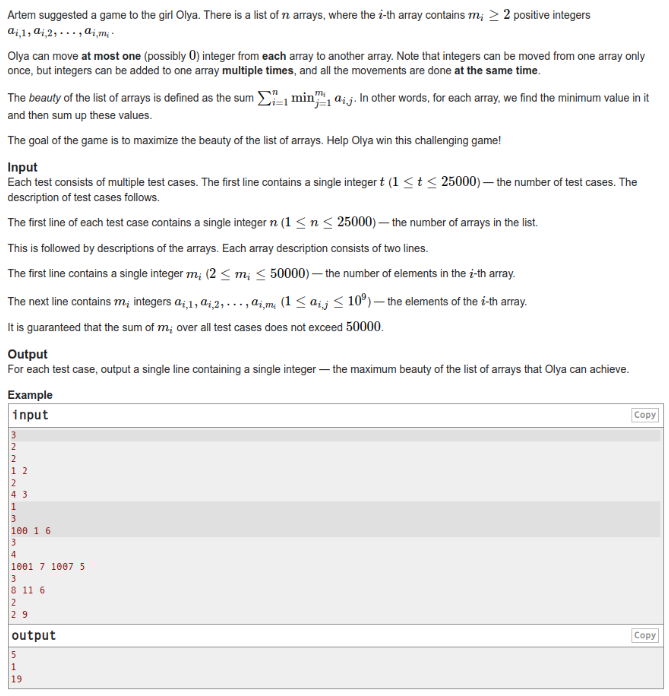
Olya and the Game with arrays



Artem suggested a game to the girl Olya. There is a list of n arrays, where the i-th array contains $m_i \geq 2$ positive integers $a_{i,1}, a_{i,2}, \ldots, a_{i,m_i}$.

Olya can move **at most one** (possibly 0) integer from **each** array to another array. Note that integers can be moved from one array only once, but integers can be added to one array **multiple times**, and all the movements are done **at the same time**.

The *beauty* of the list of arrays is defined as the sum $\sum_{i=1}^{n} \min_{j=1}^{m_i} a_{i,j}$. In other words, for each array, we find the minimum value in it and then sum up these values.

The goal of the game is to maximize the beauty of the list of arrays. Help Olya win this challenging game!

Input
Each test consists of multiple test cases. The first line contains a single integer t ($1 \leq t \leq 25000$) — the number of test cases. The description of test cases follows.

The first line of each test case contains a single integer n ($1 \leq n \leq 25000$) — the number of arrays in the list.

This is followed by descriptions of the arrays. Each array description consists of two lines.

The first line contains a single integer m_i ($2 \leq m_i \leq 50000$) — the number of elements in the i-th array.

The next line contains m_i integers $a_{i,1}, a_{i,2}, \ldots, a_{i,m_i}$ ($1 \leq a_{i,j} \leq 10^9$) — the elements of the i-th array.

It is guaranteed that the sum of m_i over all test cases does not exceed 50000.

Output
For each test case, output a single line containing a single integer — the maximum beauty of the list of arrays that Olya can achieve.

Example

input Copy

```
3
2
2
1 2
2
4 3
1
3
100 1 6
3
4
1001 7 1007 5
3
8 11 6
2
2 9
```

output Copy

```
5
1
19
```

→ Approach :-

- Observe that only min and second min elements are responsible for answer.
- So we just sort the array.
- Send all the elements to that array which has smallest second min value.

formula : min + sum of all second min element

− second min. of second min element.

```
#include <bits/stdc++.h>
#include <climits>

using namespace std;

static int cmp(vector<int> &v1, vector<int> &v2) {
  return v1[1] < v2[1];
}

long long getBeauty(vector<vector<int>> input) {
  for (int i = 0; i < input.size(); i++) {
    sort(input[i].begin(), input[i].end());
  }
  sort(input.begin(), input.end(), cmp);

  long long beautySum = 0;

  int mini = INT_MAX;
  int index = -1;

  for(int i = 0; i < input.size(); i++) {
    int first = input[i][0];

    if(mini > first) {
      mini = first;
      index = i;
    }
  }

  for(int i = 0; i < input.size(); i++) {
    int first = input[i][0];
    int second = input[i][1];

    if(i == 0) {
      beautySum += mini;
    } else {
      beautySum += second;
    }
  }

  return beautySum;
}

int main() {
  int t;
  cin >> t;
  while (t--) {
    int n;
    cin >> n;
    vector<vector<int>> input;
    while (n--) {
      int size;
      cin >> size;
      vector<int> arr;
      for (int i = 0; i < size; i++) {
        int val;
        cin >> val;
        arr.push_back(val);
      }
      input.push_back(arr);
    }
    cout << getBeauty(input) << endl;
  }

  return 0;
}
```

Training Before the Olympiad

Masha and Olya have an important team olympiad coming up soon. In honor of this, Masha, for warm-up, suggested playing a game with Olya:

There is an array a of size n. Masha goes first, and the players take turns. Each move is described by the following sequence of actions:

● If the size of the array is 1, the game ends.

● The player who is currently playing chooses two **different** indices i, j $(1 \le i, j \le |a|)$, and performs the following operation — removes a_i and a_j from the array and adds to the array a number equal to $\lfloor \frac{a_i + a_j}{2} \rfloor \cdot 2$. In other words, first divides the sum of the numbers a_i, a_j by 2 rounding down, and then multiplies the result by 2.

Masha aims to maximize the final number, while Olya aims to minimize it.

Masha and Olya decided to play on each non-empty prefix of the initial array a, and asked for your help.

For each $k = 1, 2, \ldots, n$, answer the following question. Let only the first k elements of the array a be present in the game, with indices $1, 2, \ldots, k$ respectively. What number will remain at the end with optimal play by both players?

Input

The first line contains an integer t $(1 \le t \le 10^4)$ — the number of test cases.

The first line of each test case contains a single integer n $(1 \le n \le 10^5)$ — the size of the array.

The second line contains n integers $a_1, a_2, \ldots, a_n$ $(1 \le a_i \le 10^9)$ — the array on which Masha and Olya play.

It is guaranteed that the sum of n over all test cases does not exceed 10^5.

Output

For each test case, output n integers. The k-th of these numbers should be equal to the number that will remain at the end with optimal play by both players, on the array consisting of the first k elements of the array a.

Example

input

```
4
1
31
6
6 3 7 2 5 4
3
3 10 11
5
7 13 11 19 1
```

output

```
31
6 8 16 18 22 26
3 12 24
7 20 30 48 50
```

→ Approach :-

- Observe that Olya will try to add one even and one odd number, so as to reduce the sum by 1.
- Let's say we have

  x odd & $(n-x)$ even no.

- Masha adds two odd no. to make one even.

  $(x-2)$ odd & $(n-x+1)$ even no.

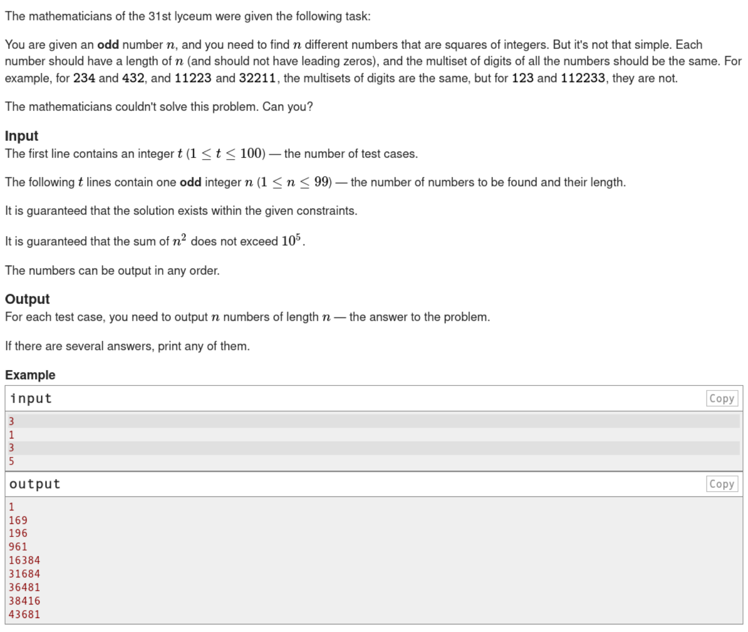
- Olya add 1 odd & 1 even to make 1 odd no.

  $(x-2)$ odd & $(n-x)$ even no.

- Note that the no. of even no. remains constant.

```cpp
#include <iostream>
#include <vector>
using namespace std;

vector<long long> computeFinalNumbers(int n, const vector<int> &a) {
    vector<long long> result;
    long long current_sum = 0;
    int odd_count = 0;

    for (int k = 0; k < n; ++k) {
        current_sum += a[k];
        if (a[k] % 2 != 0) {
            odd_count++;
        }
        if (k == 0) {
            result.push_back(a[0]);
            continue;
        }

        long long final_value;
        if (odd_count % 3 == 0) {
            final_value = current_sum - (odd_count / 3);
        } else if (odd_count % 3 == 1) {
            final_value = current_sum - (odd_count / 3) - 1;
        } else {
            final_value = current_sum - (odd_count / 3);
        }

        result.push_back(final_value);
    }

    return result;
}

int main() {

    int t;
    cin >> t;
    vector<vector<long long>> results;

    for (int i = 0; i < t; ++i) {
        int n;
        cin >> n;
        vector<int> a(n);
        for (int j = 0; j < n; ++j) {
            cin >> a[j];
        }
        results.push_back(computeFinalNumbers(n, a));
    }

    // Output the results
    for (const auto &result : results) {
        for (long long num : result) {
            cout << num << " ";
        }
        cout << endl;
    }

    return 0;
}
```

Mathematical Problem



The mathematicians of the 31st lyceum were given the following task:

You are given an **odd** number n, and you need to find n different numbers that are squares of integers. But it's not that simple. Each number should have a length of n (and should not have leading zeros), and the multiset of digits of all the numbers should be the same. For example, for 234 and 432, and 11223 and 32211, the multisets of digits are the same, but for 123 and 112233, they are not.

The mathematicians couldn't solve this problem. Can you?

Input

The first line contains an integer t ($1 \leq t \leq 100$) — the number of test cases.

The following t lines contain one **odd** integer n ($1 \leq n \leq 99$) — the number of numbers to be found and their length.

It is guaranteed that the solution exists within the given constraints.

It is guaranteed that the sum of n^2 does not exceed 10^5.

The numbers can be output in any order.

Output

For each test case, you need to output n numbers of length n — the answer to the problem.

If there are several answers, print any of them.

Example

input

```
3
1
3
5
```

output

```
1
169
196
961
16384
31684
36481
38416
43681
```

→ Approach :-

- For $n = 1$, answer is 1.
- For $n = 3$, hard code the answer,
  ie 196, 169 & 961.
- For $n+2$, we will have the values of n, multiply each of the values with 100 so we will have n values.
- Now, we are only sort of 2 more values.
- Now, add zeroes in b/w 9_6_1 & 1_6_9 to get the remaining two value
- No. of zeroes to be added

  $3 \to 0$

  $5 \to 2$ (one in each space)

  $7 \to 4$ $\hookrightarrow \dfrac{i-1}{2}$, where $i = 5, 7, 9, \ldots$

  $9 \to 6$

```cpp
#include <bits/stdc++.h>

using namespace std;

vector<string> getTheSquaresofLenN(int n) {
  if (n == 1)
    return {"1"};
  vector<string> ans = {"169", "196", "961"};
  if (n == 3)
    return ans;

  for (int i = 5; i <= n; i += 2) {
    for (int j = 0; j < ans.size(); j++) {
      string temp = ans[j];
      temp += "00";
      ans[j] = temp;
    }
    int zeroes = (i - 3) / 2;
    string temp1 = "961", temp2 = "169";
    string toBeInserted = "";
    for(int i = 0; i < zeroes; i++) {
        toBeInserted += "0";
    }
    temp1.insert(1, toBeInserted);
    temp2.insert(1, toBeInserted);
    temp1.insert(2 + zeroes, toBeInserted);
    temp2.insert(2 + zeroes, toBeInserted);
    ans.push_back(temp1);
    ans.push_back(temp2);
  }
  return ans;
}

int main() {
  int t;
  cin >> t;
  while (t--) {
    int n;
    cin >> n;
    vector<string> ans = getTheSquaresofLenN(n);
    for (int i = 0; i < n; i++) {
      cout << ans[i] << endl;
    }
  }

  return 0;
}
```

Minimum increament to make array unique

You are given an integer array `nums`. In one move, you can pick an index `i` where `0 <= i < nums.length` and increment `nums[i]` by `1`.

Return *the minimum number of moves to make every value in* `nums` ***unique***.

The test cases are generated so that the answer fits in a 32-bit integer.

Example 1:

```
Input: nums = [1,2,2]
Output: 1
Explanation: After 1 move, the array could be [1, 2, 3].
```

Example 2:

```
Input: nums = [3,2,1,2,1,7]
Output: 6
Explanation: After 6 moves, the array could be [3, 4, 1, 2, 5, 7].
It can be shown with 5 or less moves that it is impossible for the array to have all
unique values.
```

Constraints:

- `1 <= nums.length <= 10^5`
- `0 <= nums[i] <= 10^5`

→ Naive :-

• We will traverse through the array and store the already encountered number in an unordered_set.

```
class Solution {
public:
    int minIncrementForUnique(vector<int>& nums) {
        int incRequired = 0;
        unordered_set<int> st;

        for(int i = 0; i < nums.size(); i++) {
            int temp = nums[i];
            if(st.find(nums[i]) ≠ st.end()) {
                while(st.find(temp) ≠ st.end()) {
                    incRequired++;
                    temp++;
                }
            }
            st.insert(temp);
        }
        return incRequired;
    }
};
```

Time Complexity : $O(n^2)$

Space Complexity : $O(n)$

→ Optimized :-

3 2 1 2 1 7

1 → 2 mini = -1

2 → 2 ∵ mini < key

3 → 1

7 → 1

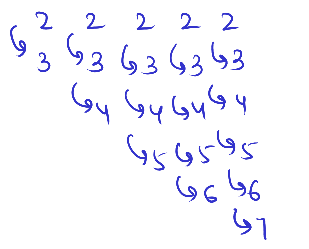
- That means, till now the key element has not occurred yet.
- So, we can reduce the freq. of key by 1. Now, key element has occured.
- So, the remaining values (key) which are equal to key needs to be increamented.
- 1 → 1 mini = 1

   2 → 2 $+1 \overset{1}{\hookrightarrow}{}_{2}$

   3 → 1

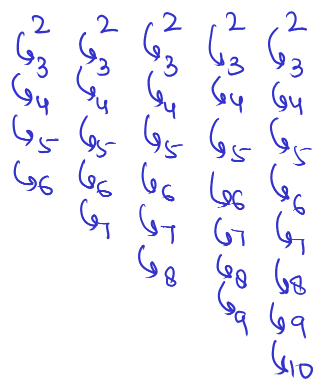
   7 → 1

- Now, mini = 2
  - if the freq. of 2 has been like 5.



1 2 3 4 5 (AP)

  - what if the mini would have been 5



4 5 6 7 8 (AP)

↑
a

- Now, we just need to find

$$S_n = \frac{n}{2}\{2a + (n-1)d\}.$$

$a = (\text{mini} - \text{key} + 1)$

$5 - 2 + 1 = 4$

$n = \text{freq.}$

$d = 1$

```
class Solution {
public:
    int minIncrementForUnique(vector<int>& nums) {
        int incRequired = 0;
        map<int, int> st;

        for(int i = 0; i < nums.size(); i++) {
            st[nums[i]]++;
        }

        int mini = -1;
        for(auto [key, value]: st) {
            int freq = value;
            if(mini < key) {
                freq--;
                mini = key;
            }
            int increament = ((freq) * (2 * (mini - key + 1) + freq - 1)) / 2;
            incRequired = incRequired + increament;

            mini = mini + freq;
        }

        return incRequired;
    }
};
```

Time Complexity: O(nlogn)

Space Complexity: O(n)

→ Better Approach:-

- The above solⁿ can be modified to use array instead of map for better time complexity.

```
class Solution {
public:
    int minIncrementForUnique(vector<int>& nums) {
        int n = nums.size();
        int max_val = 0;
        int minIncrements = 0;

        // Find maximum value in nums using a loop
        for (int val : nums) {
            max_val = max(max_val, val);
        }

        // Create a frequencyCount vector to store the frequency of each value
        // in nums
        vector<int> frequencyCount(n + max_val + 1, 0);

        // Populate frequencyCount vector with the frequency of each value in
        // nums
        for (int val : nums) {
            frequencyCount[val]++;
        }

        // Iterate over the frequencyCount vector to make all values unique
        for (int i = 0; i < frequencyCount.size(); i++) {
            if (frequencyCount[i] <= 1) continue;

            // Determine excess occurrences, carry them over to the next value,
            // ensure single occurrence for current value, and update
            // minIncrements.
            int duplicates = frequencyCount[i] - 1;
            frequencyCount[i + 1] += duplicates;
            frequencyCount[i] = 1;
            minIncrements += duplicates;
        }

        return minIncrements;
    }
};
```

Time Complexity: O(n + maxx)

Space Complexity: O(n+maxx)

Closest three sum

Given an array, **arr** of integers, and another number **target**, find three integers in the array such that their sum is closest to the target. Return the sum of the three integers.

Note: If there are multiple solutions, return the **maximum** one.

Examples :

Input: arr[] = [-7, 9, 8, 3, 1, 1], target = 2
Output: 2
Explanation: There is only one triplet present in the array where elements are -7,8,1 whose sum is 2.

Input: arr[] = [5, 2, 7, 5], target = 13
Output: 14
Explanation: There is one triplet with sum 12 and other with sum 14 in the array. Triplet elements are 5, 2, 5 and 2, 7, 5 respectively. Since abs(13-12) ==abs(13-14) maximum triplet sum will be preferred i.e 14.

→ Approach :-

- Sort the array.
- Now, iterate to each element using 3 pointers to get the required sum.

```
class Solution {
  public:
    int threeSumClosest(vector<int> arr, int target) {
        sort(arr.begin(), arr.end());
        int n = arr.size();
        int ans = INT_MAX;
        for(int i = 0; i < n; i++) {
            int start = i + 1, end = n - 1;

            while(start < end) {
                int sum = arr[start] + arr[end] + arr[i];

                if(sum == target) return target;
                else if(sum > target) {
                    end--;
                } else {
                    start++;
                }

                if(abs(ans - target) > abs(sum - target)) ans = sum;
                if(abs(ans - target) == abs(sum - target)) {
                    ans = max(ans, sum);
                }
            }
        }
        return ans;
    }
};
```

Time Complexity : $O(N^2)$
Space Complexity : $O(1)$