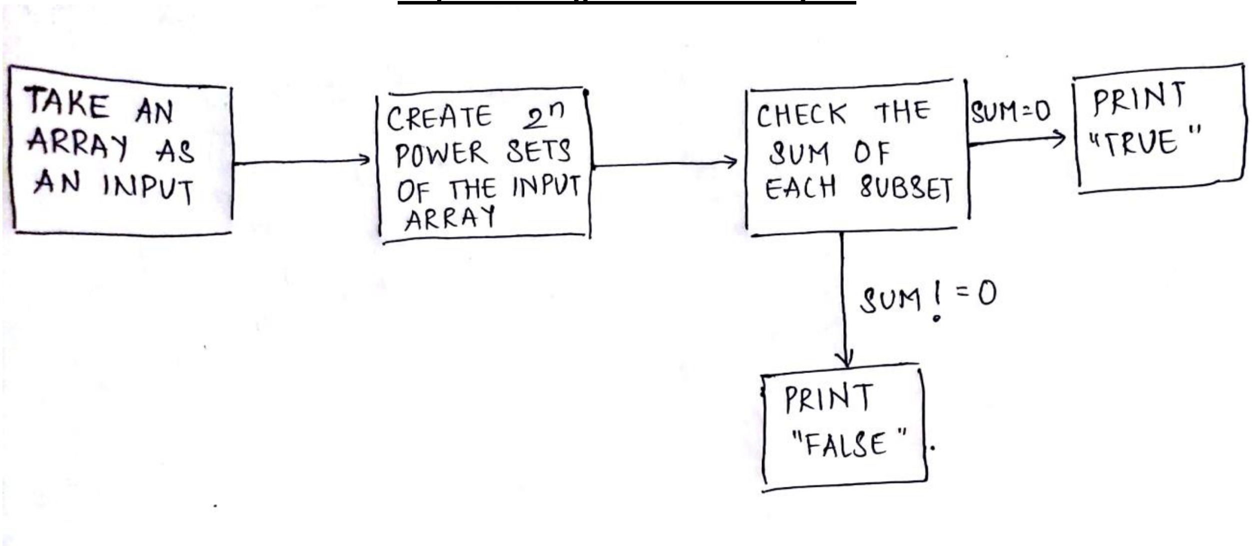
Report: Assignment 3 –Zero.java



The programs selects bunch of elements from the given set which sums up to zero. To solve this problem, we have written a function *checkZero* which accepts start and end pointers to iterate on a static array. The algorithm starts with making 2^n power sets of the input array. While these combinations are made, we store the sum of every element added to the combination in a variable called as sum. Also, we keep a track of all the elements added to combination in an array called as temp. Later, if we get a set of elements which sum up to zero, temp array is used to show the user which specific elements were used to make sum zero.

Design changes during implementation

The current algorithm is of O(2^n). We tried to reduce the complexity to O(2^n/2) by making few modifications to the current algorithm, which were:

1) Split the array in mid-way. Say now we have array A and array B.
2) Find out all the power set of all the possible sums for each of the element in the arrays(Each will run for O(2^n/2). Assume we have now 2 new generated arrays, A_Pow and B_Pow.
3) Moreover, as you make these power set, maintain 2 new arrays which will store the combination of the elements used form the set A/B for making the sum. Say its named as A_Comb and B_Comb. These will used to show which elements were used to make the sum 0.
4) Now sort A_Pow and B_Pow using Bubble sort and while swapping a element from A_Pow or B_Pow, swap the respective combination of elements from A_Comb or B_Comb.Here, Bubble sort will run for O(n^2).
5) Given the sort list, now we need to check if the element in A_Pow and element in the B_Pow
   Sums up to 0 in time O(2^n/2).
6) To do that, scan the A_Pow in decreasing order and B_Pow in increasing order.
7) Whenever the sum of the current element in the A_Pow and the current element in the B_Pow is more than *0*, the algorithm moves to the next element in the A_Pow.

8) If it is less than *0*, the algorithm moves to the next element in the B_Pow. If two elements with sum *0* are found, then stops.

The only good thing about this algorithm is that it’s O(2^n/2) but the overhead of finding 2 power set’s, then maintaining an array of combination of those sums, then sorting both the arrays with bubble sort and then iterating the final sorted arrays was too huge. And hence we thought of going with our current approach.