HQ6

Name: Debnath Kundu (MT22026) & Manvendra Kumar Nema (MT22038)

Answer (1) :

//Given a set A, create a polynomial with coefficient A[i] , A[i]=1 if given in set A else 0 and, index represent the power and elements coefficient.return polynomial.

```
1.  def createPolynomial( A, M ):   //creating a list of ( 2* M +1 ) //elements which are
    initializing the to zero
2.          temp = [ 0 ]*( 2* M+1)
3.          for i in A  :
4.                  temp[ i + M ] += 1
5.          return temp

6.  def setUnion(A,B,M): // returns the number of unique elements in the 2 sets
    whose elements are in the range {-M, ..., M}
7.          A = createPolynomial( A, M)
8.          B = createPolynomial( B, M)
9.          counter=0
10.         for i in 0—2*M:
11.                 C[i] = A[i]+ B[i]
12.                 if c[i] !=0:
13.                         counter+=1
14.         return counter
```

The **createPolynomial** function takes **Set A** as input whose elements are in range **{-M,M}**. It creates an array whose index contains the exponential powers of x and the value at that index contains the coefficient of the respective term of x in the polynomial.

The **setUnion** function takes **Set A & B** as input whose elements are in range **{-M,M}**.

Line 7-8: are converting **set A** and **set B** into two polynomials, by using the above mentioned function **createPolynimial**.
Line 9: Initializes the counter to 0.
Line 10-13: is a for loop used to traverse over the elements of **A** and **B** of same exponential and add their coefficients.
If the sum of coefficients is greater than 0 then that means, either that element is in set A or in set B or both, which further implies that the elements has at least occurred once. Hence we increase the counter by 1.
So each value of the counter represents a unique value that exists in either of sets.

Proof of Complexity (createPolynomial):
Line 2 is used to create & initialize an array of size 2M+1, Space complexity is O(M)
Line 3-4 contains for loop which traverses over the above array of size **2M +1.**
So Time complexity is order of **O(M)**

So Time Complexity of **createPolynomial** is **O(M)**

Proof of Complexity(setUnion):
Line 7-8 requires total time complexity of 2*O(M)
Line 9 require time complexity O(1)
Line 10-13 is a for loop which traverse over array of size 2*M+1 that is O(M)

Let **T(M)** be the time complexity of **setUnion** function where M is range of number:
So, T(M) = 2*O(M) +O(1) + O(M)
$T(M) = O(M)$
Since, **M=cN** => $T(N) = O(N)$
So final Time complexity is **O(N)**
And space complexity is is **O(N)**

Answer (2) :

1. **def setToPoly(A,M):** // transform the given set into a polynomial of range -M to M.
 // create a list of (2* M +1) elements which are all initialized to zero
2. temp = [0]*(2*M+1)
3. for i in A :
4. temp[i] = 1
5. return temp;

6. **def setSum(A,B,C,M):** // Find if there exists sum zero on add any number from 3 sets return true if such pair exist else return false. Here A,B,C are given set with element with range of { -M , M }
7. A = A+M // adding a constant M to all elements of A
8. B = B+M // adding a constant M to all elements of B
9. C = C+M // adding a constant M to all elements of C
10. X = **setToPoly**(A,M)
11. Y = **setToPoly**(B,M)
12. Z = **setToPoly**(C,M)
13. temp = **FFTMULTIPLY**(X,Y) //from JeffEricson's book
14. result = **FFTMULTIPLY**(temp,Z)
15. **return (result[3*M] > 0)**

The **setToPoly** function transforms a given set into a polynomial with a maximum degree of **2*M**, where each member of the given set **A** becomes an exponential power of x and has a coefficient of 1; while the elements outside the set, but within the range of **0-2*M**, have a coefficient of 0.

Lines 7-9: The **setSum** function takes 3 sets containing elements within the **range {-M , M}** as inputs, which is difficult to work with, so these sets are transformed to the range

[0,2*M] which means that the earlier problem of selecting 3 elements from each set whose sum was equal to 0, is now transformed into selecting 3 elements from each set whose sum is equal to **3*M**.

Lines 10-12: After transformation, the sets are given to the function **setToPoly** which returns a polynomial as shown above in the definition of **setToPoly.**

Then all the three polynomials **X,Y,Z** are multiplied as shown above in **Line 13-14** using **FFTMULTIPLY** function (refer to **Jeff Erickson** book for the actual definition). It returns polynomial coefficients whose index represents the power of the polynomial.

Now since now we have to find **3*M index (**discussed above), if the coefficient is non zero, **then,** there exists a triplet of numbers whose sum is zero.
Proof of Complexity (setToPoly)- This function creates a list of size (**2M+1)** and traverses over each element present. Hence it takes constant time.
So, the space Complexity is **O(M)** & Time complexity is **O(M).**

Proof of Complexity (setSum)-

- **Line 7,8,9:** each requires traversal and transformation of the elements. In the worst case, it might need to travers **(2M +1)** elements as the range of elements is {-M,M}. Hence it takes **3*O(M) time.**
- **Line 10,11,12:** calls the **setToPoly** function which will take **3*O(M)** time.
- **Line 13,14:** The **FFTMULTIPLY** function has a time complexity of **O(M log (M))** where M is the order of size inputs. Hence, it will will take **2*O(M log (M))** time.

Let **T(M)** be the time complexity of **setSum** function given range of numbers in {-M,M}

So, **T(M) = 3*O(M) + 3*O(M) + 2*O(M log (M))**

T(M) = O(M) + O(M log (M))

Hence, $T(M) = O(M\, log\, (M))$

The solution of the recurrence is **O(M log(M))**, and **the Space Complexity is O(M)**
M=cN where N is the total number of elements so Time Complexity is O(N log(N)).