**Nearest Smaller Tower:-**

Medium Accuracy: 68.97% Submissions: 7K+ Points: 4

Given an array where each element (**arr[i]**) represents the height of the tower. Find for each tower, the nearest possible tower that is shorter than it. You can look left or right on both sides.  
**Note :**

* If two smaller towers are at the same distance, pick the smallest tower.
* If two towers have the same height then we choose the one with a smaller index.

**Example 1:**

**Input:**

arr[] = {1,3,2}

**Output:**

{-1,0,0}

**Explanation:**

For **0th** Index : no tower is smallest, so **-1**.

For **1st** Index : For 3, here 1 & 2 both are

small & at a same distance, so we will pick 1,

beacuse it has smallest value, so **0(Index)**

For **2nd** Index : here 1 is smaller, so **0(Index)**

So the final output will be which consistes

Indexes are {-1,0,0}.

**Example 2:**

**Input:**

arr[] = {4,8,3,5,3}

**Output:**

{2,2,-1,2,-1}

**Explanation:**

For **0th** Index : here 3 is the smaller, so **2(Index)**

For **1st** Index : For 8, here 4 & 3 both are

small & at a same distance, so we will pick 3, so **2(Index)**

For **2nd** Index : no tower is smallest, so **-1**.

For **3rd** Index : For 5, here 3 & 3 both are

small & at a same distance, so we will pick

**3**(2nd Index) because it smaller Index, so **2(Index)**

For **4th** Index : no tower is smallest, so **-1**.

So the final output will be which consistes

Indexes are {2,2,-1,2,-1}.

**Your Task:**  
You don't need to read input or print anything. Your task is to complete the function **nearestSmallerTower()**which takes an array of heights of the towers as input parameter and returns an array of **indexes** of the nearest smaller tower. If there is no smaller tower on both sides then**return -1** for that tower.

**Expected Time Complexity:** O(N)  
**Expected Auxiliary Space:** O(N)

**Constraints:**  
1 <= N <= 105  
1 <= arr[i] <= 105

**Code :-**

1. //{ Driver Code Starts
2. //Initial Template for C++
3. #include <bits/stdc++.h>
4. using namespace std;
5. // } Driver Code Ends
6. //User function Template for C++
7. class Solution
8. {
9. public:
10. vector<int> nearestSmallerTower(vector<int> &arr)
11. {
12. int minl, minr, findl=false, findr=false, find=false;
13. vector<int> ans(arr.size());
14. int mini = INT\_MAX;
16. for(auto i:arr)
17. mini = min(mini, i);
19. for(auto i=0; i<arr.size(); ++i){
20. if(arr[i]==mini){
21. ans[i]=-1;
22. }
23. }
25. for(int i=0; i<arr.size(); ++i){
26. int j=i-1, k=i+1;
27. if(ans[i]==-1)
28. continue;
29. while(j>=0 || k<arr.size()){
30. findl=false, findr=false, find=false;
31. if(j>=0 && arr[j]<arr[i]){
32. minl=arr[j];
33. findl=true;
34. }
35. if(k<arr.size() && arr[k]<arr[i]){
36. minr=arr[k];
37. findr=true;
38. }
40. if(findl==false && findr==false && k>arr.size() && arr[j]==arr[i]){
41. ans[i] = ans[j];
42. find=true;
43. break;
44. }
45. else if(findl==true && findr==true){
46. if(minl < minr)
47. ans[i] = j;
48. else if(minr < minl)
49. ans[i] = k;
50. else
51. ans[i] = j;
52. find=true;
53. break;
54. }
55. else if(findl==true){
56. ans[i]=j;
57. find=true;
58. break;
59. }
60. else if(findr==true){
61. ans[i]=k;
62. find=true;
63. break;
64. }
66. if(j>=0) j--;
67. if(k<arr.size()) k++;
68. }
69. if(findl==false && findr==false && find==false)
70. ans[i]=-1;
71. }
72. return ans;
73. }
74. };
75. //{ Driver Code Starts.
76. int main()
77. {
78. int t;
79. cin >> t;
80. while (t--)
81. {
82. int n;
83. cin >> n;
84. vector<int> v(n);
85. for (int i = 0; i < n; i++)
86. {
87. cin >> v[i];
88. }
89. Solution ob;
90. vector<int> ans = ob.nearestSmallerTower(v);
91. for (int i = 0; i < n; i++)
92. {
93. cout << ans[i] << " ";
94. }
95. cout << "\n";
96. }
97. return 0;
98. }
99. // } Driver Code Ends

**S.C :- O(1)**

**Logic :-**

Basically a 3-pointer algorithm where starting from the current array item, in another loop we move towards left and right using 2 more pointers and after geeting immediate nearest smaller value( of tower height) we get the answer and break from the inner loop. NOTE :- If we have traversed from the curtrent index to the extreme right index(and cooresponding left index) but we did not find any smaller values then if we get any value equals to the current value on going towards left, we will copy the same answer( memoisation technique / to optimize).