1049. Last Stone Weight II <medium>

int lastStoneWeightII(vector<int>& stones) {

/\* dynamic programming - bottom up

Objective: We need to partition the array into two subsets such that the difference of their sums is minimised.

Of the given denominations of stones, we need to see the maximum target achievable. (0 < target <= stone\_sum/2)

dp[i][j] tells whether target j is achievable or not with atmost i stones.

So, find the max j that can be achieved. let it be s, return abs(2\*s - stone\_sum).

\*/

int n = stones.size();

int stone\_sum = 0;

for(int i = 0; i < n; i++)

stone\_sum += stones[i];

int target = stone\_sum % 2 == 0 ? stone\_sum/2 : (stone\_sum +1)/2;

vector<vector<int>> dp(n+1, vector<int>(target+1, false));

for(int i = 0; i <= n; i++)

dp[i][0] = true;

int s = 0;

for(int i = 1; i <= n; i++)

{

for(int j = 1; j <= target; j++)

{

if(j >= stones[i-1])

dp[i][j] = dp[i-1][j] | dp[i-1][j-stones[i-1]];

else

dp[i][j] = dp[i-1][j];

if(dp[i][j] == true)

s = max(s,j);

}

}

return abs(2\*s - stone\_sum);

}

class Solution {

public:

int lastStoneWeightII(vector<int>& stones) {

const int sum = accumulate(begin(stones), end(stones), 0);

vector<bool> dp(sum + 1);

dp[0] = true;

int s = 0;

for (int stone : stones)

for (int w = sum / 2; w > 0; --w) {

if (w >= stone)

dp[w] = dp[w] || dp[w - stone];

if (dp[w])

s = max(s, w);

}

return sum - 2 \* s;

}

};