Please enter your name and uID below.

Name: Devin White

uID: u0775759

Collaborators, if any, and how you collaborated:

Submission notes

- Due at 11:59 pm (midnight) on Thursday, Dec 1st.
- Solutions must be typeset using one of the template files. For each problem, your answer must fit in the space provided (e.g. not spill onto the next page) *without* space-saving tricks like font/margin/line spacing changes.
- Upload a PDF version of your completed problem set to Gradescope.
- Teaching staff reserve the right to request original source/tex files during the grading process, so please retain these until an assignment has been returned.
- Please remember that for this problem set, you are allowed to collaborate in detail with your peers, as long as you cite them. However, you must write up your own solution, alone, from memory. If you do collaborate with other students in this way, you must identify the students and describe the nature of the collaboration. You are not allowed to create a group solution, and all work that you hand in must be written in your own words. Do not base your solution on any other written solution, regardless of the source.

1. (Partition in P)

   (a) The recurrence for the algorithm for partition is as follows:

   $$\text{Partition(i,m)} = \begin{cases} TRUE & m = 0 \\ FALSE & m! = 0 \text{ and } n = 0 \\ Partition(i+1, m) || Partition(i+1, m - A[i]) & otherwise \end{cases}$$

   As we can see, this recurrence is the same as subset sum, however, we need to perform some modifications so that this works appropriately with negative numbers or possible negative sums. As such, the steps of the algorithm work as so:

   i. iterate through the original whole set of integers, and sum up the total of every single integer.
   ii. if this total sum is divisible by 2(even), then we can proceed to the next step. If it isn't, then false is immediately returned
   iii. if the original total sum is positive, we simply go to step v
   iv. if the original total sum is negative, we loop through the entire array once, changing each negative value to a positive value, and each positive value to a negative value. This is done in order to preserve the original values, but make it work with the subset sum algorithm if the total is negative. We then multiply the sum by -1 so that it becomes positive. We then go to step v.
   v. if proceeding to this step, we can then use the subset-sum algorithm using SUM/2 as the argument

   The sub problems being solved by this algorithm are as follows: Whether or not we wish to add the next element or not to the current sum. So, for example, if the overall set is S = $1, 2, 3, 4, 5, 6$ and the target is 4, the first subproblems to be solved would be asking, should I add 1 to my sum or not, the next whether to add two to my sum or not, and so on and so for. In this case, the broader overall question being asked does there exist a case where a set adding up to SUM/2 can be found, as by necessity if one set adds to SUM/2, the remaining items in the set must also add to SUM/2 if sum %2 = 0? Essentially it boils down to the same questions as the basic version of subset sum: "There is a subset of X that includes x and whose sum is T." or "There is a subset of X that excludes x and whose sum is T".This algorithm can be memoized using a 2D array of size [(n+1)*2,(Sum+1)*2](i believe (n+1),(Sum+1) work just as well, but extra space is left for possible negatives to prevent leaving the array.) This is to account for the possibility of negative numbers in the array, and thus the range must be large enough to account for negatives. Because each entry Each entry A[i, m] depends on at most two other entries, both of the form SS[i + 1,·]. the array can be filled from bottom to top in the outer loop, that is, from i = 0 to n or SUM, and the inner loop being filled in an arbitrary order, but we'll say j=0 to SUM or N for convenience from bottom to top in the outer loop, and considering the elements in each row in arbitrary order in the inner loop. In other words, it is filled bottom up, with j before i. This algorithm also runs in O(n*M) time because it iterates through 2 loops, one which is reliant on size of the set, n, and one that relies on the total sum of all elements of the set, M. The other loops for changing negatives or positive can be ignored, so it runs in O(nM)

   (b) The reason that this algorithm doesn't imply that P = NP is because the polynomial factor is the size of the absolute sum of the input, rather than the input itself. In other words, the partition problem and its solution are not polynomial to n. To qualify as a true polynomial-time algorithm, the running time must be a polynomial function of the input size—the number of bits required to represent the input the values of the elements of X and the target sum may possibly be exponentially larger than the number of input bits in n.

2. (6Partition$^+$)

(a)
- A certificate for 6Partition+ is the integer elements to choose from set s. A valid YES-instance is such that all n 6 element sets are equal to one another(with duplicates allowed). For example, the Set $S = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\}$ can be partitioned into $S1 = \{10, 11, 12, 1, 2, 3\} = 39$ $S2 = \{4, 5, 6, 7, 8, 9\} = 39$ which would return true. A set can only return true if the total sum of the original set is divisible by 6.
- Check that the initial input has a sum/6(e.g, check if n 6 element multisets exist)
- if sum%6 = 0
- check that each sub-multiset has the exact same sum, if so, then it is YES (E.G) Sum the entire original set S if the total sum % 6 != 0 return no. If the sum of the original entire set sum % 6 = 0 we can then check all possible n 6 element subsets. if in any of them all of S1 = S2 = ...Sn we return true. Otherwise, return false.
- This algorithm runs in polynomial time since the sum and the amount of subsets to check are finite.

(b) This reduction will not work because it is only checking for a specific instance of Sϵ3Partition and S'ϵ6Partition+. In essence, it does not check for an arbitrary instance of 3Partition or 6Partition. More Specifically, if Sϵ3Partition, then the sum of each of the n subsets of S must necessarily be equivalent to each other. However, by constructing the new set S', the sum of each subset will be double the sum of any subset of S, meaning that S' does not necessarily satisfy the 3Partition problem's constraints. In other words the newly transformed subsets in 6partition+ are not necessarily equivalent to the original partitions in 3partition. This transformation will only work a single way. As an example, The subsets $\{0, 2, 6\}\{1, 3, 4\}$ are not equivalent to the subsets $\{0, 1, 2, 3, 4, 6\}\{0, 1, 2, 3, 4, 6\}$ in 6Partition+, there is no natural way to reduce a subset in 6Partition+ to 3Partition in a natural, arbitrary way where the sets or their sums remain equivalent, and the reduction from 3Partition to 6Partition+ is not valid.

(c) Assume there is an arbitrary algorithm which takes as input S and transforms S into a new set S', where S' = S + 3n Copies of a large number T., and Sum of 6 partition+ = Sum of 3Partition+3nT ($\rightarrow$) Because the value T is a very large number which for all intents and purposes is infinity, then we can transform any YES instance of 3-partition into a valid YES instance of 6-partition+ with 3 T's in every single new set of 6 elements. Using the original input S which creates n 3-element sets of size where if the total sum = SUM3, each n subset has a sum of SUM3/n, using our transformation, we can get a set S' which can be subdivided into an equivalent amount of n 6-element sets of size (SUM3/n)+3T(or (Sum3+3nT/n). In other words, if Sϵ3Partition, then there exists a YES where an input of S can be partitioned into 3 subsets such that all n subsets have the same sum of sum/n which we will call SUM3, in which case, we can transform S $\rightarrow$ S' by adding 3n Copies of T, along with the original S, into S', which has a total sum of 3nT+(Sum3). S' can then be divided into n 6 element subsets where each subset is the equivalent of the sum of the subsets of S, or (SUM3/n)+3T, where the equivalent subsets of both 3Partition and 6Partition+ contain the same elements besides the additional 3T in the subsets of 6Partition+. Therefore Sϵ3Partition implies S'ϵ6Partition+
($\leftarrow$) If S' ϵ6Partition+, then there exists a partition S' into 6 element subsets of equivalent sum, with a total sum of S' = SUM6. We can transform S' into S by removing 3nT from S', which will create S, with a total sum of (SUM - 3nT) in addition, this means that for any of the n 6-element subsets in S' of sum = SUM6/n, we can create an equivalent 3-element subset of sum = (SUM6/n)-3T. The sum in each subset of 3Partition will be equivalent to the sum in each subset of 6partition minus 3T. Therefore, S'ϵ6partition+ implies that Sϵ3Partition