**DP-3**

**Delete & Earn:**

class Solution {

public int deleteAndEarn(int[] nums) {

if(nums == null || nums.length == 0)

return 0;

int max = Integer.MIN\_VALUE;

//update num

for(int num:nums){

if(max < num)

max = num;

}

//max+1

int[] arr = new int[max+1];

for(int num: nums){

arr[num]+= num;

}

int skip = 0, choose = arr[0];

for(int i = 0; i< arr.length; i++){

int prevSkip = skip;

skip = Math.max(skip,choose);

choose = prevSkip+ arr[i];

}

return Math.max(skip,choose);

}

}

The time complexity is O(N)

The space complexity is O(N).

Minimum Falling path:

class Solution {

public int minFallingPathSum(int[][] matrix) {

if(matrix == null || matrix.length == 0)

return 0;

if(matrix.length == 1)

return matrix[0][0];

int m = matrix.length, n = matrix[0].length;

int[][] result = new int[m][n];

for (int i = 0; i < m; i++) {

result[0][i] = matrix[0][i];

}

for (int j = 1; j < m; j++) {

for (int i = 0; i < m; i++) {

if (i == 0) {

result[j][i] = Math.min(

result[j - 1][i], result[j - 1][i + 1]) + matrix[j][i];

} else if (i == n - 1) {

result[j][i] = Math.min(result[j - 1][i], result[j - 1][i -1]) + matrix[j][i];

} else {

result[j][i] = matrix[j][i] + Math.min(

Math.min(result[j - 1][i], result[j - 1][i + 1]),

result[j - 1][i - 1]);

}

}

}

int res = result[n - 1][0];

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

if (result[n - 1][i] < res) {

res = result[n - 1][i];

}

}

return res;

}

}

The time complexity is O(N2).