Лабораторная работа 3.

Массивы

Часть 0.

Задание 1. С-массив против std::vector: «Почему длина стала 2, а сумма — мусор?»

Цель. Понять ключевые отличия int arr[] от std::vector<int>: фиксированный размер, отсутствие проверки границ, «затухание» массива до указателя при передаче в функцию, вычисление длины, а также преимущества vector.

1) Скопируйте и запустите код

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

// Задача: прочитать n чисел, вывести их сумму и "длину".
void printAndSum(int arr[]) {                // BUG: arr уже "указатель"
    int n = sizeof(arr) / sizeof(arr[0]);    // BUG: размер указателя / sizeof(int)
    int sum = 0;
    for (int i = 0; i <= n; ++i) {           // BUG: <= и еще и неверный n
        sum += arr[i];                       // UB: выход за пределы
    }
    cout << "n=" << n << " sum=" << sum << "\n";
}

int main() {
    int n; cin >> n;
    int arr[n];                              // BUG: VLA — не стандарт C++ (зависит от компилятора
    for (int i = 0; i < n; ++i) cin >> arr[i];
    printAndSum(arr);
    cout << arr[n] << "\n";                  // BUG: выход за границу
}
```

2) Прогоните контрольные входы

- n=3, a=1 2 3 → ожидается n=3 sum=6, фактически получите «странные» числа.
- n=1, a=10 → возможен крах/мусор.

3) «Зонды» для диагностики

- Выведите sizeof(arr) внутри printAndSum — увидите размер указателя, а не массива.
- Добавьте cerr << "&arr=" << (void*)arr << "\n"; — убедитесь, что это адрес, а не контейнер.

4) Исправление (два канонических пути)

Вариант A (современный, рекомендуемый): std::vector

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

void printAndSum(const vector<int>& a) {
    long long sum = 0;
    for (size_t i = 0; i < a.size(); ++i) sum += a[i];      // или range-for
    cout << "n=" << a.size() << " sum=" << sum << "\n";
}

int main() {
    size_t n; cin >> n;
    vector<int> a(n);
    for (size_t i = 0; i < n; ++i) cin >> a[i];
    printAndSum(a);
    // cout << a.at(n) << "\n";      // .at() бросит исключение: защита от выхода за границы
}
```

Вариант B (если всё же сырой массив)

- Обязательно передавайте длину: void f(int* a, int n);
- никогда не рассчитывайте n внутри функции через sizeof(a);
- аккуратно используйте границы цикла < n.

5) Мини-сводка-теория (коротко)

- int arr[] — фикс. размер, не знает свою длину, при передаче «затухает» до int*, нет проверки границ.
- std::vector<int> — динамический размер, знает .size(), методы .push_back()/.resize(), проверка .at(), безопаснее.

Критерии приёма. Исправленный код выводит корректные n и sum на тестах; нет выходов за границы; есть краткий ответ «почему sizeof дал размер указателя».

Задание 2. std::vector: «Почему size=0 после reserve? почему тормозит insert в начало?»

Цель. Разобраться с size vs capacity, reserve vs resize, сложностью вставок и инвалидацией итераторов.

1) Скопируйте и запустите код

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

int main() {
    int n; cin >> n;

    vector<int> v;
    v.reserve(n);                   // BUG: capacity растет, size остаётся 0
    for (int i = 0; i < n; ++i)
        v[i] = i;                   // BUG: запись по [] вне size -> UB

    cout << "size=" << v.size() << "\n";     // ожидали n? получите 0/крах

    // Очень неэффективно: вставляем в начало в цикле — квадратичная сложность
    for (int x = 1; x <= 5000; ++x) v.insert(v.begin(), x);

    // Инвалидация итераторов: push_back меняет буфер
    for (auto it = v.begin(); it != v.end(); ++it)
        if (*it % 2 == 0) v.push_back(*it);  // BUG: it может стать невалидным

    cout << "ok\n";
}
```

2) Что проверить и объяснить

- Почему reserve не меняет size (только вместимость).
- Почему запись v[i] валидна лишь для i < size.
- Почему insert(v.begin(), …) в цикле медленно (сдвиг элементов каждый раз).
- Почему push_back во время прохода итератором инвалидирует it.

3) Исправление (пошагово)

1. Замените reserve на resize или используйте push_back:

```cpp
vector<int> v;
v.resize(n);
for (int i = 0; i < n; ++i) v[i] = i;
// или:
// for (int i = 0; i < n; ++i) v.push_back(i);
```

2. Вместо вставок в начало — накапливайте в конец и затем reverse(v.begin(), v.end()); (или формируйте во временный tmp, потом v.swap(tmp)), при необходимости v.reserve(total) заранее.
3. Чтобы удваивать контейнер значениями, **не** модифицируйте v по iterator-циклу: сохраните исходный размер и индексируйтесь:

```
size_t old = v.size();
v.reserve(old * 2);
for (size_t i = 0; i < old; ++i) v.push_back(v[i]);
```

Критерии приёма. После правки size == n, программа не падает. Вставки в начало заменены на линейное решение + reverse. Нет инвалидации итераторов; добавление в конец выполнено безопасно.

Задание 3. Двумерные структуры: «Почему строки пустые, индексы путаются, а сумма колонки неверна?»

Цель. Отработать правильную инициализацию 2D через vector<vector<int>>, понять разницу reserve/resize, индексацию [i][j] и альтернативу «плоским» массивом (row-major).

1) Скопируйте и запустите код

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

// Задача: прочитать матрицу n×m, заполнить числами i+j, затем посчитать сумму столбца col.
int main() {
    int n, m, col;
    cin >> n >> m >> col;

    // Вариант №1: "плоский" C-массив от размера ввода (VLA) — не стандарт C++
//  int a[n][m];                    // BUG: не переносимо и опасно, индексы могут выйти за границы

    // Вариант №2: vector<vector<int>> с reserve -> UB при обращении по []
    vector<vector<int>> A;
    A.reserve(n);                   // BUG: capacity для вектора строк, НО size = 0
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < m; ++j)
            A[i][j] = i + j;        // BUG: обращение к несуществующей строке/элементу

    long long sum = 0;
    for (int i = 1; i <= n; ++i)    // BUG: off-by-one и перепутаны индексы
        sum += A[col][i];           // BUG: нужно A[i][col]

    cout << "sum=" << sum << "\n";
}
```

2) Что починить (пошагово)

1. Правильная инициализация 2D-вектора:

```
vector<vector<int>> A;
A.assign(n, vector<int>(m));          // или: vector<vector<int>> A(n, vector<int>(m));
for (int i = 0; i < n; ++i)
    for (int j = 0; j < m; ++j)
        A[i][j] = i + j;
```

2. Проверка границ 0 ≤ col < m, корректная индексация и диапазоны в цикле:

```
if (col < 0 || col >= m) { cout << "ERROR\n"; return 0; }
long long sum = 0;
for (int i = 0; i < n; ++i) sum += A[i][col];
cout << "sum=" << sum << "\n";
```

3. Бонус: «плоское» хранение в одном векторе (row-major) — быстрее и компактнее:

```
vector<int> B(n * m);
auto at = [&](int i, int j) -> int& { return B[i * m + j]; };
for (int i = 0; i < n; ++i)
  for (int j = 0; j < m; ++j)
    at(i,j) = i + j;
long long sum2 = 0;
for (int i = 0; i < n; ++i) sum2 += at(i, col);
```

3) Контрольные сценарии

- n=2, m=3, col=1. Матрица i+j:

```
[0,1,2]
[1,2,3]
```

Сумма столбца 1 = 1 + 2 = 3.

- col = 0 → сумма 0 + 1 = 1.
- Неверный col=-1 или col≥m → ERROR.

Критерии приёма.

- Отсутствуют VLA и обращения к несуществующим элементам.
- Верная инициализация (assign/resize), корректные границы циклов.
- Сумма столбца совпадает с расчетом на тестах.

- (Бонус) Показан эквивалент с «плоским» вектором.

Часть 1.

Выполните следующие задачи по ссылке:

https://informatics.msk.ru/mod/statements/view.php?id=87429#1

Часть 2.

Общие правила для всех заданий

• Используйте std::vector. Тип индекса — size_t или int, но внимательно к переполнениям.

• Не используйте «магические числа»: объявляйте константы (const int BASE=10;).

• Формат вывода: элементы одного вектора печатайте в одну строку через пробел; для матриц — по строкам.

• В каждом варианте приведите минимум 3 граничных теста (например, пустой результат, «все подходят», «никто не подходит», границы индексов, одиночные элементы).

Задание 1. Фильтр/преобразование вектора

Ввод: N (1..100000), затем N целых.

Вывод: новый вектор B, полученный из A **по условию варианта** и/или преобразованием. Печатать B в одну строку через пробел (если B пуст — печатать пустую строку).

Требования: проход по A одним или двумя циклами; не использовать erase в середине вектора по одному элементу (при необходимости сформируйте B через push_back).

Выберите свой номер:

1. Оставьте в B числа из A, у которых сумма цифр — простая; в B кладите эту сумму (а не исходное число).
2. Оставьте числа, взаимно простые с S, где S — сумма всех элементов A; в B запишите исходные числа.
3. Оставьте числа с неповторяющимися цифрами (все десятичные цифры разные); B = исходные числа.
4. Оставьте числа, у которых ровно два единичных бита в двоичной записи; B = исходные.

5. Оставьте полупростые i=p·q (p,q — простые, допускается p=q); в B запишите упорядоченную пару p q как два соседних числа.
6. Оставьте числа, у которых Наибольший_общий_делитель(i, sumDigits(i)) = 1; B = исходные.
7. Оставьте палиндромы (десятичная запись равна реверсу); B = исходные.
8. Оставьте числа, которые делятся на произведение ненулевых цифр; B = исходные.
9. Оставьте числа, для которых первая и последняя цифра совпадают; B = исходные.
10. Оставьте числа, у которых количество делителей = 4; B = исходные.
11. Оставьте квадраты или кубы; в B положите корень (для квадратов) или целый корень куба.
12. Оставьте числа Фибоначчи (≤ max(A)); B = исходные.
13. Оставьте числа Харшада (делятся на сумму цифр); B = исходные.
14. Оставьте числа с чередующейся чётностью цифр; B = исходные.
15. Оставьте числа, чей квадрат заканчивается на исходное число (автоморфные); B = исходные.
16. Оставьте числа, у которых битовая чётность (кол-во единичных битов) — нечётная; B = исходные.
17. Оставьте числа, для которых i % firstDigit(i) == 0; B = исходные.
18. Оставьте числа, у которых нет простых делителей из множества {2,3,5}; B = исходные.
19. Оставьте числа, являющиеся треугольными (k(k+1)/2); B = исходные.
20. Оставьте числа, у которых φ(i) (функция Эйлера) — чётная; B = исходные.

Задание 2. Таблица значений по вектору X: функция + дополнительный расчёт

Ввод: N, затем N вещественных x_i.

Вывод: для каждого x_i — x_i и f(x_i) с fixed << setprecision(3); после таблицы— дополнительная величина варианта.

Требования: пропускать элементы, для которых нарушен домен (sqrt<0, ln≤0), вести счётчик пропусков и печатать его в конце.

Выберите свой номер:

1. f(x)=x*x-3x+2. Доп.: min f, max f.
2. f(x)=sin(x)/x, считать f(0)=1. Доп.: число смен знака f.
3. f(x)=|x|+ln(x+2) (домен x>-2). Доп.: сколько строк пропущено.
4. f(x)=sqrt(5-x) (домен x≤5). Доп.: среднее f.
5. f(x)=exp(-x)*cos(x). Доп.: индекс x, где |f| максимален.
6. f(x)=ln(1+x*x) (всюду). Доп.: трапецеидурная оценка интеграла по всей выборке как по неравномерной сетке (соседи по исходному порядку).
7. f(x)=x/(1+x*x). Доп.: кол-во i, где |f(x_i)|<0.1.
8. f(x)=sqrt(|x-1|)+ln(x+3) (домен x>-3). Доп.: медиана f (если N нечётно; иначе среднее двух середин после **временной** сортировки копии значений f).
9. f(x)=tanh(x). Доп.: первая позиция, где f(x)>0.9 (иначе -1).
10. f(x)=sqrt(4-x*x) (домен |x|≤2). Доп.: площадь по трапециям при сортировке x по возрастанию.
11. f(x)=ln(x) (домен x>0). Доп.: кол-во i, где дробная часть f < 0.5.
12. f(x)=cosh(x)-1. Доп.: max f.
13. f(x)=1/(1+x*x). Доп.: число i, где f(x_i)≥0.5.
14. f(x)=x^3-2x. Доп.: кол-во локальных экстремумов по последовательности f(x_i) (сравнение соседей).
15. f(x)=ln(x+1)/(x+1) (домен x>-1). Доп.: сумма положительных значений f.
16. f(x)=|sin(3x)|. Доп.: доля i, где f>0.8.
17. f(x)=x*exp(-x). Доп.: индекс максимума.
18. f(x)=atan(x). Доп.: кол-во попаданий f∈(0.5,1.0).
19. f(x)=ln(2-x) (домен x<2). Доп.: min f.
20. f(x)=sqrt(x)+sqrt(3-x) (домен 0≤x≤3). Доп.: max f.

Задание 3. Обработка последовательностей: участки/окна/префиксы

Ввод: N, затем N целых (или вещественных — указано).

Вывод: одно значение или вектор в соответствии с вариантом.

Требования: никаких контейнеров кроме vector; все вычисления — одним/двумя проходами где возможно.

Выберите свой номер:

1. Длина максимального неубывающего подотрезка.
2. Длина максимального строго возрастающего подотрезка.
3. Кол-во локальных максимумов (строго больше обоих соседей).
4. Кол-во смен знака между соседями.
5. Самая длинная серия равных подряд; вывести длину и значение серии (значение — первое из серии).
6. Кол-во отрезков длины ≥2, где чётность элементов чередуется.
7. Максимальная сумма подмассива (алгоритм Кадане).
8. Максимальная сумма подмассива фиксированной длины K (ввод K, 1≤K≤N) — «скользящее окно».
9. Сумма на каждом окне длины K → вывести вектор из N-K+1 сумм.
10. Кол-во пар i<j, где a_i > a_j (число инверсий). Подсказка: используйте модифицированный merge-count (O(N log N)).
11. Кол-во элементов, строго больших среднего всех предыдущих (онлайн).
12. Кол-во элементов, для которых |a_i - a_{i-1}| = 1 (соседние «ступеньки»).
13. Длина максимального нулевого подмассива (подряд идущие нули).
14. Кол-во пар однонаправленных соседей: (a_{i-1}≤a_i≤a_{i+1}) или (a_{i-1}≥a_i≥a_{i+1}).
15. Кол-во индексов i, где a_i — медиана тройки (a_{i-1}, a_i, a_{i+1}) (для 1..N-2).
16. Кол-во «пиковых плато»: серия одинаковых значений длины ≥2, такая что соседние элементы слева и справа меньше значений плато.
17. Длина максимального подмассива с суммой = 0 (целые).
18. Длина максимального подмассива с количеством единиц = количеством нулей (для массива из 0/1).
19. Кол-во индексов i, где a_i — делитель a_{i+1} (для i=0..N-2).
20. Кол-во индексов i, где a_i — новый максимум по префиксу (строго больше всех предыдущих).

Задание 4. Двумерные массивы (vector<vector<int>>).

Ввод: n m (1..1000), затем n×m целых.

Вывод: зависит от варианта (матрица/значение).

Требования: инициализируйте как vector<vector<int>> A(n, vector<int>(m)); или «плоским» vector<int> B(n*m) с адресацией B[i*m+j]. Корректно проверяйте границы.

Выберите свой номер:

1. Переставьте строки так, чтобы суммы строк шли по невозрастанию. Выведите матрицу.
2. Удалите нулевые столбцы (все элементы столбца = 0). Выведите новую матрицу.
3. Поменяйте местами столбцы с минимальной и максимальной суммой.
4. Транспонируйте квадратную матрицу (если не квадрат — вывести ERROR).
5. Обнулите элементы ниже главной диагонали; выведите матрицу.
6. Замените каждый элемент на сумму четырёх соседей (вверх/вниз/лево/право), границы считать имеющими недостающих соседей = 0.
7. Найдите строку с максимальным количеством локальных максимумов (элемент больше четырёх соседей по сторонам, учитывая границы). Выведите индекс строки (0-based).
8. Нормализуйте каждую строку: вычесть из неё минимум строки, затем (при желании) делить на (max-min) — если (max==min), оставить строку нулями.
9. «Крест»: обнулите все элементы на центральных строке и столбце (если n и m нечётные), иначе — обнулите ближайшие к центру две строки/столбца.
10. Поверните матрицу на 90° по часовой (только для квадратной, иначе ERROR).
11. Поменяйте местами минимальный и максимальный элемент матрицы (если несколько — брать первое в порядке чтения).
12. Для каждого столбца выведите количество уникальных значений в нём.
13. Постройте вектор S из сумм по столбцам; выведите S.

14. Сформируйте вектор индексов столбцов, где элементы строго возрастают сверху вниз.
15. Обнулите «контур» (первую и последнюю строку/столбец); выведите матрицу.
16. Замените каждый элемент на среднее арифметическое по окну 3×3 (границы: учитывайте только реально существующие соседи; целочисленное деление округлять вниз).
17. Найдите след симметрии: выведите YES, если матрица симметрична относительно главной диагонали (квадратная), иначе NO.
18. Для каждого i отсортируйте строку i по возрастанию, затем перестройте матрицу так, чтобы суммы строк шли по невозрастанию.
19. Найдите самую длинную диагональ (в направлении главной) с одинаковыми элементами; выведите её длину.
20. Удалите повторяющиеся строки (оставьте первое вхождение каждого шаблона строки). Выведите матрицу.

Задание 5. Частоты, «моды», уникальность, слияние

Ввод: зависят от варианта (обычно N и элементы).

Вывод: одно значение или вектор.

Требования: частоты реализуйте в vector<int> (например, для цифр 0..9 — размер 10), не используйте ассоциативные контейнеры.

Выберите свой номер:

1. Подсчитайте **моду** (наиболее частое значение) для целых A в диапазоне [-1000..1000]. При равенстве — наименьшее значение.
2. Подсчитайте частоты **цифр 0..9** во всех числах A (по десятичным цифрам каждого числа), выведите 10 чисел.
3. Проверьте, является ли B **перестановкой** A (вводится N, затем A, затем B той же длины).
4. Удалите **повторяющиеся элементы**, сохранив первый порядок появления (stable unique). Выведите результат.
5. Найдите **количество различных** значений в A (используйте сортировку копии или отметки в частотном массиве, если диапазон мал).
6. Слейте **два отсортированных** массива A и B в один отсортированный C.

7. Подсчитайте частоты остатков A_i % K (ввод K>0), выведите вектор из K частот.
8. Проверьте, есть ли в A **большинство** (majority element: встречается > N/2); выведите YES/NO и значение, если YES. (Подсказка: Boyer–Moore + проверка.)
9. Подсчитайте **количество пар равных элементов** (i<j, A_i=A_j).
10. Отсортируйте по **сумме цифр** (при равенстве — по возрастанию самого числа). Выведите отсортированный массив.
11. Стабильно **разделите** массив: сначала элементы, удовлетворяющие условию P(x) (ввод: порог T, P(x): x≥T), затем остальные; относительный порядок внутри групп сохраняется.
12. Сформируйте массив **частот значений** на интервале [L..R] (ввод L,R, L≤A_i≤R).
13. Выведите **k наименьших** элементов (ввод k), сохранив исходный порядок среди выбранных (допускается двухпроходный алгоритм: метки + сбор).
14. Постройте массив **префиксных сумм** и по Q запросам l r отвечайте sum(l..r) (0-based).
15. Подсчитайте **количество инверсий** (как в 33-10) и выведите это число.
16. «Сдвиг» массива вправо на k позиций (циклический).
17. **Проверка на анаграмму чисел**: два массива одинаковой длины равны после перестановки цифр внутри каждого элемента? (Сравнивайте отсортированные «подписей» цифр.)
18. Выведите все элементы, встречающиеся **ровно два раза**, в порядке первого появления.
19. Постройте **гистограмму** длин серий равных значений: массив, где по индексу len стоит количество серий длины len.
20. Удалите элементы, у которых сумма цифр **чётная**, сохранив порядок прочих; выведите результат.

Задание 6. Пары и последовательности в массиве

Ввод: N (1..100000), затем N целых чисел A[i].
У некоторых вариантов есть дополнительный параметр (K, M, T, D) — вводится после N и до массива.

Вывод: то, что требует ваш вариант (одно число, пара индексов/значений, длина отрезка, и т.п.).Если результат неоднозначен — используйте правило из формулировки (например, «при равенстве выбрать пару с меньшим левым индексом, затем с меньшим правым»).

Ограничения/требования:

- Используйте std::vector. Не храните лишние копии, где это не нужно.
- Где возможно — одно- или двухпроходные решения O(N). Сортировку используйте только если это явно допускается.
- Аккуратно обрабатывайте крайние случаи: пустой ответ, N=1, отсутствие требуемых пар/отрезков.

Варианты

1. **Соседние с одинаковой чётностью.** Выведите количество соседних пар (i, i+1), у которых A[i] % 2 == A[i+1] % 2.
2. **Соседние с взаимной простотой.** Выведите число пар (i, i+1), где gcd(A[i], A[i+1]) == 1.
3. **Соседние со сменой знака.** Выведите количество пар (i, i+1), где A[i] * A[i+1] < 0 (ноль не считается ни положительным, ни отрицательным).
4. **Максимальная сумма соседей.** Найдите пару соседей с максимальной суммой A[i] + A[i+1]. Выведите i и i+1. При равенстве сумм — **меньший i**.
5. **Максимальное произведение пары (где угодно).** Найдите пару индексов i < j с максимальным произведением A[i] * A[j]. Выведите значения пары.
   Подсказка: отслеживайте две наибольшие и две наименьшие (самые отрицательные) величины.
6. **Разность = K (где угодно).** Вводится K. Выведите количество пар (i, j), i < j, таких что |A[i] - A[j]| == K.
   Разрешено: отсортировать копию A и пройти двумя указателями.
7. **Сумма кратна M (соседние).** Вводится M>0. Посчитайте пары (i, i+1), где (A[i] + A[i+1]) % M == 0.
8. **Одинаковые элементы на расстоянии K.** Вводится K ≥ 1. Выведите количество пар (i, i+K), где A[i] == A[i+K]. Если N ≤ K, ответ 0.
9. **Битовая «дружба» соседей.** Выведите число пар (i, i+1), где (A[i] & A[i+1]) == 0 (побитовое И равно нулю).

10. **Сумма цифр у пары одинакова (где угодно).** Подсчитайте пары i < j, у которых sumDigits(A[i]) == sumDigits(A[j]). Подсказка: можно посчитать частоты по сумме цифр (диапазон сумм небольшой) и сложить C(cnt, 2).

11. **Максимальный отрезок с чередующейся чётностью.** Найдите длину максимального подотрезка, где чётность соседей чередуется (чёт/нечёт/чёт/…).

12. **Максимальный отрезок со сменой знака.** Длина максимального подотрезка, где знак чередуется (положительный/отрицательный/…; нули обрывают отрезок).

13. **Максимальный отрезок с ограничением шага.** Вводится T ≥ 0. Длина максимального подотрезка, на котором для **каждых соседей** выполняется |A[i] - A[i+1]| ≤ T.

14. **Максимальный «почти константный» отрезок (≤ D различных).** Вводится D ≥ 1. Длина максимального подотрезка, в котором встречается **не более D различных значений**. (Реализуйте «скользящее окно» с подсчётом частот через вспомогательный массив/вектор или с сортировкой значений-индексов, если диапазон мал.)

15. **Максимальный арифметический отрезок.** Длина максимального подотрезка с постоянной разностью A[i+1] - A[i] (не короче 2). Если все одиночные — ответ 1 или 2 по вашей договорённости (укажите в выводе).

16. **Максимальный невозрастающий отрезок.** Длина максимального подотрезка, где A[i] >= A[i+1] для всех соседей.

17. **Максимальный отрезок «бинарный баланс».** Массив содержит только 0/1. Длина максимального подотрезка, в котором #0 == #1. Подсказка: префиксные суммы p[i] по +1 для 1 и -1 для 0; ищем равные p[i] с максимальной дистанцией.

18. **Максимальный отрезок с суммой = 0.** Целые числа. Найдите максимальную длину подотрезка с суммой 0. Подсказка: префикс-суммы + первый индекс каждого значения.

19. **Максимальный отрезок с ограничением по модулю.** Вводится M>0. Длина максимального подотрезка, где сумма элементов на отрезке **кратна M**. Подсказка: префикс-суммы по модулю M + первый индекс каждого остатка.

20. **Максимальный «ступенчатый» отрезок (разность ±1).** Длина максимального подотрезка, на котором |A[i] - A[i+1]| == 1 для всех соседей.