HACKWITHINFY

SAMPLE QUESTIONS

Infosys®
Navigate your next

1. EASY: FOOD STAMPS

You want to buy food from a store. You have a scoring system that uses a unit called taste points
.
Each time you buy a type of food, you can measure its tastiness by the number of taste points you get from that food.

You have N types of food. You can buy any type any number of times, as long as the total number of meals does not exceed M.

However, you don't want to grow tired of a food if you buy it too often. Therefore, you will get v[i] – d[i] × (ti – 1) taste points when you buy the i-th type of food for the ti-th time.
Find the maximum number of taste points you can achieve.

Input Format

- The first line contains an integer, n, denoting the number of types of food you can buy.
- The next line contains an integer, m, denoting the maximum number of meals you can buy.
- Each line i of the n subsequent lines (where 0 ≤ i < n) contains an integer describing v[i].
- Each line i of the n subsequent lines (where 0 ≤ i < n) contains an integer describing d[i].

Constraints

1 <= n <= 10^5
1 <= m <= 10^9
1 <= v[i] <= 10^9
1 <= d[i] <= 10^9

Sample Test Cases

Case 1

Input:
1
1
5
2

Output:
5

Explanation:
You can only buy the first food once and get 5 taste points.

Case 2

Input:
2
2
5
7
2
4

Output:
12

Explanation:
You can buy 1 meal of the first type and 1 meal of the second type of food. The answer therefore will be 5+7 = 12

Case 3

Input:
3
5
5
7
9
2
4
6

Output:
27

Explanation:
you can buy the 2 meals of the first type and 2 meals of the second type and 1 meal of the third type and get 27 taste points as follows:
first type: 5 + (5 - 2) * 1 = 8
second type: 7 + (7 - 4) * 1 = 10
third type: 9 = 9
answer: 8 + 10 + 9 = 27

2. MSS WITH SWAPS

Given an array **a** of length n and an integer **k**. You must perform the following operation exactly k times: choose two indices **i, j** and swap**(ai, aj).**

Find the **maximum possible MSS** (maximum subarray sum) after performing the above operation exactly k times.

Note:
Swapping the same pair again is allowed but useless (a double-swap cancels out). Therefore, performing exactly k swaps is equivalent to at most k useful swaps.

Input Format

- The first line contains an integer, n, denoting the size of array
- The next line contains an integer, k, denoting the number of swaps.
- Each line i of the n subsequent lines (where 0 ≤ i < n) contains an integer describing a[i].

Constraints

2 <= n <= 500

0 <= k <= n

-1000 <= a[i] <= 1000

Sample Test Cases

Case 1

Input:
3
1
1
-5

Output:
3

Explanation:
By swapping 1 and -5, we get a maximum subarray sum equal to 1 + 2 = 3.

Case 2

Input:
3
0
5
-1
5

Output:
2

Explanation:
The maximum subarray sum is the sum of [2] which is equal to 2.

Case 3

Input:
3
0
1
-5
2

Output:
2

Explanation:
The maximum subarray sum is the sum of [2] which is equal to 2.

2. HARD : LOCK & PARITY

You are given **N locks** in a row (1-indexed). Each lock **i** has a value **L[i]**. There is also **one key under each lock**, and key **j** has value **L[j]**.

You may assign **some** keys to **some** locks under the following rules:

1. You may assign key **j** to lock **i** only if: **j < i**. (Each key can only be used on a lock to its right.)

2. Assignments where the key and lock have the **same value** are forbidden: **L[j] ≠ L[i]** (So the effective value is never zero.)

3. Assigning key **j** to lock **i** gives: **E = |L[j] – L[i]|**

4. Each **lock** can be assigned **at most once**, and each **key** can be used **at most once.**

5. Let **even** be the number of assignments with even effective value, and **odd** be the number of assignments with odd effective value. A set of assignments is valid only if: **even ≥ odd**. This condition applies to the final chosen set.

6. You must perform **at least one assignment**.

Find the **minimum possible sum** of effective values over all valid assignment sets. If no valid set exists, output -1 .

Notes:

- Arrays are **1-indexed.**
- Effective value can never be zero due to the rule L[j] ≠ L[i] .
- Parity is evaluated **after the entire assignment set** is chosen

Input Format

- The first line contains an integer, N, denoting the number of locks.
- Each line i of the N subsequent lines (where 1 ≤ i <= N) contains an integer describing L[i].

Constraints

1 <= **N** <= 200
1 <= **L[i]** <= 10^5

Sample Test Cases

Case 1

Input:
6
41
54
15
4
54
4

Output:
26

Explanation:
1. Build allowed assignments (j < i and L[j] ≠ L[i]) and their costs
List of candidate pairs (sorted increasing by cost):
(3 → 4): |15 – 4| = 11 — odd
(3 → 6): |15 – 4| = 11 — odd
(1 → 2): |41 – 54| = 13 — odd
(1 → 5): |41 – 54| = 13 — odd
(1 → 3): |41 – 15| = 26 — even
(1 → 4): |41 – 4| = 37 — odd
(1 → 6): |41 – 4| = 37 — odd
(2 → 3): |54 – 15| = 39 — odd
(3 → 5): |15 – 54| = 39 — odd
2 → 4): |54 – 4| = 50 — even
(2 → 6): |54 – 4| = 50 — even
(4 → 5): |4 – 54| = 50 — even
(5 → 6): |54 – 4| = 50 — even

(Only the smallest several are shown above — full set was enumerated in the solver.)

2. Check cheapest possibilities
The absolute cheapest edges are the two 11-cost edges (both odd).
Any selection consisting of a single odd (e.g. cost = 11) is invalid because even = 0 < odd = 1.
Two odd edges (11 + 11 = 22) would have odd = 2, even = 0 → still invalid.
To satisfy parity, we need at least one even assignment in the final set (or an equal number of even
and odd with evens ≥ odds).
The smallest even-cost edge is 26 (1 → 3). Any cheaper combination that satisfies parity would have to sum to < 26 and include at least one even — but there is no even with cost < 26 (next evens are 50). Thus no valid set has total **< 26.**

3. Chosen optimal assignment
Pick (1 → 3) with cost 26 (even).
4. Parity check and final cost
even = 1, odd = 0 → satisfies even ≥ odd.
Total cost = **26.**

Case 2

Input:
6
45
6
38
6
15
38

Output:
30

Explanation:

1. Allowed assignments and costs (sorted)
Smallest-cost candidate assignments:
(1 → 3): |45 – 38| = 7 — odd
(1 → 6): |45 – 38| = 7 — odd
s(2 → 5): |6 – 15| = 9 — odd
(4 → 5): |6 – 15| = 9 — odd
(3 → 5): |38 – 15| = 23 — odd
(5 → 6): |15 – 38| = 23 — odd
(1 → 5): |45 – 15| = 30 — even ← smallest even
(2 → 3): |6 – 38| = 32 — even
(2 → 6): |6 – 38| = 32 — even
(3 → 4): |38 – 6| = 32 — even
(4 → 6): |6 – 38| = 32 — even
(1 → 2): |45 – 6| = 39 — odd
(1 → 4): |45 – 6| = 39 — odd

2. Check cheapest possibilities
There are many edges with cost < 30, but the cheapest ones are **odd (7, 9, 23, etc.).**
A single odd edge is invalid (even = 0 < odd = 1).
Combining two small odd edges (e.g. 7 + 7 = 14) still results in odd = 2, even = 0 which is invalid.
Any set that satisfies parity must include at least one even edge. The smallest even edge available
is cost 30 (1 → 5). There is no even with cost < 30 in this instance.
Therefore no valid assignment set can have **total < 30.**

3. Chosen optimal assignment
Pick (1 → 5) with cost **30 (even).**

4. Parity check and final cost
even = 1, odd = 0 → valid.
Total cost = **30.**

Case 3

Input:
6
6
59
1
25
59
50

Output:
24

Explanation:

1. Allowed assignments and costs (sorted). Smallest candidate edges:

(1 → 3): |6 – 1| = 5 — odd
(2 → 6): |59 – 50| = 9 — odd
(5 → 6): |59 – 50| = 9 — odd
(1 → 4): |6 – 25| = 19 — odd
(3 → 4): |1 – 25| = 24 — even ← smallest even
(4 → 6): |25 – 50| = 25 — odd
(2 → 4): |59 – 25| = 34 — even
(4 → 5): |25 – 59| = 34 — even
(1 → 6): |6 – 50| = 44 — even
(3 → 6): |1 – 50| = 49 — odd
(1 → 2): |6 – 59| = 53 — odd
(1 → 5): |6 – 59| = 53 — odd
(2 → 3): |59 – 1| = 58 — even
(3 → 5): |1 – 59| = 58 — even

2. Check cheapest possibilities
The very cheapest edges (5, 9, 9, 19) are odd. Any single odd is invalid. Two odds alone are invalid as well (odd would exceed even).
The smallest even-cost edge is 24 (3 → 4). There is no even with cost < 24, so you cannot form a valid set whose total is smaller than 24.
Combinations that include the small odds plus an even would have total ≥ (small odd + smallest even) ≥ 5 + 24 = 29 > 24, so they are worse than taking the **single even 24.**

3. Chosen optimal assignment
Pick (3 → 4) with cost **24 (even).**

4. Parity check and final cost
even = 1, odd = 0 → valid.

Total cost = **24.**

Complex : Layer-Split Path Maximization with Penalties

You are given an undirected graph with **N** nodes and **M** edges.
Each node u has a layer **L[u]** (integer from 1 to **K**) and a value **V[u]**.
You must choose a simple path (no repeated nodes) such that:

- Layer Constraint Along the chosen path, the sequence of layers must be non-decreasing:
- L[u1] ≤ L[u2] ≤ ... ≤ L[ut]

Penalty for Layer Jumps whenever the path moves from a node with layer x to layer y where y > x, you pay a cost: penalty = $(y - x)^2$

Find the **maximum value** of **(sum of V[u] over the path) – (sum of penalties).**

Input Format

- The first line contains an integer, N, denoting the number of rows in layers.
- The next line contains an integer, M, denoting the number of rows in edges.
- The next line contains an integer, K, denoting the max layer.
- Each line i of the N subsequent lines (where 0 ≤ i < N) contains 2 space separated integers each
- describing the row layers[i].
- Each line i of the M subsequent lines (where 0 ≤ i < M) contains 2 space separated integers each
- describing the row edges[i].
- Each row in edges[i] contains 2 space separated integers u and v denoting an edge between node
- u and node v.

Constraints

$1 <= N <= 10^5$

$1 <= M <= 10^5$

$1 <= K <= 10^5$

-10^9 <= layers[i][j] <= 10^9

0 <= edges[i][j] <= N-1

Sample Test Cases

Case 1

Input:

```
2
1
10
1 10
3 100
0 1
```

Output:

```
106
```

Explanation:

Only possible simple path: 0 → 1

Layers: 1 → 3 (valid, non-decreasing)

Penalty = $(3 – 1)^2$ = 4

Value gained = 10 + 100 = 110

Final score = 110 – 4 = 106

Output = 106

Case 2

Input:

```
3
2
3
1 10
2 20
3 30
0 1
1 2
```

Output:

```
58
```

Explanation:

Path: 0 → 1 → 2

Layer sequence: 1 → 2 → 3 (valid)

Penalties:

1→2 : $(2−1)^2$ = 1

2→3 : $(3−2)^2$ = 1

Total penalty = 2

Value sum = 10 + 20 + 30 = 60

Final = 60 – 2 = 58

Output = 58

Case 3

Input:

```
3
2
3
1 -5
2 100
3 -10
0 1
1 2
```

Output:

```
100
```

Explanation:

Possible path: 0 → 1 → 2

But node 0 has negative value, and node 2 has negative value.

So best choice is to start the path at node 1 only.

Path: 1

Value = 100

No penalties

Score = 100

Why not 1 → 2 ?

Penalty = $(3−2)^2$ = 1

Value = 100 + (−10) = 90

Total = 90 – 1 = 89 < 100

Why not 0 → 1 ?

Value = −5 + 100 = 95

Penalty = $(2−1)^2$ = 1

Final = 94 < 100

Output = 100