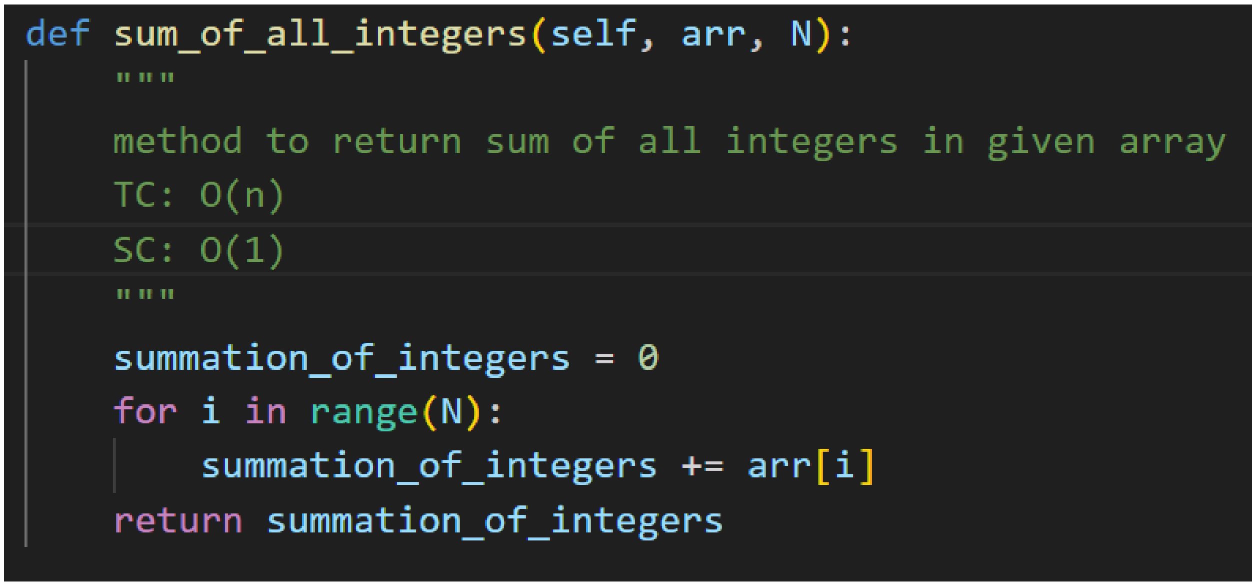
Q1. Given an array of **N** integers. Your task is to print the sum of all of the integers.

Example 1:
Input:
4
1 2 3 4
Output:
10

Example 2:
Input:
6
5 8 3 10 22 45
Output:
93

Brute force:
-> iterate and find the sum.
TC: O(n)
SC: O(1)



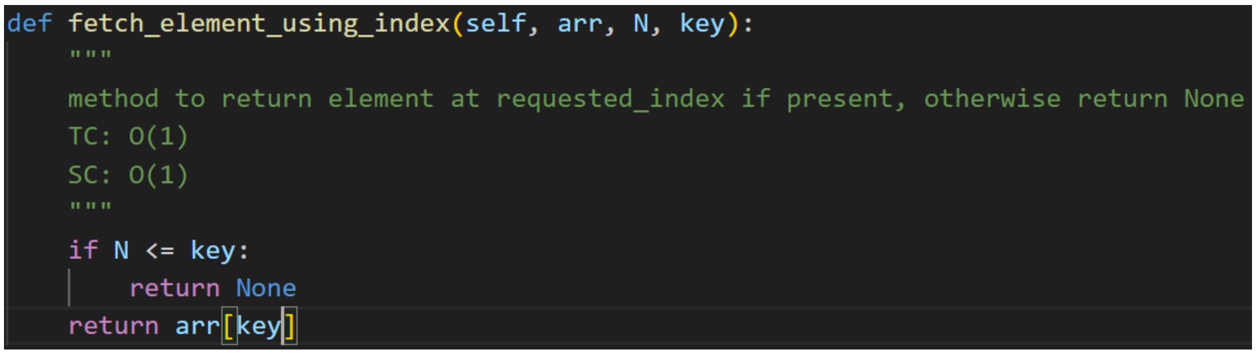
```python
def sum_of_all_integers(self, arr, N):
    """
    method to return sum of all integers in given array
    TC: O(n)
    SC: O(1)
    """
    summation_of_integers = 0
    for i in range(N):
        summation_of_integers += arr[i]
    return summation_of_integers
```

Q2. Given an array **A[]** of **N** integers and an index **Key**. Your task is to print the element present at index key in the array.

Example 1:
Input:
5 2
10 20 30 40 50
Output:
30

Example 2:
Input:
7 4
10 20 30 40 50 60 70
Output:
50

Brute force:
-> directly access the element at required index.
TC: O(1)
SC: O(1)



```python
def fetch_element_using_index(self, arr, N, key):
    """
    method to return element at requested_index if present, otherwise return None
    TC: O(1)
    SC: O(1)
    """
    if N <= key:
        return None
    return arr[key]
```

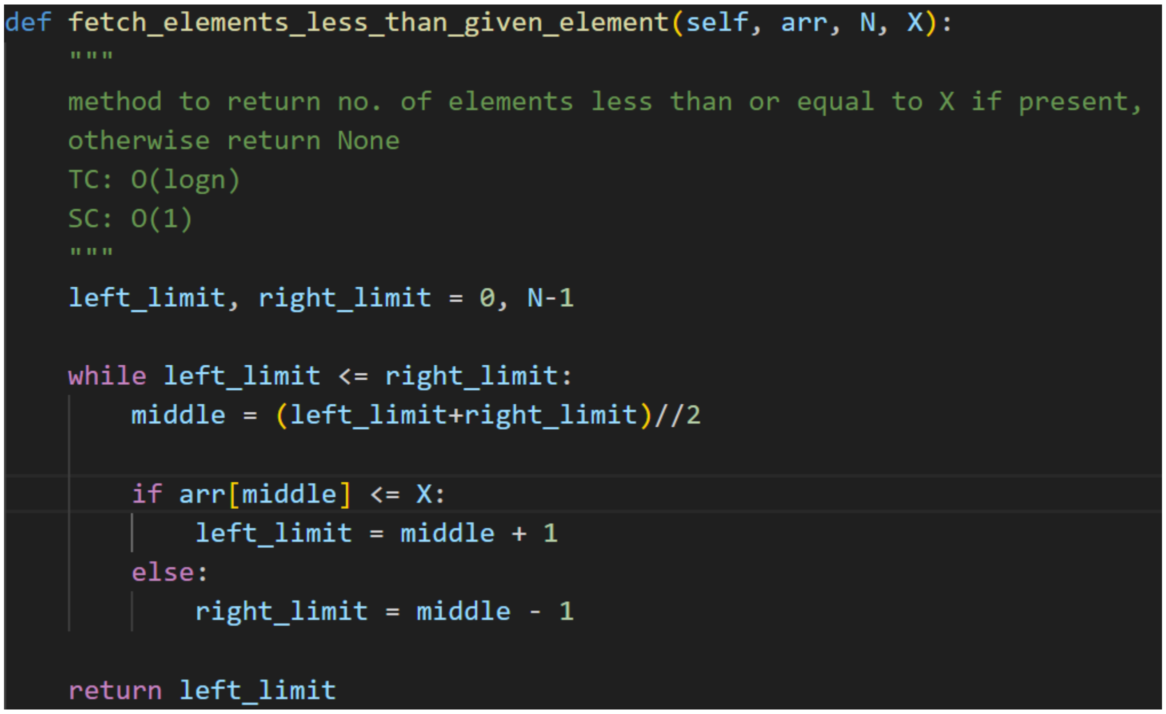
Q3. Given an sorted array **A** of size **N**. Find number of elements which are less than or equal to given element **X**.

Example 1:
Input:
N = 6
A[] = {1, 2, 4, 5, 8, 10}
X = 9
Output:
5

Example 2:
Input:
N = 7
A[] = {1, 2, 2, 2, 5, 7, 9}
X = 2
Output:
4

Brute force:
-> iterate through the array and count the no. of elements until X is found.
TC: O(n)
SC: O(1)

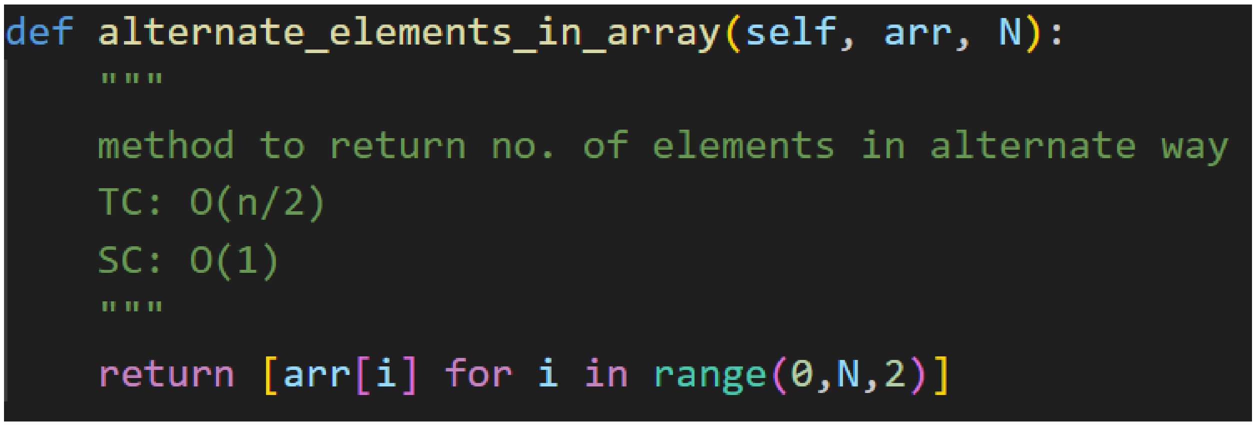
Optimal:
-> use binary search to find the index of next greater element than X.
TC: O(logn)
SC: O(1)



```python
def fetch_elements_less_than_given_element(self, arr, N, X):
    """
    method to return no. of elements less than or equal to X if present,
    otherwise return None
    TC: O(logn)
    SC: O(1)
    """
    left_limit, right_limit = 0, N-1

    while left_limit <= right_limit:
        middle = (left_limit+right_limit)//2

        if arr[middle] <= X:
            left_limit = middle + 1
        else:
            right_limit = middle - 1

    return left_limit
```

Q4. You are given an array **A** of size **N**. You need to print elements of A in alternate order (starting from index 0).
Example 1:
Input:
N = 4
A[] = {1, 2, 3, 4}
Output:
1 3
Example 2:
Input:
N = 5
A[] = {1, 2, 3, 4, 5}
Output:
1 3 5

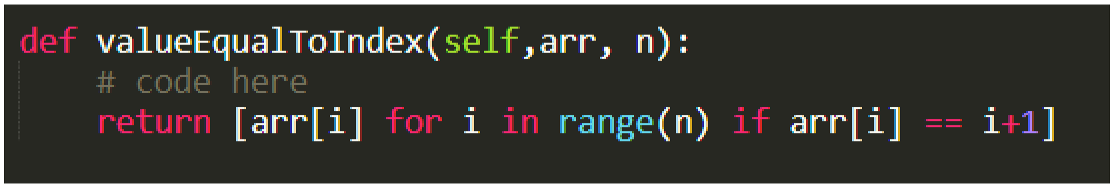
Brute force:
-> iterate through array and increment by 2.
TC: O(n/2)
SC: O(1)



```python
def alternate_elements_in_array(self, arr, N):
    """
    method to return no. of elements in alternate way
    TC: O(n/2)
    SC: O(1)
    """
    return [arr[i] for i in range(0,N,2)]
```

Q5. Given an array **Arr** of **N** positive integers. Your task is to find the elements whose value is equal to that of its index value (Consider 1-based indexing).
Example 1:
Input:
N = 5
Arr[] = {15, 2, 45, 12, 7}
Output: 2
Explanation: Only Arr[2] = 2 exists here.
Example 2:
Input:
N = 1
Arr[] = {1}
Output: 1
Explanation: Here Arr[1] = 1 exists.

Brute force:
-> iterate through array and check if element == i+1.
TC: O(n)
SC: O(1)



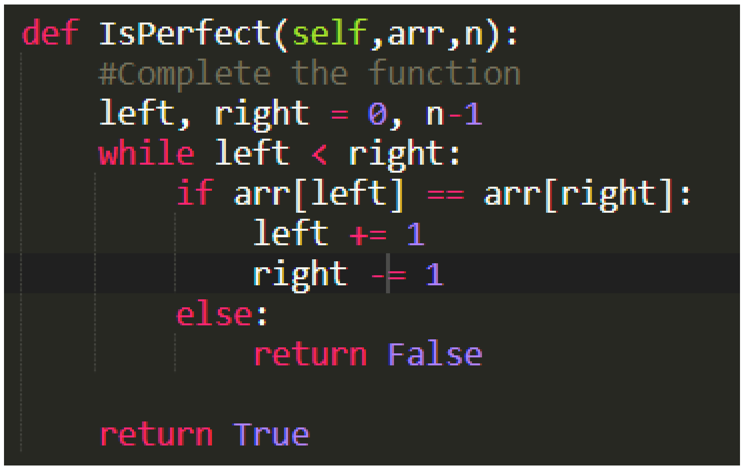
```python
def valueEqualToIndex(self,arr, n):
    # code here
    return [arr[i] for i in range(n) if arr[i] == i+1]
```

Q6. Given an array of size **N** and you have to tell whether the array is perfect or not. An array is said to be perfect if it's reverse array matches the original array. If the array is perfect then print "PERFECT" else print "NOT PERFECT".
Example 1:
Input : Arr[] = {1, 2, 3, 2, 1}
Output : PERFECT
Explanation:
Here we can see we have [1, 2, 3, 2, 1]
if we reverse it we can find [1, 2, 3, 2, 1]
which is the same as before.
So, the answer is **PERFECT**.

Example 2:
Input : Arr[] = {1, 2, 3, 4, 5}
Output : NOT PERFECT

Brute force:
-> iterate through array, and compare current element at i with element at n-1-i, if at any point equality doesn't hold, return False, otherwise True.
TC: O(n)
SC: O(1)

Optimal:
-> use two pointer logic, maintain a left(0) and right pointer(n-1), compare both till l<r, if at any point equality doesn't hold, return False, otherwise True.
TC: O(n/2)
SC: O(1)



```python
def IsPerfect(self,arr,n):
    #Complete the function
    left, right = 0, n-1
    while left < right:
        if arr[left] == arr[right]:
            left += 1
            right -= 1
        else:
            return False

    return True
```

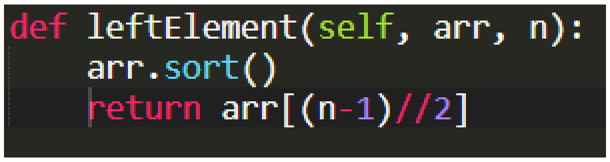
Q7. Given an array of length **N**, at each step it is reduced by 1 element. In the first step the maximum element would be removed, while in the second step minimum element of the remaining array would be removed, in the third step again the maximum and so on. Continue this till the array contains only 1 element. And find the final element remaining in the array.

Example 1:
Input:
N = 7
A[] = {7, 8, 3, 4, 2, 9, 5}

Ouput:
5
Explanation:
In first step '9' would be removed, in 2nd step '2' will be removed, in third step '8' will be removed and so on. So the last remaining element would be '5'.

Example 2:
Input:
N = 8
A[] = {8, 1, 2, 9, 4, 3, 7, 5}
Ouput:
4

-> brute force:
-> sort the array and return the middle element
TC: O(nlogn)
SC: O(1)



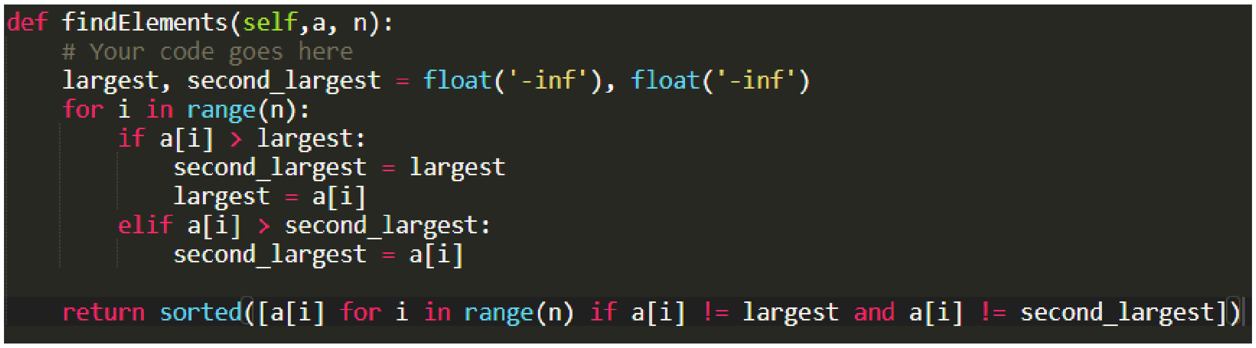
```python
def leftElement(self, arr, n):
    arr.sort()
    return arr[(n-1)//2]
```

Q8. Given an array of **N** distinct elements, the task is to find all elements in array except two greatest elements in sorted order.

Example 1:
Input :
a[] = {2, 8, 7, 1, 5}
Output :
1 2 5
Explanation :
The output three elements have two or
more greater elements.
Example 2:
Input :
a[] = {7, -2, 3, 4, 9, -1}
Output :
-2 -1 3 4

-> brute force:
-> sort the array and return the array except last two values
TC: O(nlog)
SC: O(1)

-> better:
-> traverse the array and find the maximum and second maximum element in array.
-> again traverse array and return all elements except maximum and second maximum
TC: O(n + n)
SC: O(1)

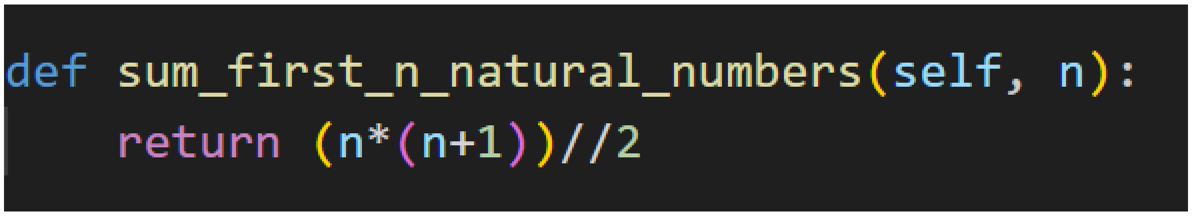


```python
def findElements(self,a, n):
    # Your code goes here
    largest, second_largest = float('-inf'), float('-inf')
    for i in range(n):
        if a[i] > largest:
            second_largest = largest
            largest = a[i]
        elif a[i] > second_largest:
            second_largest = a[i]

    return sorted([a[i] for i in range(n) if a[i] != largest and a[i] != second_largest])
```

Q9. Write a program to find the sum of the given series 1+2+3+ . . . . . .(**N** terms)
Example 1:
Input:
N = 1
Output: 1
Explanation: For n = 1, sum will be 1.
Example 2:
Input:
N = 5
Output: 15
Explanation: For n = 5, sum will be 1
+ 2 + 3 + 4 + 5 = 15.

Brute force:
-> in a loop, add all the numbers upto n
TC: O(n)
SC: O(1)

Better:
-> use formula for sum of first n natural numbers, n*(n+1)/2
TC: O(1)
SC: O(1)



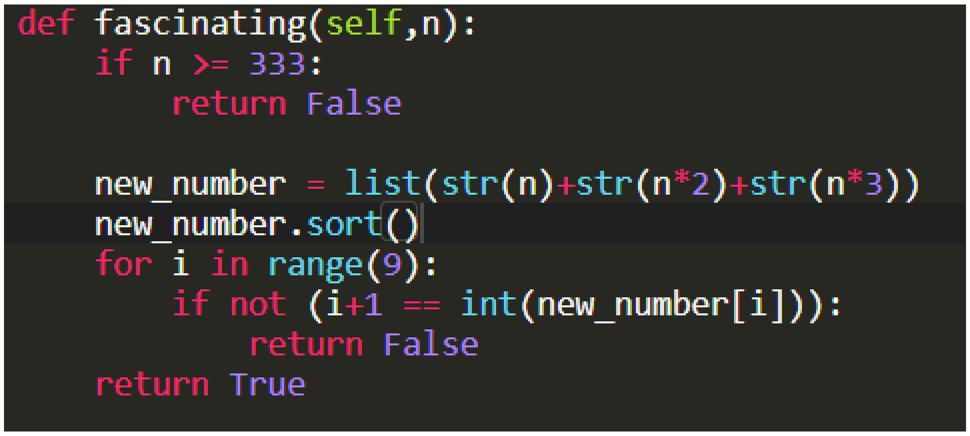
```python
def sum_first_n_natural_numbers(self, n):
    return (n*(n+1))//2
```

Q10. Given a number **N**. Your task is to check whether it is fascinating or not.
Fascinating Number: When a number(should contain 3 digits or more) is multiplied by 2 and 3 ,and when both these products are concatenated with the original number, then it results in all digits from 1 to 9 present exactly once.
Example 1:
Input:
N = 192
Output: Fascinating
Explanation: After multiplication with 2 and 3, and concatenating with original number, number will become 192384576 which contains all digits from 1 to 9.

Example 2:
Input:
N = 853
Output: Not Fascinating
Explanation: It's not a fascinating number.

Brute Force:
-> calculate new number by multiplying given number by 2 and 3 and then maintain a hash map to count the no. Of occurrences each element has and answer accordingly.
TC: O(n)
SC: O(n + n)

Better:
-> by observation, no number greater than 333 is eligible for fascinating number as it new number will cross 9 digits.
-> convert given number into string and also convert n*2 and n*3 in string and convert them into list and sort them, now run a loop if element are in order and unique.
TC: O(n) (sorting 9 elements is constant time)
SC: O(9)



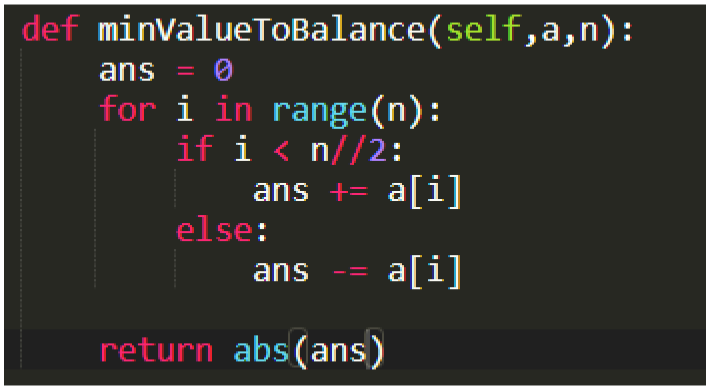
```python
def fascinating(self,n):
    if n >= 333:
        return False

    new_number = list(str(n)+str(n*2)+str(n*3))
    new_number.sort()
    for i in range(9):
        if not (i+1 == int(new_number[i])):
            return False
    return True
```

Bonus Question
Given an array of even size **N**, task is to find minimum value that can be added to an element so that array become balanced. An array is balanced if the sum of the left half of the array elements is equal to the sum of right half.

Example 1:
Input:
N = 4
arr[] = {1, 5, 3, 2}
Output: 1
Explanation:
Sum of first 2 elements is 1 + 5 = 6,
Sum of last 2 elements is 3 + 2 = 5,
To make the array balanced you can add 1.

Example 2:
Input:
N = 6
arr[] = { 1, 2, 1, 2, 1, 3 }
Output: 2
Explanation:
Sum of first 3 elements is 1 + 2 + 1 = 4,
Sum of last three elements is 2 + 1 + 3 = 6,
To make the array balanced you can add 2.

Brute force:
-> run a loop and add elements in first half of array, then start subtracting value in second half of array from the sum found in first half, return the absolute value at the end.
TC: O(n)
SC: O(1)


```
def minValueToBalance(self,a,n):
    ans = 0
    for i in range(n):
        if i < n//2:
            ans += a[i]
        else:
            ans -= a[i]

    return abs(ans)
```