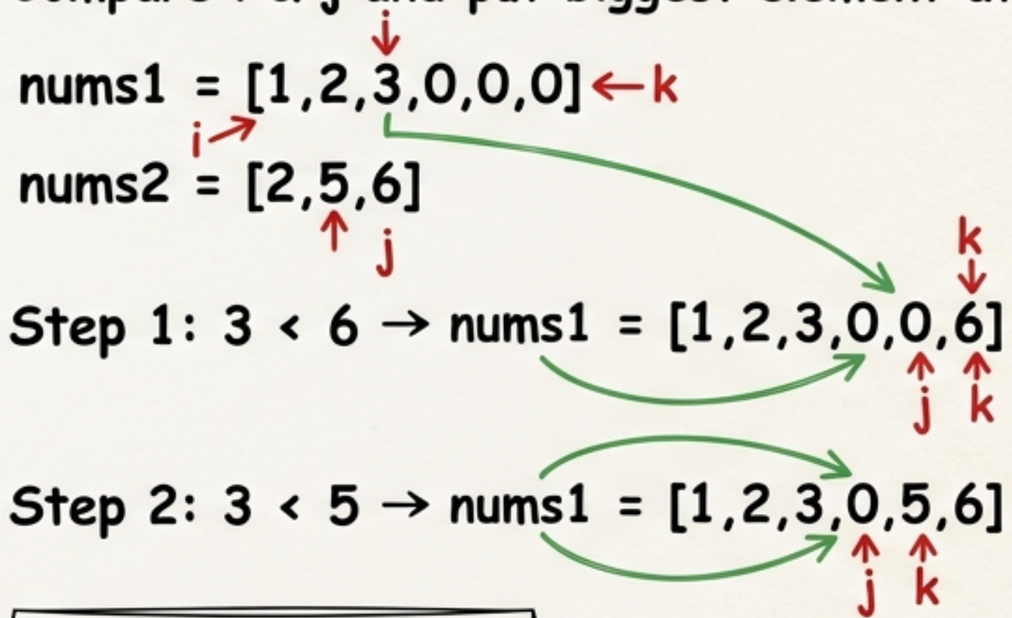
Idea I (Start from beginning - Fails)




```
int i = m-1, j = n-1, k = m+n-1;
while (i >= 0 && j >= 0) {
   if (nums1[i] < nums2[j]) {
      nums1[k--] = nums2[j--];
   } else {
      nums1[k--] = nums1[i--];
}
// remaining elements in j
while (j >= 0) {
   nums1[k--] = nums2[j--];
}
```

Idea II (Start from end - Works!)

Compare i & j and put biggest element at k



TC=O(m+n), SC=O(1)

Count Pairs Whose Sum is less than Target (2824)

Example: nums = [-1, 1, 2, 3, 1], target = 2

Brute force count show s with pairs 3 pairs. TC=O(n^2), SC=O(1)

Optimized Approach:

Sort → [-1, 1, 1, 2, 3], target = 2

i ↗ j ↖

i = -1, j = 3. Sum = 2. Not < target (2) ✗. j--

i = -1, j = 2. Sum = 1 < 2 ✓

Means iske upar wale bhi pairs satisfy honge

Calculation: count = count + (j-i) = 3. i++

→ Pairs (-1,1), (-1,1), (-1,2) are valid

```java
Collections.sort(nums); int i=0, j=nums.size()-1, count=0; while(i < j) {
    if (nums.get(i) + nums.get(j) < target) { count = count + (j-i); i++; }
     else { j--; } }
return count;
```

TC=O(n log n), SC=O(log n)

Two Sum Leetcode (1)

Example: nums=[2,7,11,15], target=9. Output [0,1].

Optimized (HashMap):

at index 0: val=2. ask: (9-2=7)? Not in set. put 2

at index 1: val=7. ask: (9-7=2)? Yes. return indices

Set

value:	index
2:	0

TC=O(n), SC=O(n)

Two Sum II - Input Array is Sorted (167)

Example: numbers = [1,3,4,5,7,10,11], target = 9. [Here, don't use extra space]

[1, 3, 4, 5, 7, 10, 11]
 ↑ i ↑ j

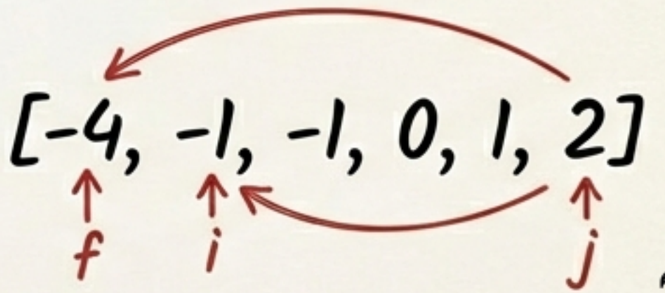
1 + 11 = 12 > 9 → j--
1 + 10 = 11 > 9 → j--
1 + 7 = 8 < 9 → i++
3 + 7 = 10 > 9 → j--
3 + 5 = 8 < 9 → i++
4 + 5 = 9 == 9 ✓ return i and j

```
i=0, j=numbers.length-1;
while(i < j) {
    sum = numbers[i] + numbers[j];
     if (sum > target) j--;
     else if (sum < target) i++;
     else return new int[] {i+1, j+1};
}
return new int[] {-1, -1};
```

3Sum (15)

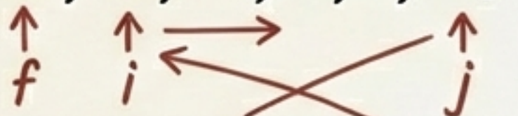
nums = [-1,0,1,2,-1,-4], target = 0. *return triplet value & do not add duplicate triplet

1. Sort: [-4, -1, -1, 0, 1, 2]
2. Fix one element f, then find two other elements (i, j) such that nums[i] + nums[j] == -nums[f].



3Sum - Logic & Code

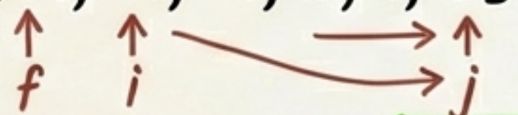
Iteration 1: [-4, -1, -1, 0, 1, 2]



target = 4

sum < 4, i++ No pair found.

Iteration 2: [-4, -1, -1, 0, 1, 2]

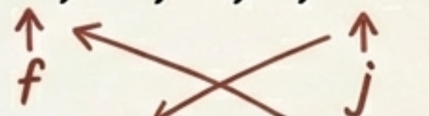


target = 1

`-1 + 2 = 1 ✓ Found [-1, -1, 2]

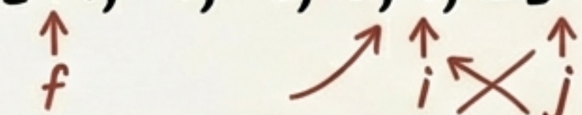
Iteration 3: [-4, -1, -1, 0, 1, 2]

Skip duplicate f values valuuu to avoid duplicate triplets.



i++, j--

Iteration 4: [-4, -1, -1, 0, 1, 2]



target = 0

sum = 3>0, j-- No pair found.

Code:

```
public List<List<Integer>> threeSum(int[] nums) {
    Arrays.sort(nums);
    List<List<Integer>> res = new ArrayList<>();
    for (int f=0; f < nums.length; f++) {
        if (nums[f] > 0) break;
        if (f==0 || nums[f] != nums[f-1]) {  // Skip duplicates
            twoSumHelper(f, nums, res);
        }
    }
    return res;
}
```

```
void twoSumHelper(int f, int[] nums, List<List<Integer>> res) {
    int i = f+1, j = nums.length-1;
    while (i < j) {
        int sum = nums[f] + nums[i] + nums[j];
        if (sum > 0) j--;
        else if (sum < 0) i++;
        else {
            res.add(Arrays.asList(nums[f], nums[i++], nums[j--]));
            // Skip duplicate i's and j's
            while (i < j && nums[i] == nums[i-1]) i++;
            while (i < j && nums[j] == nums[j+1]) j--;
        }
    }
}
```

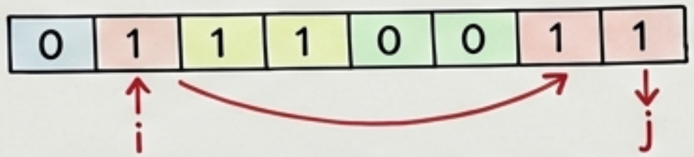
TC=O(n^2), SC=O(log n) (for sort)

Day-4 Two Pointer Pattern (3 Popular Interview Q)

Problem 1: Sort two colors

Example: nums = [0,1,1,1,0,0,1,1] ⟶ [0,0,0,1,1,1,1,1]

Optimized Logic: One pass. Pointer i starts at 0, j starts at end.
If nums[i]==1 and nums[j]==0, swap.

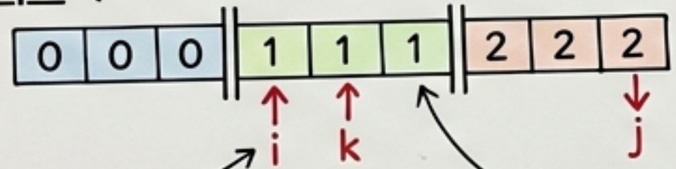


Problem 2: Sort Colors Leetcode (75)

↳ This problem is also called Dutch National Flag Problem ✓

Example: nums = [2,0,2,1,1,0] → [0,0,1,1,2,2]

Concept (The Three Pointers)



i says: j ke piche sara = 0
j says: j ke aage sara = 2

k is the current element being considered.

if (nums[k] == 0): swap with nums[i], i++, k++
if (nums[k] == 2): swap with nums[j], j--
if (nums[k] == 1): k++

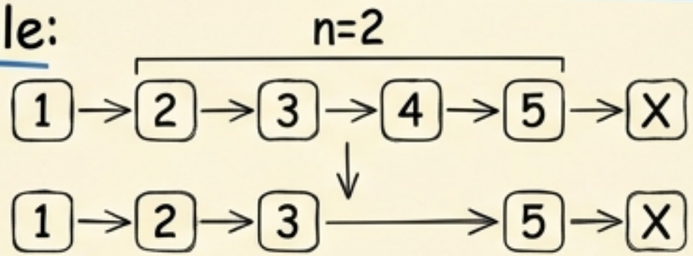
Code (Dutch National Flag):

```
int i=0, j=nums.length-1, k=0;
while(k ⩽ j) {
    if (nums[k] == 0) {
        swap(nums[k++],nums[i++]);
    } else if (nums[k] == 2) {
        swap(nums[k], nums[j--]);
    } else { // nums[k] == 1
        k++;
    }
}
```

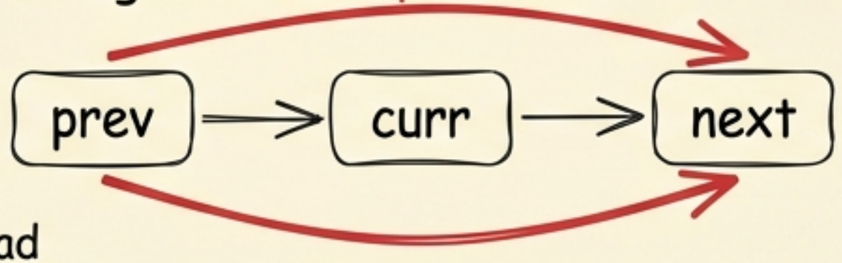
TC=O(n), SC=O(1)

Remove Nth Node From End of List (19)

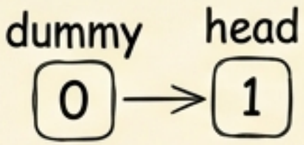
Example:



Deletion Logic Visual: prev.next



Approach I: Two Pass

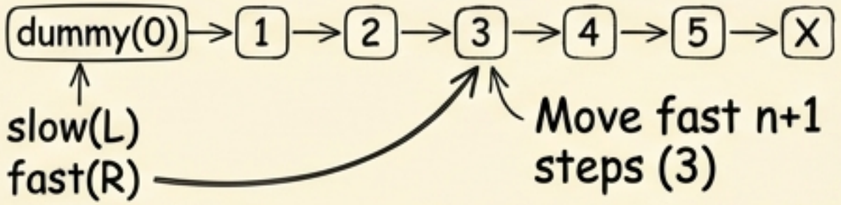


1. Iterate once to find total length L.
2. Calculate target node from start: d = L - n.
3. Iterate again d-1 times to find the node before the target and delete.

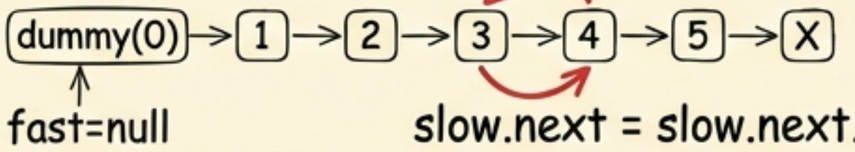
TC=O(L), SC=O(1)

Approach II: Optimized (One Pass)

- Concept: Use two pointers, fast (R) and slow (L).
- Visual Walkthrough:



3. Now, move fast and slow together until fast is null.



Code (One Pass):

```
ListNode dummy = new ListNode(0);
dummy.next = head;
ListNode l = dummy, r = dummy;
// Move r n+1 steps ahead
for (int i=0; i<n+1; i++) {
    r = r.next;
}
// Move both until r is null
while (r != null) {
    l = l.next;
    r = r.next;
}
l.next = l.next.next;
return dummy.next;
}
```

TC=O(L), SC=O(1)

Day-5 Two Pointer Pattern (Interview Questions)

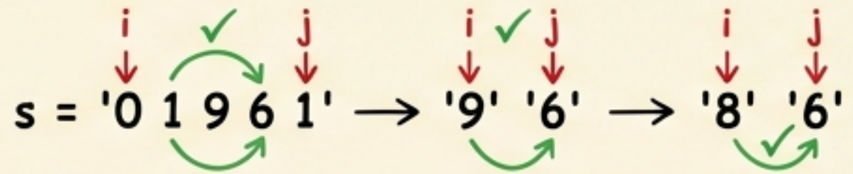
Problem 1: Strobogrammatic Number

Looks the same after 180° rotation. (0,1,8,6,9)

0↔0 ✓ 1↔1 ✓ 8↔8 ✓ 6↔9 ✓
0↔0 1↔1 8↔8 6↔9

Use two pointers, i and j, from the ends. Check if char(i) and char(j) are a valid pair using a map.

s = "19861"



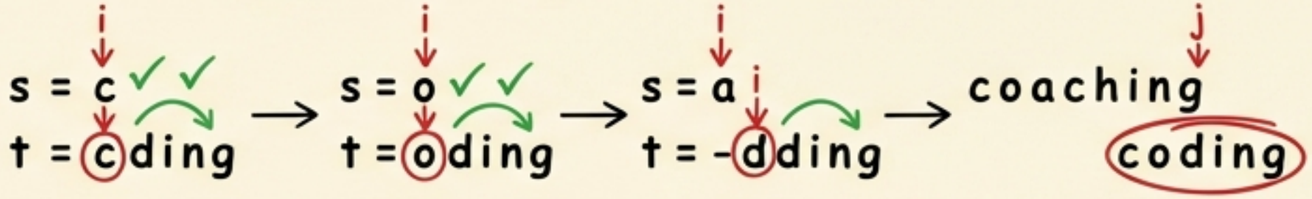

```
Map<Character,Character> map = new HashMap<>();
map.put('0','0'); map.put('1','1'); map.put('8','8');
map.put('6','9'); map.put('9','6');
int i=0, j=s.length()-1;
while(i <= j) {
    if(!map.containsKey(s.charAt(i)) ||
            map.get(s.charAt(i)) != s.charAt(j)){
        return false;
    i++; j--;
}
return true;
```

TC=O(N), SC=O(1)

Problem 2: Append Characters to String to Make Subsequence (2486)

Find the longest prefix of t that is a subsequence of s.

s = "coaching", t = "coding"



Characters to append = t.length() - j 6 - 2 = 4

```
int i=0, j=0;
while(i < s.length() && j < t.length()) {
    if(s.charAt(i) == t.charAt(j)) {
        j++;
    i++;
}
return t.length() - j;
```

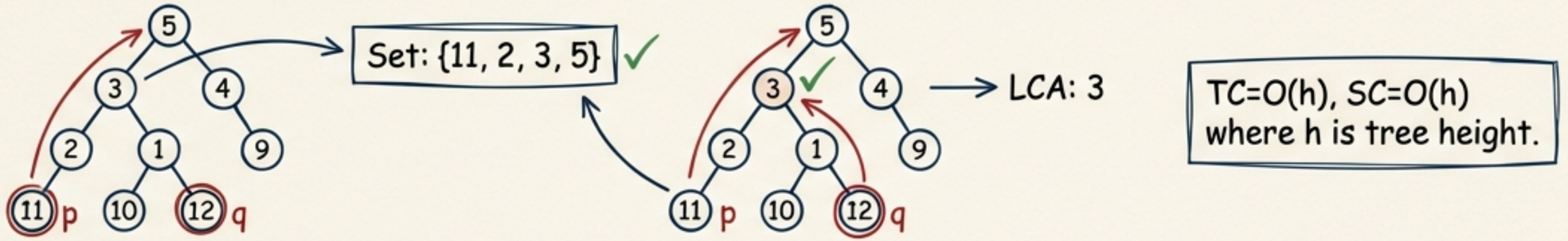
TC=O(min(N, M)), SC=O(1)

Lowest Common Ancestor of a Binary Tree III (1650)

Context: Nodes have a parent pointer.

Approach 1: Brute Force (with extra space)

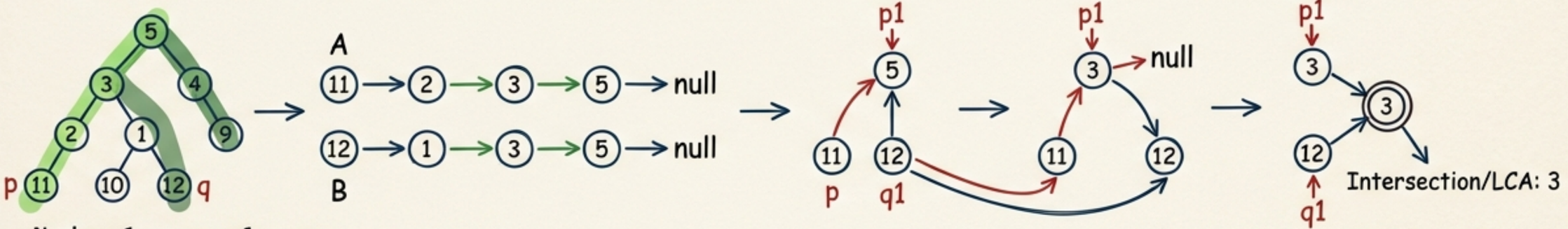
Logic: Store all parents of node p in a HashSet. Then, traverse up from q and return the first parent found in the set.



TC=O(h), SC=O(h)
where h is tree height.

Approach 2: Optimized (Two Pointers, O(1) space)

Concept: Treat the paths from p to the root and q to the root as **two linked lists**. We need to find their intersection.




```
Node p1 = p, q1 = q;
while (p1 != q1) {
    p1 = (p1 == null) ? q : p1.parent;
    q1 = (q1 == null) ? p : q1.parent;
}
return p1;
```

TC=O(h), SC=O(1)

Day-6 Two Pointer Pattern (On String)

Reverse Words in a String Leetcode (151)

① Return a string of the words in **reverse order concatenated by a single space.**
② Do not include any extra spaces.
Example: s = " hello world " → o/p = "world hello"

Approach I: Using Built-in Functions (Brute Force)

s = " hello world the "
↓
trim() → "hello world the" (trim function used to remove spaces at starting & ending)
↓
split(" +") → ["hello", "world", "the"] (split function used to separate...)
↓
reverse array → ["the", "world", "hello"]
↓
join(" ") → "the world hello"

```
String trimmed = s.trim();
String[] arr = trimmed.split("\\s+");
int i=0, j=arr.length-1;
while(i<j) {
    // Standard array reverse swap logic here
    String temp = arr[i];
    arr[i] = arr[j];
    arr[j] = temp;
    i++; j--;
}
return String.join(" ", arr);
```

TC=O(n), SC=O(n)

Approach II: Without using any function

Trick: Reverse String + Reverse each Word

1. Remove starting/ending/extra spaces.
2. Reverse the whole string: hello world the → eht dlrow olleh
3. Reverse each word individually: eht dlrow olleh → the world hello

Reverse Words in a String - Approach II Code

```
// 1. Clean extra spaces
StringBuilder sb = new StringBuilder();
String s = "    hello    world    "; // assume this is input
s = s.trim();
for(int i = 0; i < s.length(); i++){
    if(s.charAt(i) == ' ' && s.charAt(i-1) == ' '){
        continue;
    }
    sb.append(s.charAt(i));
}
```

→ e.g., "   hello   world   " → "hello world"

```
// 2. Reverse the entire string
int i = 0, j = sb.length() - 1;
while (i < j) {
    char temp = sb.charAt(i);
    sb.setCharAt(i, sb.charAt(j));
    sb.setCharAt(j, temp);
    i++; j--;
}
```

i, j

→ "hello world"
→ "dlrow olleh"

```
// 3. Word by word reverse
int start = 0, end = 0;
while (start < sb.length()) {
    // Find end of word
    while(end < sb.length() && sb.charAt(end) != ' ') {
        end++;
    }
    // Reverse the word from p1=start to p2=end-1
    int p1 = start, p2 = end - 1;
    while(p1 < p2) {
        // (swap logic for p1 and p2)
        char temp = sb.charAt(p1);
        sb.setCharAt(p1, sb.charAt(p2));
        sb.setCharAt(p2, temp);
        p1++; p2--;
    }
    start = end + 1;
    end = start;
}
return sb.toString();
```

start

p1

p2

"dlrow olleh"
→ "world hello"

TC = O(n)
SC = O(n) (for StringBuilder)