Test 1

Recurrence Relation for Tower of Hanoi Problem

Send Feedback

The recurrence relation capturing the optimal execution time of the Towers of Hanoi problem with n discs is :

This problem may have one or more correct answers

- [ ] T(n) = 2T(n − 2) + 2
- [ ] T(n) = 2T(n − 1) + n
- [ ] T(n) = 2T(n/2) + 1
- [x] T(n) = 2T(n − 1) + 1 ✔

Hurray! Correct Answer

Complexity of different operations in a sorted array.

Send Feedback

Which of the following operations is not O(1) for an array of sorted data. You may assume that array elements are distinct.

This problem may have one or more correct answers

- [ ] Find the ith largest element
- [x] Delete an element ✔
- [ ] Find the ith smallest element
- [ ] All of the above

Hurray! Correct Answer

Solution Description

The worst-case time complexity for deleting an element from an array can become O(n).

Complexity of a Recurrence Relation

Send Feedback

Which one of the following correctly determines the solution of the recurrence relation with T(1) = 1?

T(n) = 2T(n/2) + Logn

This problem may have one or more correct answers

- [x] O(N) ✔
- [ ] O(NlogN)
- [x] O(N^2)
- [ ] O(log N)

Solution Description

Please find the images for the step by step explanation in the following links:.
https://files.codingninjas.in/step1-10997.jpg
https://files.codingninjas.in/step2-10998.jpg

The log series summation can be understood from here: https://stackoverflow.com/questions/44231116/is-complexity-ologn-logn-2-logn-4-logn-8-log2-olog

Does s contain t ?

Send Feedback

Given two string s and t, write a function to check if s contains all characters of t (in the same order as they are in string t).
Return true or false.
Do it recursively.
E.g. : s = "abchjsgsuohhdhyrikkknddg" contains all characters of t="coding" in the same order. So function will return true.
Input Format :

```
Line 1 : String s
Line 2 : String t

Output Format :

true or false

Sample Input 1 :

abchjsgsuohhdhyrikkknddg
coding

Sample Output 1 :

true

Sample Input 2 :

abcde
aeb

Sample Output 2 :

false
```

```
public class Solution {
   public static boolean checkSequence(String a, String b) {

       int i=0, j=0;
       while(i<a.length() && j<b.length()){
           if(a.charAt(i)==b.charAt(j)){
               i++;
               j++;
           }
           else
               i++;
       }
       if(j==b.length())
           return true;
       return false;
   }
}
```

```
Maximum Profit on App
Send Feedback
You have made a smartphone app and want to set its subscription price such
that the profit earned is maximised. There are certain users who will
subscribe to your app only if their budget is greater than or equal to
your price.
You will be provided with a list of size N having budgets of subscribers
and you need to return the maximum profit that you can earn.
Lets say you decide that price of your app is Rs. x and there are N number
of subscribers. So maximum profit you can earn is :
```

m * x

where m is total number of subscribers whose budget is greater than or equal to x.

Input format :

Line 1 : N (No. of subscribers)
Line 2 : *Budget* of subscribers (separated by space)

Output Format :

Maximum profit

Constraints :
1 <= *N* <= 10^6
1 <=budget[i]<=9999

Sample Input 1 :

4
30 20 53 14

Sample Output 1 :

60

Sample Output 1 Explanation :
Price of your app should be Rs. *20* or Rs. *30*. For both prices, you can get the profit Rs. *60*.

Sample Input 2 :

5
34 78 90 15 67

Sample Output 2 :

201

Sample Output 2 Explanation :
Price of your app should be Rs. *67*. You can get the profit Rs. *201* (i.e. *3* * 67).

```
import java.util.Arrays;

public class solution {

   public static int maximumProfit(int budget[]) {
       // Write your code here
       Arrays.sort(budget);
       // System.out.println();
       int i = 0;
       int maxProfit = 0, profit = 0;
       while (i < budget.length) {
           int n = budget.length - i;
           profit = budget[i] * n;
```

```
            if (profit > maxProfit) {
                maxProfit = profit;
            }
            int temp = i;
            while (i < budget.length && budget[temp] == budget[i]) {
                i++;
            }
        }
        return maxProfit;

    }

}
```

Split Array
Send Feedback
Given an integer array *A* of size N, check *if* the input array can be splitted in two parts such that -

- *Sum* of both parts is equal
- *All* elements in the input, which are divisible by 5 should be in same group.
- *All* elements in the input, which are divisible by 3 (but not divisible by 5) should be in other group.
- *Elements* which are neither divisible by 5 nor by 3, can be put in any group.

Groups can be made with any set of elements, i.e. elements need not to be continuous. And you need to consider each and every element of input array in some group.
Return true, *if* array can be split according to the above rules, else return false.
Note : *You* will get marks only *if* all the test cases are passed.
Input Format :

Line 1 : *Integer* N (size of array)
Line 2 : *Array A* elements (separated by space)

Output Format :

true or false

Constraints :

1 <= *N* <= 50

Sample Input 1 :

```
2
1 2
```

Sample Output 1 :

```
false
```

```
Sample Input 2 :

3
1 4 3

Sample Output 2 :

true

public class solution {

   public static boolean helper(int arr[], int n, int start, int lsum, int
rsum)
   {
       // If reached the end
       if (start == n)
           return lsum == rsum;

       // If divisible by 5 then add to the left sum
       if (arr[start] % 5 == 0)
           lsum += arr[start];

       // If divisible by 3 but not by 5
       // then add to the right sum
       else if (arr[start] % 3 == 0)
           rsum += arr[start];

       // Else it can be added to any of the sub-arrays
       else

           // Try adding in both the sub-arrays (one by one)
           // and check whether the condition satisfies
           return helper(arr, n, start + 1, lsum + arr[start], rsum)
           || helper(arr, n, start + 1, lsum, rsum + arr[start]);

       // For cases when element is multiple of 3 or 5.
       return helper(arr, n, start + 1, lsum, rsum);
   }

   // Function to start the recursive calls
   static boolean splitArray(int arr[])
   {
       // Initially start, lsum and rsum will all be 0
       return helper(arr, arr.length, 0, 0, 0);
   }

}
```