Question: https://leetcode.com/problems/delete-and-earn/

* Every possibility needs to be considered, i.e. a binary decision needs to be made for each number of whether that number should be included or deleted. This is a hint that this is a typical binary decision tree problem of yes or no, where for each num a choice has to be made, which will influence the further computation and thus the final result
* For every nums[i] = x included, x - 1 and x + 1 must be excluded (if they exist in nums) and points earned for such x = x (the number itself)
* If x is included once, then it can be included for all its repetitions r, so total points from including x = x + x + ... (r times) = x \* r. This is a hint for creating a num-to-frequency map
* For each num, assume that the array ends at that num, so you only need to consider the x - 1 neighbor. Since the actual solution will iterate through the full array, x + 1 will be automatically taken care of as x + 1 in next iteration will be the new x and will handle the case. This technique is pretty typical in DP problems so if you have problem grasping it, I'd recommend solving [LC 198. House Robber](https://leetcode.com/problems/house-robber/discuss/336340/Java-or-Time-and-Memory-beats-100-or-4-methods) or similar problems before this one, as this one is just a slight modification of that classic problem
* If x is included, then x - 1 must be excluded, so earning = earning by excluding x - 1 + earning by including x
* If x is excluded, then x - 1 can be included or excluded, depending on which one give greater earining, so earning = max(earning by including x - 1, earning by including x + 1)
* Result will be max of including or excluding the nums[last]
* To simplify the algorithm, consider all ints in the range [min, max] so that for each x in nums[i] there is an x - 1 with frequency = 0.
* Every possibility needs to be considered, i.e. a binary decision needs to be made for each number of whether that number should be included or deleted. This is a hint that this is a typical binary decision tree problem of yes or no, where for each num a choice has to be made, which will influence the further computation and thus the final result
* For every nums[i] = x included, x - 1 and x + 1 must be excluded (if they exist in nums) and points earned for such x = x (the number itself)
* If x is included once, then it can be included for all its repetitions r, so total points from including x = x + x + ... (r times) = x \* r. This is a hint for creating a num-to-frequency map
* For each num, assume that the array ends at that num, so you only need to consider the x - 1 neighbor. Since the actual solution will iterate through the full array, x + 1 will be automatically taken care of as x + 1 in next iteration will be the new x and will handle the case. This technique is pretty typical in DP problems so if you have problem grasping it, I'd recommend solving [LC 198. House Robber](https://leetcode.com/problems/house-robber/discuss/336340/Java-or-Time-and-Memory-beats-100-or-4-methods) or similar problems before this one, as this one is just a slight modification of that classic problem
* If x is included, then x - 1 must be excluded, so earning = earning by excluding x - 1 + earning by including x
* If x is excluded, then x - 1 can be included or excluded, depending on which one give greater earining, so earning = max(earning by including x - 1, earning by including x + 1)
* Result will be max of including or excluding the nums[last]
* To simplify the algorithm, consider all ints in the range [min, max] so that for each x in nums[i] there is an x - 1 with frequency = 0.

Code:  
class Solution {

public int deleteAndEarn(int[] nums) {

HashMap<Integer, Integer> mem = new HashMap<Integer, Integer>();

int min = Integer.MAX\_VALUE;

int max = Integer.MIN\_VALUE;

for (int num : nums) {

mem.compute(num, (k, v) -> v == null ? 1 : ++v);

min = Math.min(min, num);

max = Math.max(max, num);

}

int prevIncEarn = 0;

int prevExcEarn = 0;

for (int i = min; i <= max; i++) {

int incEarn = prevExcEarn + i \* mem.getOrDefault(i, 0);

int excEarn = Math.max(prevIncEarn, prevExcEarn);

prevIncEarn = incEarn;

prevExcEarn = excEarn;

}

return Math.max(prevIncEarn, prevExcEarn);

}

}

Github Link :<https://lnkd.in/ecwtJeaz>