Programming Assignment 3

Tristan Fry

April 2023

1 The Algorithm

The algorithm is a dynamic programming algorithm using top down / memoization to solve this problem. We make use of a hash-table or a python dictionary in order to get and track the occurrences of each number. We use another hash-table or python dictionary which we use to store our computed values. We also use this memoization dictionary to check to see if our max sum up to our current value has already been computed in which case we simply return the value, which we preform recursively. We also keep in mind the constraints of the problem in which the adjacent numbers to our current number cannot be equal, in which we decide if we want to take or skip, where we take if the adjacent number not equal and skip if the adjacent numbers are equal. As our final answer we call the recursive function one final time at the end in which it will give use the max possible sum, which will be our final answer.

2 Pseudo code

```
count = countOccurrences(nums)
nums = SortedAndRemoveDuplicates(nums)
store = {∅}
function DP(i)
    if i < 0 then
        return 0
    end if
    if i is in store then
        return store[i]
    else
        skip = DP(i - 1)
        take = count[nums[i]] * nums[i]
        if i > 0 and nums[i - 1] == nums[i] - 1 then take += DP(i - 2)
        else
            take += DP(i - 1)
        end if
    end if
```

```
        store[i] = maximum(skip,take)
        return store[i]
    end function
    return DP(length(nums) - 1)
```

3 Run time

The run time is $O(nlogn)$, we are sorting before which takes O(log n). Then finding and preforming all the computations takes $O(n)$ time, therefore we get $O(nlogn)$

4 Correctness

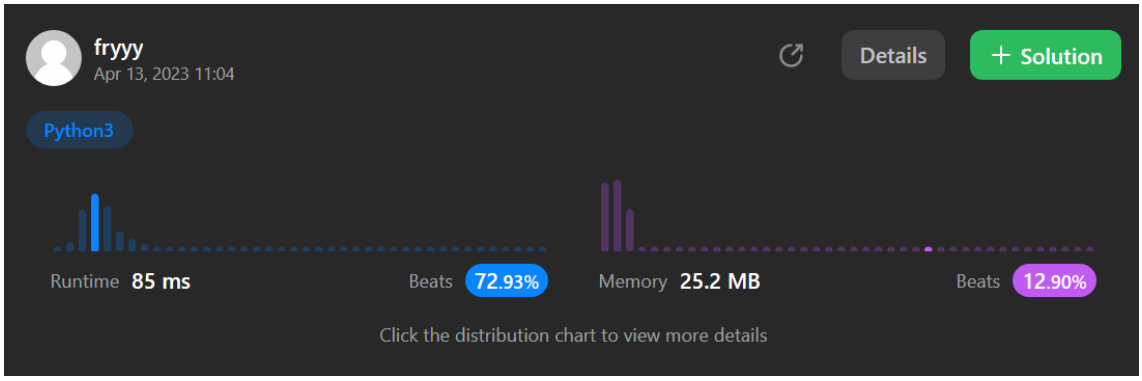
We can use a proof of induction to prove the correctness.

Base case: i $<$ 0, since i is less than 0, we cannot take any element as the max earn is 0. This is because 0 is the highest earn we can get when we don't select an element. Meaning we have iterated through the entire input array.

Inductive Step: Lets assume that the algorithm is correct for inputs up to i - 1. Thus we must show that it works for i. The algorithm gets the max earn value, we do this by either skipping or taking the element, and then we recursively get the max earn for the remaining elements. The correct choice for skipping or taking the current element depends on the adjacent (i - 1) element was taken or not. Because, if it was taken then we cannot take the current element, because they are adjacent. If it was not taken then we can take it, and then recursively compute the max earn for the rest of the elements.

The algorithm takes into account all of the choices possible, and will compute the max earn for each choice. We also know that the algorithm works for elements smaller than the current input, and because it works for all choices, it works for this as well. Thus we have proved the correctness by mathematical induction.

5 Leet Code Submission



6 Code

```
class Solution:
def deleteAndEarn(self, nums: List[int]) -> int:
    # Count the occurrences of each numbern and store in a dict /
        hash table
    count = Counter(nums)

    # remove duplciates than sort
    nums = sorted(set(nums))

    #empty dictionary for storing cmputed values
    store = {}

    # Define the recursive helper function
    def dp(i):
        if i < 0:
            # base case
            return 0
        if i in store:
            # reurn val if already computed
            return store[i]
        else:
            # assigning the values to skip and take respectively
            skip = dp(i-1)
            take = count[nums[i]] * nums[i]

            #checks adjacent numbers to see if they are equal to the
                current number
            #if they are we cannot take as the problem states
            if i > 0 and nums[i-1] == nums[i] - 1:take += dp(i-2)
            else:take += dp(i-1)
```

```
            store[i] = max(skip, take)
            return store[i]

        # Call the recursive function on last time to get our final
            answer
        return dp(len(nums)-1)
```