\n --> this is to represent enter

\ backward slash

Set:

If we try to add duplicate element i.e. element which are already present in Set it simply ignore those values. Null values are accepted by Set. The order in which elements are put is or add is not retained in set i.e. element wouldn’t be store in that order. But internally a hash would be generated & the values would be stored with respect to generated hash.

Pre-increment/decrement: let i=1; if(--i==0) now here it would decrease the value of i & then check. While if the statement is if(i--==0) then here it would first use the value of i & then decrease. Let i=10; if(i++==i) this condition is false reason is that i++ increments the value at memory location and returns old value. i returns the current value at memory location of i. if(++i==i) true.

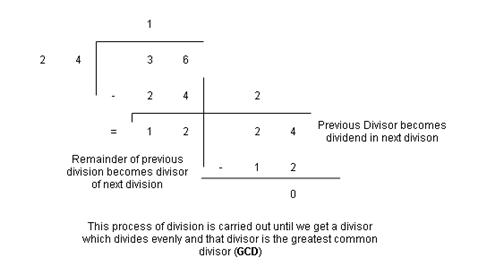
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

GCD & LCM:

So, we make use of the Division Method to find out the gcd.

In this method, we make use of the two given numbers by selecting one to be a dividend and the other to be a divisor and perform division. If we get remainder in the first division (i.e the dividend is not evenly divided by the divisor), we select the remainder to be the next divisor and the previous divisor to be the next dividend and then keep on performing till we find a number which evenly divides the dividend.

The first number that performs even division (give remainder 0), is our required gcd.

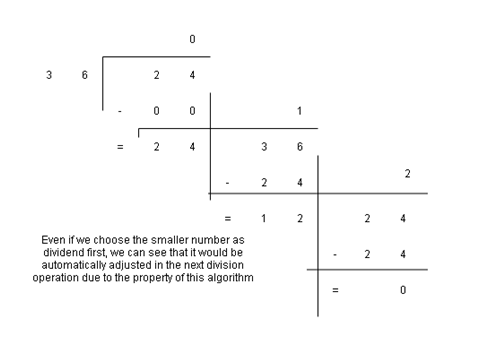


Above is the division method illustrated for two given numbers 24 and 36. We can see that 12 comes out as the first divisor which performs even division and it is indeed the gcd of these two numbers.

As far as the decision to select the dividend and divisor is, we do not necessarily have to find and put the greater number as dividend and smaller number as the divisor.

We can just put any of the two as either dividend or divisor and the algorithm will further take care of it in the very first step (in case the smaller number is the dividend first).

**This is resolved in the following way:**



There exists a relation between LCM and GCD

gcd\*lcm=n1\*n2

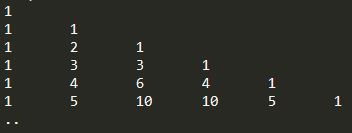
This could be further modified in to:

lcm= (n1\*n2)/gcd

We simply use this formula to get out Least Common Multiple.

But there is a catch. We used n1 and n2 for our calculations directly and therefore, they are modified and not original. To overcome this issue, before the calculation, we store the original values of n1 and n2 in temporary variables to be used later.

Q.



Up until now, we have been initializing outer loop iterator 'i' as 1 and running it until it was equal to input value n. Now, we make a slight change to it and understand why.

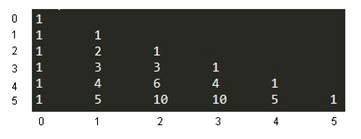
[1,5] **->**means 5 objects (1,2,3,4,5)

[0,5) **->**also means 5 objects (0,1,2,3,4) **->** so 'i' = 0 to i< n  will still display 'n' rows

In this program, we are going to start 'i' from 0 to less than n. We also start 'j' from 0 but run it till less than or equal to 'i' only because for each row, the number of columns is one more than row number. The output shows symmetry in the elements it prints. First and last entry on each row is '1'. We know that C\_0^n = 1= C\_n^n.

Comparing the last row to

C\_0^5=1;C\_1^n=5;C\_2^n=10



And after 10, the pattern is symmetric.

We also know that the mathematical relation C\_k^n=C\_(n-k)^n also holds true; which is the basis on which we can say that the pattern is symmetric, because then: C\_3^5=10; C\_4^5=5; C\_1^5=1

Now to get the required pattern printed in our code, using our outer and inner loop, we are going to use the formula C\_(k+1)^n=(C\_k^n.(n-k))/(k+1).

We modify it to suit our 'i' and 'j' variables as:C\_(j+1)^i=(C\_j^i.(i-j))/(j+1) .

Any Base Addition?

We are given two numbers in a particular number system denoted by 'b' and two numbers denoted by 'n1' and 'n2'.

In decimal number system addition rules that; once the sum goes over 9 in a particular place value, the sum must be divided by base (i.e. 10) and the quotient be added as a carry to the next higher place value, with the remainder regaining the original place value.

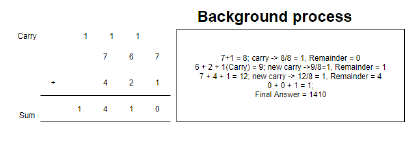
The same rules of addition apply to addition in any number system, where, if sum reaches the base or above, the sum must be divided by the base number with the quotient to be added to the next higher place value as carry and the remainder to hold the original place.

For instance;

Like adding (123)10 + (207)10 = (330)10; where 7 + 3 = 10. 10 / 10 = 1, which is added as a carry to the next higher place value addition 2 + 0 + 1 = 3 ,yielding 3 as result in final value.

Similarly, if we add (767)8 + (421)8 should follow the same rules and yield result as (1410)8.

Let us look at how this addition is taking place:

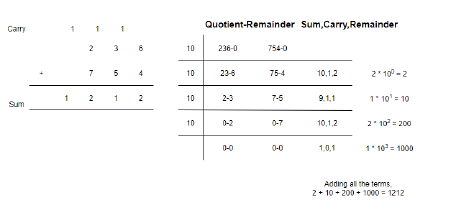


We have to understand that when we say (165)10, what we mean is 100 is occurring 5 times, 101 is occurring 6 times and 102 is occurring once.

Similarly, in (767)8, 80 is occurring 7 times, 81 is occurring 6 times and 82 is occurring 7 times again.

We have to make a carry addition once we reach the base digit of the number system in addition procedure. The additional carry is calculated as the quotient of the division of the sum by base.

Let us look at the background process in a more logically conducive and programming friendly way for better understanding. Adding 236 and 754 in base-8 with simultaneous division process side-by-side.



Now, we can see that we are extracting the rightmost digit of the numbers at a time, adding these digits up and if the sum of these digits exceeds the base-digit, we divide it by the base number. Upon division, the quotient is sent over to the next higher place value addition as carry and the remainder is written as the final sum in that particular place value. This process takes place until any of the two numbers does not become zero or the carry does not have zero in it.

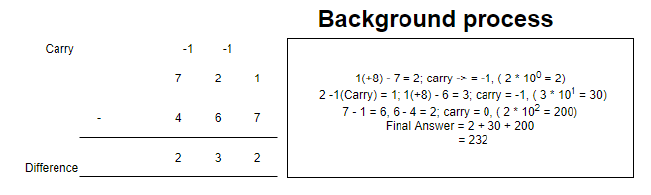
Any Base Subtraction?

We are given three numbers in this problem, a base number 'b' and two numbers belonging to that base 'n1' and 'n2' where n2 is the greater number. We have to find the difference between the two while adhering to the laws of subtractions.

The process is a bit similar to the any base addition. We extract right most digits, but here, we instead of adding them up, we subtract them. If the minuend (the number to be subtracted from) is smaller than the subtrahend (the number to be subtracted), we take a borrow from the next higher place value and reduce its value by 1.

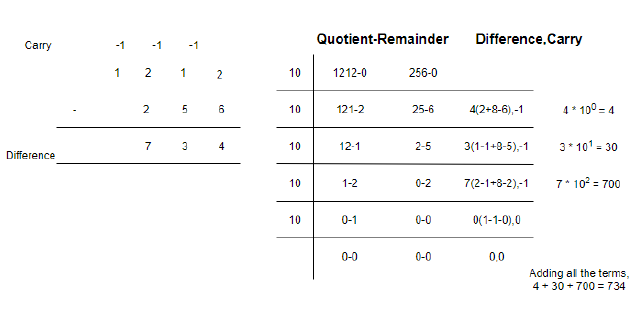
But we are not taking the number that is in the higher placer value, but a single instance of occurrence of the base number, so, we add the base number to our current minuend and then carry on with the subtraction process. We make the carry = -1, and when the subtraction of next extracted digits take place, we first settle the previous carry by adding this carry to the current minuend (had we written borrow, we would have subtracted borrow from the current minuend, but instead we are working with negative valued carries).

After that, we again check if the remaining minuend is greater than the subtrahend or not. If it is greater than the subtrahend, we follow normal subtraction and set carry = 0, else we change the state of carry = -1 and add the base number to the minuend and then subtract the subtrahend.



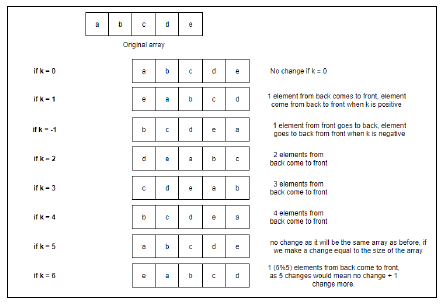
The above illustration shows how, when we subtract 7 from 1, we take a borrow 1 from 2 (or carry -1) and then subtract 7 from 1(+8) which gives us 2. When we move on to 2 as minuend, we first settle the last carry, that is, -1 in this case and then we move ahead to subtract 6 from the left 1 (after 2-1).

Let us look at another illustration which would help us to program better, with variables:



Rotate An Array?

In this problem, we need to make changes to a given array based upon the value of another given integer 'k'. Based on the positive or negative aspect of this integer, changes take place in a certain direction. The nature of the changes can be best explained using an illustration:



If k is positive, k numbers are shifted from back to the front. If k is negative, k numbers are shifted from the front to the back. If k is a multiple of the size of the array, no change is reflected in the array.

To solve this problem, we will take the help of a concept we learned in the previous problem and that is, reversing an array.

We will solve this array by dividing the given array in two parts, one part being (size of the array - k to the last index) and the second, being (index 0 to the last index - k -1). For this, we must make sure that k is a number within the size of an array. Also, as we can see in the above illustration if k exceeds the size of an array, it starts repeating the type of rotation it does. Simple observation shows that after exceeding the size, the rotation repeats in (size of array % k) fashion. For example, for k =6, the type of rotation is similar to k = 1, which is nothing but 6%5 (5 is the size of the array).

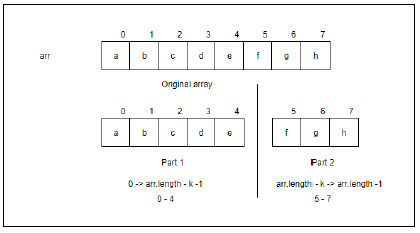
By taking the modulus, we ensure 'k' stays in the array size range. Now, if you focus on negative cases of k, you can see that as for k = -1, the type of rotation projected is similar to k = 4. There exists a very simple relation between the two. If k is negative, then if we add the

size of the array to it, it yields the same rotation [for example, k = 4 => k = (-1+5)]. So, with this relation, we can deal with the negative cases of k.

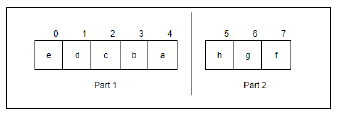
Now, let us move on to why are we dividing our original array into two parts. Well, the answer is simple. We are going to divide the array into two parts, reverse both the parts individually, and reverse the whole array together. This will yield us the required result.

Let us try on paper how this thinking is going to perform, before moving to the programming implementation as the programming implementation is pretty simple, but the thought process involved in formulation the answer relations is complex. Always remember, programming is 90% thinking and 10% typing.

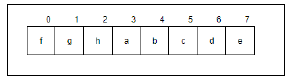
Suppose we have a character array of size 8, from a to h, in each respective index. Let k =3:



Now we reverse both the parts individually:



Now we reverse the whole array together:



After the final reversal, what we get is our whole array in the required order, the last k elements of the original order have come to the front of the array. Let part 1 be P1 and part 2 be P2. Let ( ' ) denote reverse of an array. We denote the procedure executed above mathematically like this: P1+P2

= P1'+P2'

= (P1'+P2')'

= P2''+P1''

We write it in reverse after opening the prime bracket to denote reverse in order.

And double reversal (denoted by a double prime) means no change to the original part, so what we get is: P2+P1

Rotate By 90 Degree?

First take the transpose of matrix & then reverse the columns respectively.

Print All Palindromic Substrings?

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Memory

Ex. String s1 = “hello”;

String s2 = “hello”;

String s3 = new String(“hello”);

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

Stack

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
| |  | | --- | | hello | |  | |  | |  | |  | |

Heap: There is area in heap called internpool

Interning

New

Immutability

Performance: there is performance issue of string

Diff. bet. Equal & ==

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Broken Economy?

In this problem, we need to find the ceiling and floor of the range of the given data in the given array of where it would lie.

We use binary search to solve this problem as it is an efficient method for searching in a sorted search space. The concept is very simple. When we make the comparison in binary search, every time the given data turns out to be greater than the middle element, we adjust the floor of the range to be the middle element, as the data would certainly be greater than this floor, and every time the given data is less than the middle element, we set the ceiling of the range to be the middle element, as the given data will be certainly smaller than this ceiling.

If the data exists in the search space, then the ceiling and floor would be the same.

First Index And Last Index?

In this problem, we are given an array of integers, which may contain duplicate values but sorted in order. We need to find the first and last index of their occurrence in the array, if they occur, else return -1.

We will make use of the binary search algorithm for solution but with slight modification to accommodate our purpose.

Our standard binary search algorithm dictates that

1. Initialize low = 0 and high = array.length -1

2. Input data to be searched

3. While( low<= high)

     a. Mid = (low+high)/2

     b. If(data > array[mid])

               i. Low = mid + 1

     c. else if (data < array[mid])

               i. High = mid - 1

     d. else

           i. return mid

4. return - 1

This algorithm stops as soon as it gets a match in the array for the input data. But this is useful only when only single/distinct copies of the values exist inside the array. In the problem, it is given that multiple copies may exist in the array and we need to return the starting index and ending index of the range of these duplicate values (as the array is sorted, no other values would be in between the same copies of the values).

The standard algorithm returns the mid value as index when it matches, and ends the algorithm, and -1 is returned only when the value is not found. We will declare an integer last\_index and initialize it to -1. We will run the first two conditions as is and make changes to the third condition of else, i.e., return mid. We will not end the loop on getting match, but, store this index as a potential answer in our last\_index variable. Since, we are interested in finding the last index in this case, we change our low pointer to mid + 1, and carry on our search for last index in the remaining search space excluding the current index marked as potential answer.

The algorithm for finding the last index:

1. Initialize low = 0 and high = array.length -1, last index = -1

2. Input data to be searched

3. While( low<= high)

      a. Mid = (low+high)/2

       b. If(array[mid] < data)

               i. Low = mid + 1

       c. else if(array[mid] > data)

              i. High = mid - 1

       d. else

              i. last index = mid

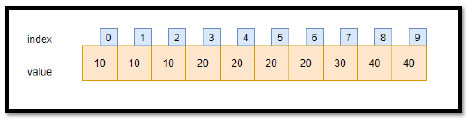
             ii. low = mid + 1

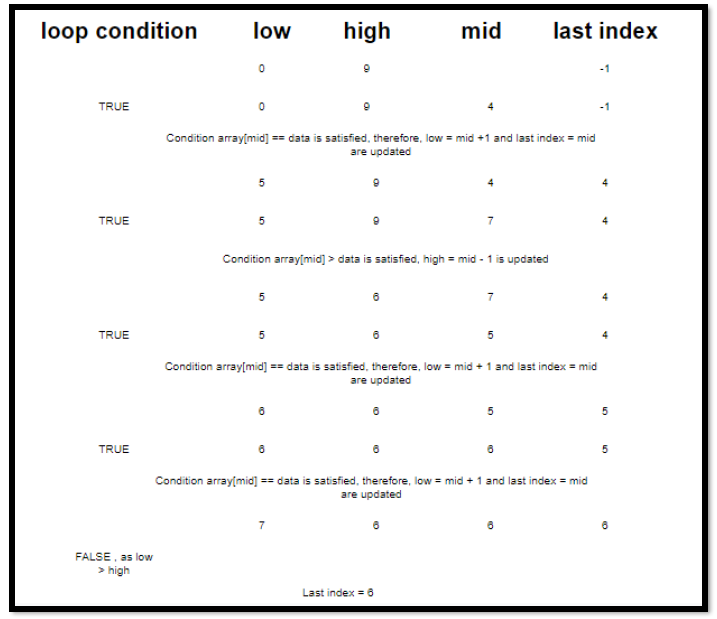
4. Return last index

Let us analyze how the above algorithm will run on a test case:

Data = 20

Array:





This is how our algorithm works for finding the last index and as we can observe, it returns the correct index of the last index of the given data = 20 before the loop breaks. 20 last appears in the index 6 of the array.

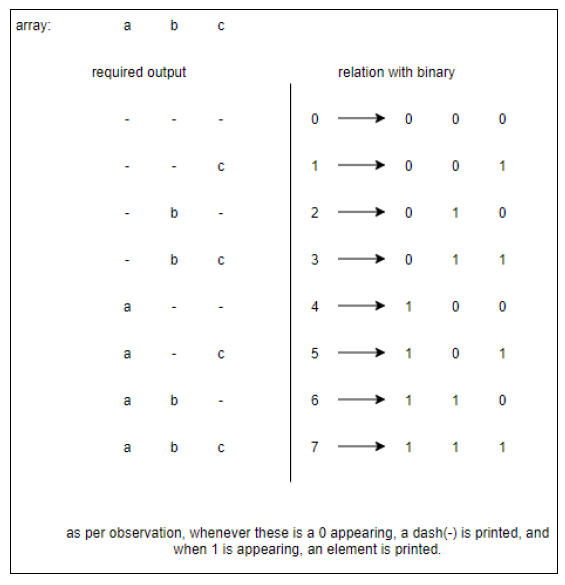
For finding the first index, we make use of the same algorithm, just in place of update last index, we update the first index = mid and then update the high pointer to mid - 1, as we have to search in the search space lower than the found first index.

Subsets Of Array?

A subset of an array is similar to a subset of a set. We print all the possible combinations of the array using each element, (including phi) which means no elements of the array. The mathematical formula for calculating subsets of a set dictates that the number of subsets of a set is equal to the 2m, where 'm' is the number of elements in the set. So, in the case of an array, it would mean the number of elements in the array or the size of the array, 2^(size of the array) will be the number of subsets.

Let us take in the case, an array of "a, b, c". Since this array has a size of 3, there would be 2^3=8 subsets.

For solving this problem, we take the help of binary numbers.



We also see that the elements are printed in an increasing order or an order where the element that is placed earlier is printed earlier in the output. We also need to handle this requirement.

array. This is done so, because if the loop was put into effect from the start of the array, the elements in the set would have been stored in a format where

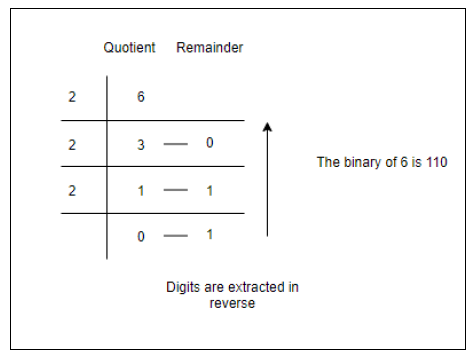
< ..middle elements.. >

would have been stored in the answer string "set" and we required the output string/pattern to be of the format

<..middle elements..>

Now, we are deciding whether the actual element is to be printed or dash (-) in place of it through the binary digits which are being extracted. A binary number is always extracting in reverse through division.

For example:



Therefore, we take the binary digits one by one and store the elements in our string in reverse order (reason for reverse traversal of the array in the inner loop).

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Memory

Ex. String s1 = “hello”;

String s2 = “hello”;

String s3 = new String(“hello”);

See that the content of all three statement is same but let’s see what will happen in memory?



When the first line runs java will check if hello already there in interpool

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

Stack

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
| |  | | --- | |  | |  | |  | |  | |  | |

Heap

Interning

New

Immutability

Performance

Equal & ==

Recursion & Backtracking > Recursion Backtracking > Knights Tour

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Questions to revise:

online-java-foundation:

Recursion and Back Tracking > Introduction to Recursion > Power-logarithmic?

Recursion and Back Tracking > Introduction to Recursion > Tower of Hanoi?

Recursion and Back Tracking > Recursion in Arrays > \*\*All Indices of Array?

Recursion and Back Tracking > Recursion with Arraylists > \*\* Get Subsequence?

Recursion and Back Tracking > Recursion with Arraylists > \*\* Get Maze Path with Jumps?

Recursion and Back Tracking > Recursion on the way up > \*\* Print Subsequence?

Recursion and Back Tracking > Recursion Backtracking > \*\* Target Sum Subsets?

Recursion and Back Tracking > Recursion Backtracking > \*\* N Queens?

Recursion and Back Tracking > Recursion Backtracking > \*\* Knights Tour?

Dynamic Programming > Time and Space Complexity > Bubble Sort [ bring the largest at last position ]

Dynamic Programming > Time and Space Complexity > Selection Sort [ keep the smallest in first position ]

Dynamic Programming > Time and Space Complexity > Merge Sort

Dynamic Programming > Time and Space Complexity > Partition An Array

Dynamic Programming > Time and Space Complexity > Quick Sort

Dynamic Programming > Time and Space Complexity > Quick Select

Linked Lists For Beginners > Design Linked List \*\* leetcode

Linked Lists For Beginners > Remove Duplicates from Sorted List \*\* leetcode

Linked Lists For Beginners > Delete Node in a Linked List \*\* leetcode

Linked Lists For Beginners > Check If Circular Linked List \*\* GFG

Linked Lists For Beginners > linked-list-cycle \*\* leetcode

Linked Lists For Beginners > Linked List Cycle II \*\*\* leetcode

Linked Lists For Beginners > remove-loop-in-linked-list \*\* GFG

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

[himansu.behera@pepcoding.com](mailto:himansu.behera@pepcoding.com)

<https://github.com/MohitBehl/pepBatches/tree/master/FJP6>

<https://docs.google.com/spreadsheets/d/1i4YhZKBSKPU9UG5lGV3M3F-GruFWeE_0ZqLfwxUvYz8/edit#gid=721852567>

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* **Ref** \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

<https://docs.google.com/spreadsheets/d/17-9mN8GQnS73unW6TevFZM3FaiRNUmre95t4WoGF3nw/edit#gid=0>