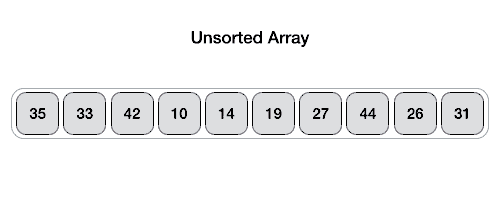
Divide and Conquer Practice Problems For A1 and A2

## Practice Problem 1

**Quick sort algorithm:**

You are given a orwne dimensional array that may contain both positive and negative integers, sort the

numbers using Quicksort strategy. 

A simple example with last element pivoting is shown in animation.

Details here

<https://en.wikipedia.org/wiki/Quicksort>

Your tasks are divided in three phases

1. Basic Quicksort with last or first element pivoting
2. Quicksort with random pivoting
3. Printing index of the corresponding numbers

Input format:

1. Number of test case T
2. Foreach test case, take integer input N, represents the number of elements
3. Input N double numbers.

Output format:

1. Test Case number
2. Print the sorted number and their corresponding index.

Sample Input:

2

4   
4 3 2 1

6  
2 -1 2 3 4 -5

Sample Output:

Case 1#  
1 2 3 4  
3 2 1 0

Case 2#  
-5 -1 2 2 3 4  
 5 1 0 2 3 4

# Practice Problem 2

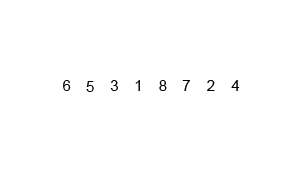
**Merge sort Problem:**

You are given a one dimensional array that may contain both positive and negative integers, sort the

numbers using Merge sort strategy.

Merge sort follows three steps

**Step 1** − if it is only one element in the list it is already sorted, return.  
**Step 2** − divide the list recursively into two halves until it can no more be divided.  
**Step 3** − merge the smaller lists into new list in sorted order.



Trivially during merge phase additional memory is used; now in this assignment your task is to do the whole procedure without using additional memory. Which is also regarded as inplace merge sort.

<https://en.wikipedia.org/wiki/MergeSort>

Input format:

1. Number of test case T
2. Foreach test case, take integer input N, represents the number of elements
3. Input N double numbers.

Output format:

1. Test Case number
2. Print the sorted number and their corresponding index.

Sample Input:

2

4   
4 3 2 1

6  
2 -1 2 3 4 -5

Sample Output:

Case 1#  
1 2 3 4  
3 2 1 0

Case 2#  
-5 -1 2 2 3 4  
 5 1 0 2 3 4

## Practice Problem 3

Today on a lecture about strings Gerald learned a new definition of string equivalency. Two strings *a* and *b* of equal length are called *equivalent* in one of the two cases:

1. They are equal.
2. If we split string *a* into two halves of the same size *a*1 and *a*2, and string *b* into two halves of the same size *b*1 and *b*2, then one of the following is correct:
   1. *a*1 is equivalent to *b*1, and *a*2 is equivalent to *b*2
   2. *a*1 is equivalent to *b*2, and *a*2 is equivalent to *b*1

As a home task, the teacher gave two strings to his students and asked to determine if they are equivalent.

Gerald has already completed this home task. Now it's your turn!

**Input**

The first two lines of the input contain two strings given by the teacher. Each of them has the length from 1 to 200 000 and consists of lowercase English letters. The strings have the same length.

**Output**

Print "YES" (without the quotes), if these two strings are equivalent, and "NO" (without the quotes) otherwise.

**Examples**

**input**

aaba

abaa

**output**

YES

**input**

aabb

abab

**output**

NO

**Note**

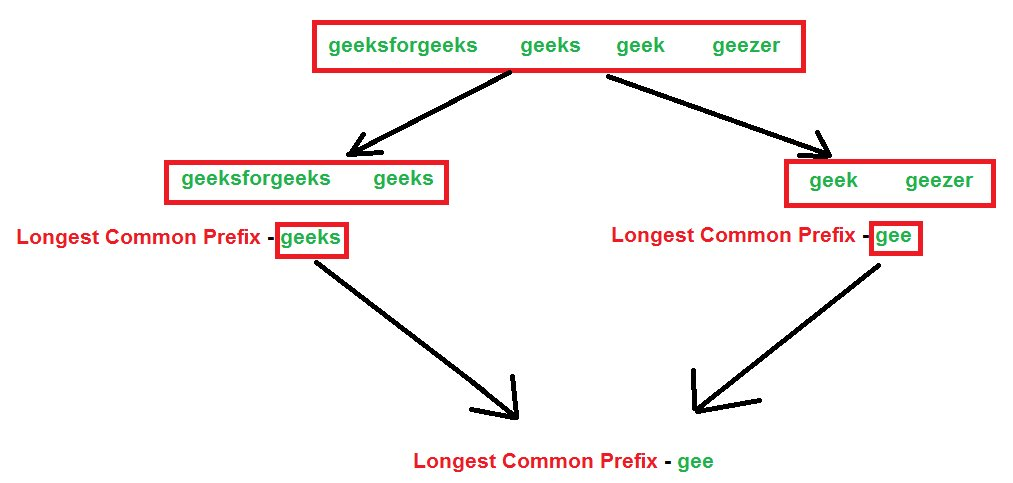
In the first sample you should split the first string into strings "aa" and "ba", the second one — into strings "ab" and "aa". "aa" is equivalent to "aa"; "ab" is equivalent to "ba" as "ab" = "a" + "b", "ba" = "b" + "a".

In the second sample the first string can be splitted into strings "aa" and "bb", that are equivalent only to themselves. That's why string "aabb" is equivalent only to itself and to string "bbaa".

# Practice Problem 4

**Problem definition:**

Longest Common Prefix Calculation:



You are given N strings, You are required to find longest common prefix among those strings. Check the above case for better understanding divide and conquer phase.

# 

# Practice Problem 5

**Largest Rectangular Area in a Histogram**

Find the largest rectangular area possible in a given histogram where the largest rectangle can be made of a number of contiguous bars. For simplicity, assume that all bars have same width and the width is 1 unit.

For example, consider the following histogram with 7 bars of heights {6, 2, 5, 4, 5, 1, 6}. The largest possible rectangular possible is 12 (see the below figure, the max area rectangle is highlighted in red)



Input format:

1. Number of test case T
2. Foreach test case, take integer input N, represents the number of elements
3. Input N Integer numbers.

Output format:

1. Test Case number
2. Print the area and their starting and ending index.

Sample Input:

2

4   
4 3 2 1

7  
6 2 5 4 5 1 6

Sample Output:

Case 1#  
9  
0 1

Case 2#  
12

# Practice Problem 6

**Maximum Subarray Sum Computation:**

You are given a one dimensional array that may contain both positive and negative integers, find the sum of contiguous subarray of numbers which has the largest sum.

For example, if the given array is {-2, -5, **6, -2, -3, 1, 5**, -6}, then the maximum subarray sum is 7 (see highlighted elements).

**The naive method** is to run two loops; the time complexity of the Naive method is O(n^2).

Using **Divide and Conquer** approach, we can find the maximum subarray sum in O(nLogn) time.

Input format:

1. Number of test case T
2. Foreach test case, take integer input N, represents the number of elements
3. Input N double numbers.

Output format:

1. Test Case number
2. Print Maximum subarray sum and the corresponding starting and ending index

Sample Input:

3

4   
1 2 3 4

6  
2 -1 2 3 4 -5

8  
-2, -5, 6, -2, -3, 1, 5, -6

Sample Output

Case 1#  
10  
0 3

Case 2#  
10  
0 4

Case 3#  
7  
2 6

# Practice Problem 7

**Find the Peak element in an Array**

Given an array of integers. Find a peak element in it. An array element is peak if it is NOT smaller than its neighbors. For corner elements, we need to consider only one neighbor.

For example, for input array {5, 10, 20, 15}, 20 is the only peak element.

For input array {10, 20, 15, 2, 23, 90, 67}, there are two peak elements: 20 and 90.

Note that we need to return any one peak element.

Following corner cases give better idea about the problem.

**1)** If input array is sorted in strictly increasing order, the last element is always a peak element. For example, 50 is peak element in {10, 20, 30, 40, 50}.

**2)** If input array is sorted in strictly decreasing order, the first element is always a peak element. 100 is the peak element in {100, 80, 60, 50, 20}.

**3)** If all elements of input array are same, every element is a peak element.

It is clear from above examples that there is always a peak element in input array in any input array.

A **simple solution** is to do a linear scan of array and as soon as we find a peak element, we return it. The worst case time complexity of this method would be O(n).

But your task is to use **Divide and Conquer**to find a peak in O(Logn) time.

Input format:

1. Number of test case T
2. Foreach test case, take integer input N, represents the number of elements
3. Input N Integer numbers.

Output format:

1. Test Case number
2. Print the peak element

Sample Input:

5

4   
5, 10, 20, 15

7  
10, 20, 15, 2, 23, 90, 67

5  
10, 20, 30, 40, 50

5  
100, 80, 60, 50, 20

5  
5 5 5 5 5

Sample Output

Case 1#  
20

Case 2#  
20 / 90

Case 3#  
50

Case 4#  
100

Case 5#  
5

## 

## Practice Problem 8

**Multiplication Using Divide and Conquer Approach**

Details for for both n digit integer numbers. You have to do that for two n digit numbers represented in character array/String

### ***Multiplication***

*Problem MULTIPLICATION*

***Input:*** *two n-digit integers a and b*

***Output:*** *product of a and b*

*example:*

*1980 = a  
 x 2315 = b  
 ---------  
 9900  
 1980  
 5940  
 + 3960  
 ---------  
 4573700 = a x b*

*This is the algorithm you learned in grade school. Notice it takes O(n^2) time.*

### ***Dividing the Problem***

*We divide each integer into two halves.*

*aL = 19 | 80 = aR  
 |  
 bL = 23 | 15 = bR  
  
  
 aL aR  
 x bL bR  
 -----------------------------  
 aL bR aR bR  
+ aL bL aR bL  
 -----------------------------  
 aL bL aL bR + aR bL aR bR*

*So our algorithm is to compute aL bL, aL bR, aR bL, and aR bR, and add.*

*T(n) <= 4 T(n / 2) + O(n)  
  
 T(n) = O(n^2)*

***Algorithm*** *Divide-Mult(a,b)****:***

***if*** *a or b has one digit,* ***then:***

*return a \* b.*

***else:***

*Let n be the number of digits in max{a, b}.*

*Let aL and aR be left and right halves of a.*

*Let bL and bR be left and right halves of b.*

*Let x1 hold Divide-Mult(aL, bL).*

*Let x2 hold Divide-Mult(aL, bR).*

*Let x3 hold Divide-Mult(aR, bL).*

*Let x4 hold Divide-Mult(aR, bR).*

***return*** *x1\*10n + (x2 + x3)\*10n/2 + x4.*

***end of if***

<http://www.cs.cmu.edu/~cburch/pgss99/lecture/0721-divide.html>