**PROB1**

**I Don’t Know**

**PROB2**

**ii)**The **PARTITION** problem takes as input a set of positive integers A = {a1 ... an} and decides if there is a way to split A into disjoint subsets A1 and A2 with equal sums.

It is a standard **NP Complete problem.**

**PROB2**: Same as PARTITION except that now you have to decide if there is a way to split A into disjoint subsets A1 and A2 such that |sum(A1) - sum(A2)| <= 525, i.e., the difference of their sum is at most 525 (and not 0, as in the original PARTITION problem).

**Aim:**

* **A:** PROB2 has a polynomial time Verification **i.e PROB2** belongs to **NP class**
* **B:** PARTITION <= PROB2

**A: PROB2 has polynomial time verification-**

Proof: Any 2 disjoint subsets of A, say X, Y. An alternative proof can also be any subset X of A, & then we calculate the other disjoint subset Y on the fly, by doing A - X.

def **verification( A**, Proof P (X, Y)  **)**

1. if | sum( X ) - SUM( Y )| <= 525:
2. return True
3. else:
4. return False

**Complexity analysis for verification:**

Assuming that set operation costs constant time, and summation of elements cost linear time in input i.e., O( n ), so total time complexity is O( n ), where n is the number of elements.

*Hence the complexity of verification is* ***O( n ) which is polynomial time verification.***

**Correctness:**

* *For any* ***YES*** *instance of* ***A****, there must be a Proof P such that* ***verification(****A, Proof P****)*** *returns* ***TRUE****.*

A **YES** instance of A can be defined as there exists 2 disjoint subsets of A, X & Y such that |sum( X ) - SUM( Y )| <= 525.

AsProof P is defined as any 2 disjoint subsets of A. Hence, there exists 2 disjoint subsets such that they are exactly the same as the above subsets X & Y.

Hence there exists a P such that **verification(**A , Proof P**)** will return **TRUE.**

* *For a* ***NO*** *instance of A,* ***verification(****A, Proof P****)*** *returns false for all proofs.*

A **NO** instance of A is defined as there does not exist 2 disjoint subsets of A, X & Y such that |sum( X ) - SUM( Y )| <= 525.

Since there is no pair of disjoint subsets of A that satisfies the condition, it's trivial to conclude that there will not be any way to split A into 2 disjoint subsets such that the difference in their sum is less than 525.

Hence, ***verification(****A, Proof P****)*** will always return FALSE, for all proofs.

Therefore, **PROB2** belongs to the **NP class.**

**B: Prove PROB2 is NP-Hard i.e PARTITION <= PROB2**

Consider the following algorithm.

def **reduce(A)**:

1. for( i = 0; i < A.length; i++ )
2. A[ i ] = A[ i ] \* 526
3. return A

**Complexity analysis Reduce:**

Line **(1)** in the reduction step takes **O(n)** time as the entire array A is traversed once.

Line **(2)** performs a constant multiplication i.e. takes **O(1)** time.

Hence, the reduction takes ***O(n)*** time i.e. it's a poly-time reduction.

**Statement of correctness lemma for the reduction:**

*Given an array* ***A,*** PARTITION *returns* ***TRUE***(if there is a way to split A into disjoint subsets A1 and A2 with equal sum.)*, iff PROB2**returns* ***TRUE*** *for the reduced array B (*there is a way to split B into disjoint subsets B1 and B2 such that |sum(B1) - sum(B2)| <= 525, i.e., the difference of their sum is at most 525) ***[B=reduce(A)]***

**Forward proof(=>):**

Say array **A** can be divided into two disjoint subsets A1 and A2, such that the sum of both disjoint sets are equal, say equals to k**.** In the reduction step, each element of A is scaled by 526. Let this reduced array be **B i.e., B=reduce(A).**

When **B** is given as input to PROB2, since B is the scaled version of A, B can also be divided into disjoint sets B1 and B2, which sums to 0.

Given, sum(A1) = sum(A2) => sum(A1) - sum(A2) = 0 => 526 \* (sum(A1) - sum(A2) ) = 0

Now, for any 2 subsets **B1 & B2 of B**, sum(B1) - sum(B2) = 526 \* ( sum(A1) - sum(A2) ) = 0 <=525.

Thus, **PROB2** will also return **TRUE**.

**Backward proof(<=):**

If y is a “TRUE” instance of PROB2, then x is a “TRUE” instance of **PARTITION*.***

***OR,*** *we can prove its contrapositive statement:*

If x is a “FALSE” instance of **PARTITION**, then y will also be a “FALSE” instance of PROB2***.***

Say array **A** cannot be partitioned into two disjoint subsets with an equal sum**, i.e. A is a FALSE** instance of **PARTITION*.***

Now in reduction, each value is scaled by 526. Let this reduced array be **B, i.e., B=reduce(A).** Let B can be reduced into two subsets B1 and B2, but since B is a scaled instance of A, B1 and B2 can not be reduced into equal disjoint subsets [ sum (B1) != sum(B2) ]

Say, sum(A1) != sum(A2) for any A1,A2.

=> (sum(A1) - sum(A2)) !=0 => |(sum(A1) - sum(A2))| = d *(d>0)*

=> 526 \* |(sum(A1) - sum(A2))| = 526\*d

=> |(sum(B1) - sum(B2))| = 526 \* d > 525 , as d>0

Hence, ***PROB2*** will also return **FALSE**.

Hence, the above reduction algorithm is a poly-time reduction from **PARTITION** to **PROB2.**

From above statements, we can conclude that PROB2 has a polynomial time verification (i.e., it belongs to NP class) and a NP-hard problem(PARTITION) can be reduced to PROB2 in polynomial time.

Hence the given problem **PROB2** is also **NP-COMPLETE**.

**Part iii)** Suppose that on a later date, someone can design a (correct) polynomial-time algorithm for PARTITION. Show that the hard problem above would then become easy.

Since PARTITION is an NP-Complete problem also we have proved that PROB2 is an NP-Complete problem. Which are reducible to each other i.e.

1. PARTITION => PROB2
2. PROB2 => PARTITION

Now since PARTITION is POLYNOMIAL time it implies that PROB2 is also reduced to POLYNOMIAL time.

I.e. if there exists a polynomial time algorithm for PARTITION, then the PROB2 problem can also be solved in polynomial time.