heycoach

9 LEETCODE CHALLENGES YOU MUST FACE IN 2023!

Swipe for more >

1. Median of Two Sorted Arrays

Given two sorted arrays nums1 and nums2 of size m and n respectively, return the median of the two sorted arrays.

The overall run time complexity should be O(log(m + n)).

Example 1:

Input: nums1 = [1,3], nums2 = [2]

Output: 2.00000

Explanation: merged array = [1,2,3] and median is 2.

Example 2:

Input: nums1 = [1,2], nums2 = [3,4]

Output: 2.50000

Explanation: merged array = [1,2,3,4] and median is (2 + 3) / 2 = 2.5.

Optimal Solution:

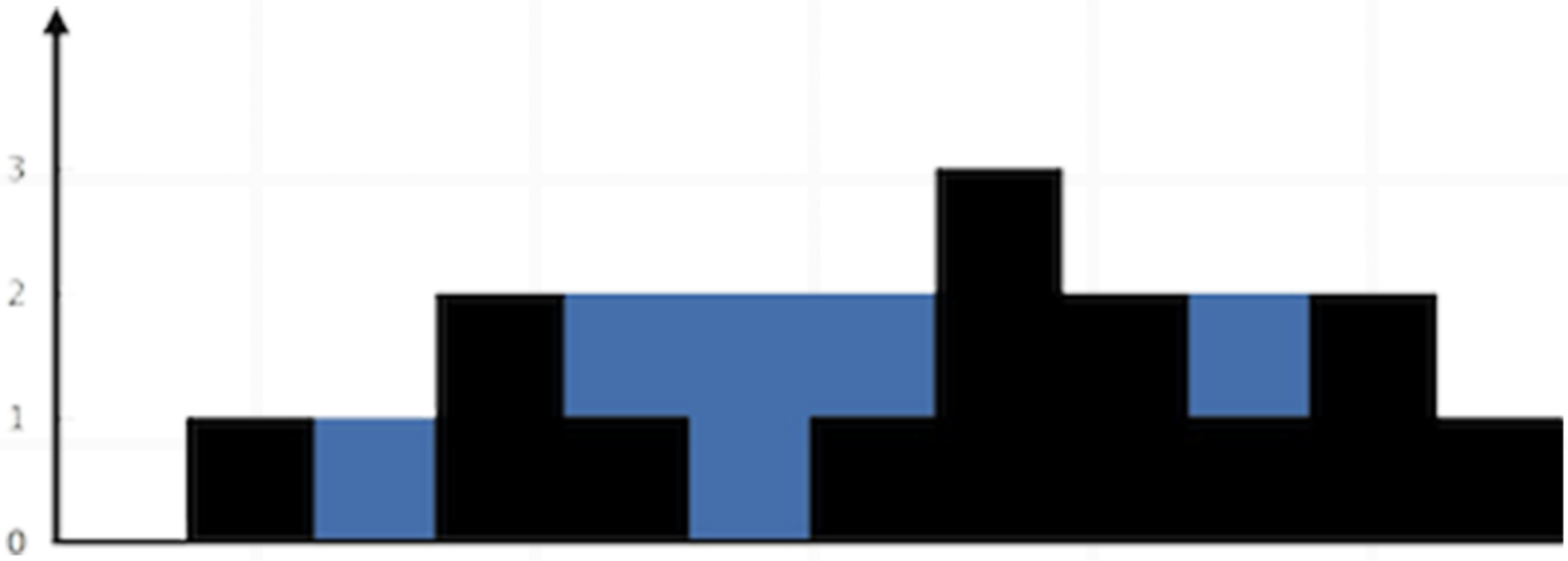
```cpp
class Solution {
public:
    double findMedianSortedArrays(vector<int>& nums1, vector<int>& nums2) {
        int n1 = nums1.size(), n2 = nums2.size();
        if(n1 > n2) return findMedianSortedArrays(nums2, nums1);

        int start = 0, end = n1;


        while(start <= end){
            int mid = (start + end) / 2;
            int cut1 = mid;
            int cut2 = (n1 + n2 + 1) / 2 - cut1;

            int l1 = (cut1 > 0) ? nums1[cut1 - 1] : INT_MIN;
            int l2 = (cut2 > 0) ? nums2[cut2 - 1] : INT_MIN;
            int r1 = (cut1 < n1) ? nums1[cut1] : INT_MAX;
            int r2 = (cut2 < n2) ? nums2[cut2] : INT_MAX;

            if(l1 <= r2 && l2 <= r1){
                return ((n1 + n2) % 2) ? max(l1, l2) : (max(l1, l2) + min(r1, r2)) / 2.0;
            }

            if(l1 > r2) end = mid - 1;
            else start = mid + 1;
        }

        return -1;
    }
};
```

2. Trapping Rain Water

Given n non-negative integers representing an elevation map where the width of each bar is 1, compute how much water it can trap after raining.

Example:

Input: height = [0,1,0,2,1,0,1,3,2,1,2,1]



Output: 6

Explanation: The above elevation map (black section) is represented by array [0,1,0,2,1,0,1,3,2,1,2,1]. In this case, 6 units of rain water (blue section) are being trapped.

Optimal Solution:

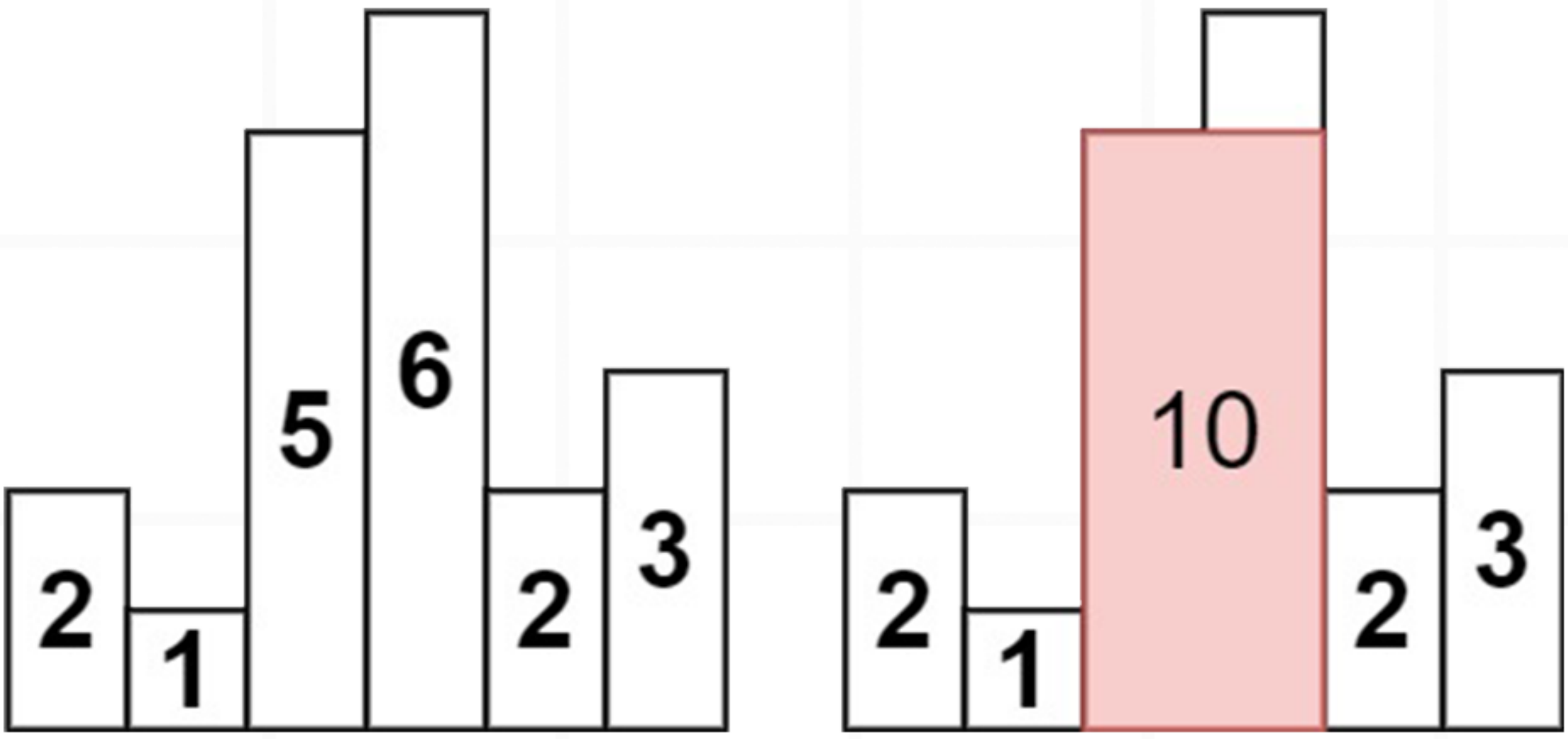
```cpp
class Solution {
public:
    int trap(vector<int>& height) {
        int n = height.size(), res = 0;
        int maxLeft = height[0], maxRight = height[n - 1];
        int left = 1, right = n - 2;


        while(left <= right){
            int waterTrapped;


            if(maxLeft <= maxRight){
                waterTrapped = maxLeft - height[left];
                maxLeft = max(maxLeft, height[left]);
                left++;
            }
            else{
                waterTrapped = maxRight - height[right];
                maxRight = max(maxRight, height[right]);
                right--;
            }


            if(waterTrapped > 0) res += waterTrapped;
        }


        return res;
    }
};
```

3. Largest Rectangle in Histogram

Given an array of integers heights representing the histogram's bar height where the width of each bar is 1, return the area of the largest rectangle in the histogram.

Example:

Input: heights = [2,1,5,6,2,3]



Output: 10

Explanation: The above is a histogram where width of each bar is 1.

The largest rectangle is shown in the red area, which has an area = 10 units.

Optimal Solution:

```cpp
class Solution {
public:
    int largestRectangleArea(vector<int>& heights) {
        stack<int>st;
        int n = heights.size(), res = 0;


        for(int i = 0; i <= n; i++){

            while(st.size() && (i == n || heights[i] < heights[st.top()])){
                int height = heights[st.top()];
                st.pop();
                int width;
                if(st.size()) width = i - st.top() - 1;
                else width = i;
                res = max(res, height * width);
            }

            if(i < n) st.push(i);
        }

        return res;
    }
};
```

4. First Missing Positive

Given an unsorted integer array nums, return the smallest missing positive integer.

You must implement an algorithm that runs in O(n) time and uses O(1) auxiliary space.

Example 1:

Input: nums = [1,2,0]

Output: 3

Explanation: The numbers in the range [1,2] are all in the array.

Example 2:

Input: nums = [3,4,-1,1]

Output: 2

Explanation: 1 is in the array but 2 is missing.

Optimal Solution:

```cpp
class Solution {
public:
    int firstMissingPositive(vector<int>& nums) {
        int n = nums.size();
        for(int i = 0; i < n; i++){
            if(nums[i] <= 0) nums[i] = n + 1;
        }


        for(int i = 0; i < n; i++){
            int ind = abs(nums[i]) - 1;
            if(ind < n && nums[ind] > 0) nums[ind] = -nums[ind];
        }


        for(int i = 0; i < n; i++){
            if(nums[i] > 0) return i + 1;
        }


        return n + 1;
    }
};
```

5. Merge k Sorted Lists

You are given an array of k linked-lists lists, each linked-list is sorted in ascending order.

Merge all the linked-lists into one sorted linked-list and return it.

Example:

Input: lists = [[1,4,5],[1,3,4],[2,6]]

Output: [1,1,2,3,4,4,5,6]

Explanation: The linked-lists are:

[

1->4->5,

1->3->4,

2->6

]

merging them into one sorted

list:

1->1->2->3->4->4->5->6

Optimal Solution:

```cpp
class Solution {
    typedef pair<int,ListNode*> p;


    ListNode* merge(ListNode* a, ListNode* b){
        ListNode* dummyHead = new ListNode(-1);
        ListNode* c = dummyHead;

        while(a && b){
            if(a->val <= b->val){
                c->next = a;
                a = a->next;
            }
            else{
                c->next = b;
                b = b->next;
            }
            c = c->next;
        }


        c->next = (a) ? a : b;
        c = dummyHead->next;
        return c;
    }


    ListNode* mergeLists(vector<ListNode*>& lists, int start, int end){
        if(start>end) return NULL;
        if(start==end) return lists[start];


        int mid = start + (end-start)/2;
        ListNode *a = mergeLists(lists,start,mid);
        ListNode *b = mergeLists(lists,mid+1,end);


        return merge(a,b);
    }


public:
    ListNode* mergeKLists(vector<ListNode*>& lists) {
        if(lists.size()==0) return NULL;
        return mergeLists(lists,0,lists.size()-1);
    }
};
```

6. Minimum Window Substring

Given two strings s and t of lengths m and n respectively, return the minimum window substring of s such that every character in t (including duplicates) is included in the window. If there is no such substring, return the empty string "".

The testcases will be generated such that the answer is unique.

Example:

Input: s = "ADOBECODEBANC", t = "ABC"

Output: "BANC"

Explanation: The minimum window substring "BANC" includes 'A', 'B', and 'C' from string t.

Optimal Solution:

```cpp
class Solution {
public:
    string minWindow(string s, string t) {
        unordered_map<char,int> target, window;
        for(auto ch : t) target[ch]++;

        int l = 0, r = 0, resLen = INT_MAX;
        pair<int,int> res = {-1, -1};
        int have = 0, need = target.size();

        while(r < s.size()){
            char ch = s[r];
            window[ch]++;
            if(window[ch] == target[ch]) have++;

            while(have == need){
                if(r - l + 1 < resLen){
                    res = {l, r};
                    resLen = r - l + 1;
                }
                char leftCh = s[l];
                window[leftCh]--;
                if(window[leftCh] < target[leftCh]) have--;
                l++;
            }

            r++;
        }

        if(resLen == INT_MAX) return "";
        return s.substr(res.first, res.second - res.first + 1);
    }
};
```

7. Sliding Window Maximum

You are given an array of integers nums, there is a sliding window of size k which is moving from the very left of the array to the very right. You can only see the k numbers in the window. Each time the sliding window moves right by one position.

Return the max sliding window.

Example:

Input: nums = [1,3,-1,-3,5,3,6,7], k = 3

Output: [3,3,5,5,6,7]

Explanation:

Window position	Max
[1 3 -1] -3 5 3 6 7	3
1 [3 -1 -3] 5 3 6 7	3
1 3 [-1 -3 5] 3 6 7	5
1 3 -1 [-3 5 3] 6 7	5
1 3 -1 -3 [5 3 6] 7	6
1 3 -1 -3 5 [3 6 7]	7

Optimal Solution:

```cpp
class Solution {
public:
    vector<int> maxSlidingWindow(vector<int>& nums, int k) {
        vector<int>res;
        deque<int> d;


        for(int i = 0; i < nums.size(); i++){
            while(d.size() && nums[d.back()] <= nums[i])
                d.pop_back();


            d.push_back(i);


            if(i >= k - 1){
                res.push_back(nums[d.front()]);
                if(d.front() == i - k + 1) d.pop_front();
            }
        }


        return res;
    }
};
```

8. Find Median from Data Stream

The median is the middle value in an ordered integer list. If the size of the list is even, there is no middle value, and the median is the mean of the two middle values.

For example, for arr = [2,3,4], the median is 3.
For example, for arr = [2,3], the median is (2 + 3) / 2 = 2.5.

Implement the MedianFinder class:

MedianFinder() initializes the MedianFinder object.
void addNum(int num) adds the integer num from the data stream to the data structure.
double findMedian() returns the median of all elements so far. Answers within 10-5 of the actual answer will be accepted.

Example:

Input:

["MedianFinder", "addNum", "addNum", "findMedian", "addNum", "findMedian"]

[[], [1], [2], [], [3], []]

Output:

[null, null, null, 1.5, null, 2.0]

Explanation:

MedianFinder medianFinder = new MedianFinder();

medianFinder.addNum(1); // arr = [1]

medianFinder.addNum(2); // arr = [1, 2]

medianFinder.findMedian(); // return 1.5 (i.e., (1 + 2)

Optimal Solution:

```cpp
class MedianFinder {
    priority_queue<int> maxHeap;
    priority_queue<int, vector<int>, greater<int>> minHeap;

    void rebalance(){
        int diff = maxHeap.size() - minHeap.size();
        if(abs(diff) <= 1)
            return;

        if(maxHeap.size() > minHeap.size()){
            minHeap.push(maxHeap.top());
            maxHeap.pop();
        }

        else{
            maxHeap.push(minHeap.top());
            minHeap.pop();
        }
    }
public:
    MedianFinder() {


    }

    void addNum(int num) {
        if(maxHeap.empty() || num <= maxHeap.top())
            maxHeap.push(num);

        else
            minHeap.push(num);

        rebalance();
    }

    double findMedian() {
        if(maxHeap.size() > minHeap.size())
            return maxHeap.top();

        else if(maxHeap.size() < minHeap.size())
            return minHeap.top();

        return (maxHeap.top() + minHeap.top()) / 2.0;
    }
};
```

9. Longest Valid Parentheses

Given a string containing just the characters '(' and ')', return the length of the longest valid (well-formed) parentheses substring.

Example:

Input: s = ")()())"

Output: 4

Explanation: The longest valid parentheses substring is "()()".

Optimal Solution:

```cpp
class Solution {
public:
    int longestValidParentheses(string s) {
        int res = 0, open = 0, close = 0;
        for(int i = 0; i < s.size(); i++){
            if(s[i] == '(') open++;
            else close++;
            if(open == close) res = max(res, 2 * open);
            if(open < close) open = 0, close = 0;
        }


        open = 0, close = 0;
        for(int i = s.size() - 1; i >= 0; i--){
            if(s[i] == '(') open++;
            else close++;
            if(open == close) res = max(res, 2 * open);
            if(open > close) open = 0, close = 0;
        }

        return res;
    }
};
```

Struggling with LeetCode hard problems?

You're not alone. These are the types of questions that can make or break your **MAANG** interviews.

Don't let tough coding questions stump you. With **HeyCoach**, you're not just prepared you're **MAANG**-ready!

1000+ engineers chose HeyCoach to land a job at a top product company.

Engineers land an average CTC of **27 LPA** with HeyCoach

Engineers got an average of **300%** hike through HeyCoach

100% placement assistance

SIGNUP NOW

(Link in Caption)