**Question :**

You are given an array of size “n” which has 3 values , easy[i] . medium[i] , hard[i]

In the day “i” , you can perform any task .

Find the maximum points you will get after N days.

**Condition :**

* You can do easy task any time.
* You can only do medium task if you did easy task on the previous day and medium task on the day before yesterday.
* You can do hard task only if yu did easy task on day before yesterday .
* Conditions are applicable only to i > 3 (1 - based indexing)

**Observation :**

Easy task on ith day = easy[i] + max (easy , medium , hard) on previous day

Medium on ith day = medium[i] + easy[i - 1] + medium [i - 2]

Hard task on ith day = hard[i] + ? + easy[i - 2]

The question says that , it is i-2 only be easy but what about i - 1 ?

It can be

* Easy task
* Medium task
* Hard Task

If it is easy ,

* hard[i] + easy[i - 1] + easy[i - 2]

If it is medium ,

* If i - 1 is medium , yesterday (i - 2) should be easy && day before yesterday should be medium (i - 3)
* hard[i] + medium[i - 1] + easy[i - 2] + medium[i - 3]

If it is hard ,

* If i - 1 is hard , day before yesterday (i -3) should be easy
* hard[i] + hard[i - 1] + easy[i - 2] + easy[i - 3]

For hard task , we need to find the maximum of it .

**Base Cases :**

dp\_easy[1] = easy[1];

dp\_medium[1] = medium[1];

dp\_hard[1] = hard[1];

dp\_easy[2] = easy[2] + Math.max(Math.max(easy[1], medium[1]), hard[1]);

dp\_medium[2] = medium[2] + Math.max(Math.max(easy[1], medium[1]), hard[1]);

dp\_hard[2] = hard[2] + Math.max(Math.max(easy[1], medium[1]), hard[1]);

dp\_easy[3] = easy[3] + Math.max(Math.max(dp\_easy[2], dp\_medium[2]), dp\_hard[2]);

dp\_medium[3] = medium[3] + easy[2] + medium[1];

dp\_hard[3] = hard[3] + easy[1] + Math.max(Math.max(easy[2], medium[2]), hard[2]);

**Code :**

class Solution {

public int maximumPoints(int [] easy , int [] medium , int [] hard) {

int[] dp\_easy = new int[N + 1];

int[] dp\_medium = new int[N + 1];

int[] dp\_hard = new int[N + 1];

// starting first few index - No rule.

dp\_easy[1] = easy[1];

dp\_medium[1] = medium[1];

dp\_hard[1] = hard[1];

dp\_easy[2] = easy[2] + Math.max(Math.max(easy[1], medium[1]), hard[1]);

dp\_medium[2] = medium[2] + Math.max(Math.max(easy[1], medium[1]), hard[1]);

dp\_hard[2] = hard[2] + Math.max(Math.max(easy[1], medium[1]), hard[1]);

dp\_easy[3] = easy[3] + Math.max(Math.max(dp\_easy[2], dp\_medium[2]), dp\_hard[2]);

dp\_medium[3] = medium[3] + easy[2] + medium[1];

dp\_hard[3] = hard[3] + easy[1] + Math.max(Math.max(easy[2], medium[2]), hard[2]);

for(int i = 3 ; i < n ; i++){

dp\_easy[i] = easy[i] + Math.max(Math.max(dp\_easy[i - 1], dp\_medium[i - 1]), dp\_hard[i - 1]);

dp\_medium[i] = medium[i] + easy[i - 1] + dp\_medium[i - 2];

dp\_hard[i] = hard[i] + easy[i - 1] + dp\_easy[i - 2];

dp\_hard[i] = Math.max(dp\_hard[i], hard[i] + medium[i - 1] + easy[i - 2] + dp\_medium[i - 3]);

dp\_hard[i] = Math.max(dp\_hard[i], hard[i] + hard[i - 1] + easy[i - 2] + dp\_easy[i - 3]);

}

return Math.max(Math.max(dp\_easy[N], dp\_medium[N]), dp\_hard[N]);

}

}