C - Vacation

Problem Summary

Taro ke paas **N** din ki summer vacation hai. Har din woh teen kaam me se ek karega:

- **A**: swim (points = a_i)
- **B**: catch bugs (points = b_i)
- **C**: homework (points = c_i)

Rule: woh **do consecutive din same activity nahi kar sakta**. Hume maximum total happiness score nikalna hai.

Input / Output

Input format

```
N
a1 b1 c1
a2 b2 c2
...
aN bN cN
```

Output

```
Maximum total points (single integer)
```

Constraints

- 1 ≤ N ≤ 10^5
- 1 ≤ a_i, b_i, c_i ≤ 10^4

- All integers
- Time limit: 2 seconds, Memory limit: 1024 MiB

Large N means solution must be O(N) time and O(N) or O(1) extra memory (but O(N) is fine here).

Key idea (Dynamic Programming)

Define `dp[i][j]` = maximum total happiness up to day `i` if on day `i` we choose activity `j`.

Let activities index: 0 = A, 1 = B, 2 = C.

Transition (for i ≥ 1):

```
dp[i][0] = a_i + max(dp[i-1][1], dp[i-1][2])
dp[i][1] = b_i + max(dp[i-1][0], dp[i-1][2])
dp[i][2] = c_i + max(dp[i-1][0], dp[i-1][1])
```

Base (i = 0):

```
dp[0][0] = a_1
dp[0][1] = b_1
dp[0][2] = c_1
```

Answer = `max(dp[N-1][0], dp[N-1][1], dp[N-1][2])`.

Why it works: on day i we pick one activity and we can add the best obtainable total from previous day that used a different activity.

Complexity

- **Time:** O(N) — each day we do O(1) work (3 transitions).
- **Space:** O(N*3) if we store full table. Can be reduced to O(1) by keeping only previous day's 3 values.

Given constraints this is perfectly fine.

Walkthrough — Sample 1

Input:

```
3
10 40 70
20 50 80
30 60 90
```

Manual DP table:

- Day 1 (i=0): dp = [10, 40, 70]
- Day 2 (i=1):
  - dp[1][0] = 20 + max(40,70) = 20 + 70 = 90
  - dp[1][1] = 50 + max(10,70) = 50 + 70 = 120
  - dp[1][2] = 80 + max(10,40) = 80 + 40 = 120
- Day 3 (i=2):
  - dp[2][0] = 30 + max(120,120) = 150
  - dp[2][1] = 60 + max(90,120) = 180
  - dp[2][2] = 90 + max(90,120) = 210

Answer = max(150,180,210) = **210** (sequence C, B, C).

Edge cases & tips

- **N = 1:** simply max(a1, b1, c1).
- **All equal values:** algorithm still works.
- **Large values:** totals can be up to N * 10^4 → for N = 10^5 this is 10^9, fits in 32-bit signed int but to be safe use 64-bit (`long long`) if you want headroom.
- **Input reading:** ensure you read exactly `3*N` integers after N; otherwise the program will block waiting for input in interactive/run environments.

- **Memory optimization:** if memory tight, use three variables `prev0, prev1, prev2` and update to `cur0, cur1, cur2` per day.

Mapping to your submitted code

- Your submitted code implements the DP exactly as explained: it reads day 1 into `dp[0]`, then loops `i=1..N-1` reading day i and computing transitions.
- Final output prints `max(dp[N-1][0], dp[N-1][1], dp[N-1][2])` — correct.
- Only practical improvements optional: use `long long` for dp (safer), add input validation if desired, or switch to 3 variable rolling-array to save memory.

Pseudocode (compact)

```
read N
read a1,b1,c1
prev = [a1,b1,c1]
for i = 2 to N:
  read a,b,c
  cur0 = a + max(prev[1], prev[2])
  cur1 = b + max(prev[0], prev[2])
  cur2 = c + max(prev[0], prev[1])
  prev = [cur0, cur1, cur2]
print max(prev)
```

Quick checklist before submitting

- Make sure standard input contains N and exactly 3*N numbers.
- No extra prompts or debug prints.
- Check 64-bit if using extreme testcases.
