Задача 2: Индексный массив — 1 балл

1. Пронумеровать элементы **с 1** (индексы 1..n)
2. Отсортировать пары (значение, индекс) по значению по возрастанию
3. При равных значениях — меньший индекс раньше
4. Выписать только индексы в полученном порядке

Пример: `7 3 9 1 5 3 8 2 6 4`

знач	7	3	9	1	5	3	8	2	6	4
индекс	1	2	3	4	5	6	7	8	9	10

Сортируем пары по значению: (1,4) (2,8) (3,2) (3,6) (4,10) (5,5) (6,9) (7,1) (8,7) (9,3)

Ответ: `4 8 2 6 10 5 9 1 7 3`

⚠ **Подвохи:**
- Нумерация **с 1**, не с 0
- При дубликатах (две тройки: индексы 2 и 6) — сначала тот у кого **меньший индекс**
- Выписывать нужно **индексы**, а не значения

Задача 6: Хэш-таблица (квадратичные пробы) — 2 балла

1. Вычислить: $h = x \bmod m$ (размер таблицы = m)
2. Если ячейка h **свободна** — вставить
3. Если **занята** (коллизия +1) — пробовать: $(h + 1^2) \bmod m$, $(h + 2^2) \bmod m$, $(h + 3^2) \bmod m$ ...
4. Итоговый ответ: таблица + **общее число коллизий**

Пример: m = 11, вставить: 22, 5, 33, 14. Уже есть: [1]=12, [3]=3, [7]=7

i	0	1	2	3	4	5	6	7	8	9	10
	22	12		3	33	5		7	14		

- 22: $22 \bmod 11 = 0$ → свободно ✓ (0 коллизий)
- 5: $5 \bmod 11 = 5$ → свободно ✓ (0 коллизий)
- 33: $33 \bmod 11 = 0$ → занято! $(33 + 1) \bmod 11 = 1$ → занято! $(33 + 4) \bmod 11 = 4$ → свободно ✓ (2 коллизии)
- 14: $14 \bmod 11 = 3$ → занято! → [4] занято! → [7] занято! → [1] занято! → $(14 + 16) \bmod 11 = 8$ → свободно ✓ (4 коллизии)

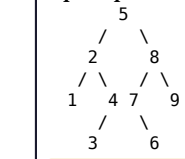
Итого коллизий: 6

⚠ **Подвохи:**
- Считать **все** коллизии (каждое занятое место при попытке вставки = +1)
- Ответ = таблица **и** число коллизий
- Пробы: $i^2 = 1, 4, 9, 16, 25...$ (не $1, 2, 3...$)

Задача 3: Высота СДП (бинарное дерево поиска) — 2 балла

1. Первый элемент = **корень**
2. Каждый следующий: меньше → **влево**, больше/равный → **вправо**
3. Высота = число **рёбер** от корня до самого глубокого листа

Пример: `5 2 8 1 4 7 9 3 6`



Уровень 0: {5}
Уровень 1: {2, 8}
Уровень 2: {1, 4, 7, 9}
Уровень 3: {3, 6}

Высота = 3 (три ребра от корня до 3 или 6)

⚠ **Подвохи:**
- Высота = рёбра, а **не** узлы (у дерева с одним корнем высота = 0)
- Дубликаты **вставляются** вправо, не игнорируются
- Если спрашивают высоту в узлах — это число уровней = высота рёбер + 1

Задача 9: k-NN классификация (k=3) — 2 балла

1. Для каждой тестовой точки T=(x,y) вычислить расстояния до **всех** обучающих точек: $d_i = \sqrt{(x - x_i)^2 + (y - y_i)^2}$
2. Сравнивать можно **квадраты** расстояний (sqrt не нужен)
3. Взять **3 ближайшие** точки, посмотреть их классы
4. Класс T = тот, который встречается **чаще** среди 3 соседей

Пример: T=(3,2), обучающие точки:

Точка	**X_1, X_2**	**Класс**	**d^2 до T(3,2)**
1	1, 1	A	$4 + 1 = 5$
2	2, 1	A	$1 + 1 = 2$
3	5, 4	B	$4 + 4 = 8$
4	6, 5	B	$9 + 9 = 18$
5	6, 6	B	$9 + 16 = 25$
6	2, 5	C	$1 + 9 = 10$
7	1, 6	C	$4 + 16 = 20$

3 наименьших d^2: точки **2 (d^2=2), 1 (d^2=5), 3 (d^2=8)**

Классы: A, A, B → большинство **A**

Ответ: T — класс A

⚠ **Подвохи:**
- Если k=3 и ничья (1A, 1B, 1C) — выбрать класс с **наименьшим** расстоянием
- Брать квадрат расстояния для сравнения — не нужен sqrt, числа те же
- Показывать работу: выписать все расстояния, подчеркнуть 3 минимальных

Альтернатива k-NN: SVM (метод опорных векторов) — 2 балла

Идея: разделяющая прямая $w_1x_1 + w_2x_2 + b = 0$ между классами с **максимальным зазором**.

Классификация точки T: подставить в $f(T) = w_1x_1 + w_2x_2 + b$:
- $f(T) > 0$ → класс **+1**
- $f(T) < 0$ → класс **−1**

Вариант А — дана прямая: просто подставь и смотри знак.

Пример: $2x_1 + x_2 - 6 = 0$, T=(3,1) → $f = 2 \cdot 3 + 1 - 6 = 1 > 0$ → класс **+1**

Вариант Б — построить прямую по опорным векторам:
1. Найти две ближайшие точки разных классов: A (+1) и B (−1)
2. Вектор нормали: $w = A - B$
3. Середина отрезка: $M = \frac{A+B}{2}$
4. Уравнение: $w_1(x_1 - M_1) + w_2(x_2 - M_2) = 0$

⚠ **Подвохи:**
- Вектор нормали **перпендикулярен** самой прямой
- Прямая разделения проходит ровно посередине между опорными точками
- Знак f(T) → класс; **модуль** f(T) → расстояние до прямой

Альтернатива k-NN: МНК (метод наим. квадратов) — 2 балла

Найти прямую $y = a + bx$, минимизирующую сумму квадратов отклонений от точек (x_i, y_i).

$$b = \frac{n \cdot \sum x_i y_i - \sum x_i \cdot \sum y_i}{n \cdot \sum x_i^2 - \left(\sum x_i\right)^2}$$

$$a = \frac{\sum y_i - b \cdot \sum x_i}{n}$$

Алгоритм: строишь таблицу со столбцами $x_i, y_i, x_i y_i, x_i^2$, считаешь суммы, подставляешь.

Пример: точки (1,2), (2,3), (3,5), (4,6). n=4.

	x_i	y_i	$x_i y_i$	x_i^2
1	1	2	2	1
2	2	3	6	4
3	3	5	15	9
4	4	6	24	16
Σ	**10**	**16**	**47**	**30**

- $b = \frac{4 \cdot 47 - 10 \cdot 16}{4 \cdot 30 - 10^2} = \frac{188 - 160}{120 - 100} = \frac{28}{20} = \mathbf{1.4}$
- $a = \frac{16 - 1.4 \cdot 10}{4} = \frac{2}{4} = \mathbf{0.5}$

Ответ: $y = 0.5 + 1.4x$

⚠ **Подвохи:**
- $\sum x_i^2 \neq \left(\sum x_i\right)^2$ — это разные числа, не путать!
- Всегда строй таблицу — без неё легко промахнуться
- Для классификации через МНК: точки выше прямой → один класс, ниже → другой

Задача 1: Трассировка C++ (макросы + рекурсия) — 3 балла

1. **Раскрыть макросы:** `PUT(a)` = вывод, `GET(a)` = ввод, `SET(a,b)` = `*(a)=b`
2. Определить: PUT стоит **до** рекурсии (печать при **спуске**) или **после** (при **подъёме**)
3. Трассировать, явно записывая стек вызовов
4. Записывать вывод в порядке выполнения PUT

f — PUT до рекурсии:

```
void f(int* a, int n) {
  if (n<=0) return;
  PUT(a[0]);    // ← сначала
  g(a+1, n-1); // ← потом
}
```

→ печатает при **спуске**

g — PUT после рекурсии:

```
void g(int* a, int n) {
  if (n<=0) return;
  f(a+1, n-1); // ← сначала
  PUT(a[0]);   // ← потом
}
```

→ печатает при **подъёме**

Таблица трюков:

Конструкция	Что делает	Пример
`arr + i`	указатель на arr[i]	`arr+1` → начиная со 2-го эл.
`*(arr+i)`	то же что arr[i]	`*(arr+2)` = arr[2]
`x << 1`	x × 2	`5 << 1` = 10
`x >> 1`	x ÷ 2 (целое)	`7 >> 1` = 3
`x & 1`	чётность (0/1)	`6 & 1` = 0, `7 & 1` = 1
`i++`	вернуть i, потом +1	`PUT(i++)` печатает старое
`++i`	сначала +1, вернуть	`PUT(++i)` печатает новое
`static int x=0`	инициализация 1 раз	x сохраняется между вызовами
`int& x`	передача по ссылке	изменения видны снаружи

⚠ **Подвохи:**
- `static` переменная — **не сбрасывается** при каждом вызове функции
- `arr+1` не меняет массив, просто передаёт сдвинутый указатель
- После рекурсии стек **раскручивается** — PUT после рекурсии выполнится!
- Если несколько функций взаимно рекурсивны — явно писать стек вызовов

Задача 8: Электронная подпись RSA — 2 балла

1. $N = P \cdot Q, \quad \varphi(N) = (P-1)(Q-1)$
2. Найти C — расширенный алгоритм Евклида (таблица ниже)
3. Подпись: $s = H(m)^D \bmod N$ — быстрое возведение в степень
4. **Проверка:** $s^C \bmod N = H(m)$ — если нет, где-то ошибка

Расширенный алгоритм Евклида — найти C (обратный к D по mod $\varphi(N)$):

i	x (остаток)	y (частное)	z (коэф)
0	$\varphi(N)$	—	0
1	D	—	1
≥2	$x_{i-2} \bmod x_{i-1}$	$x_{i-2} \div x_{i-1}$	$z_{i-2} - y_i \cdot z_{i-1}$

Когда **x = 1** — стоп, C = z этой строки. Если **C < 0**: $C = C + \varphi(N)$

Быстрое возведение в степень $H(m)^D \bmod N$:
1. D перевести в двоичное (правый бит = 2^0, следующий = 2^1 и т.д.)
2. Строить таблицу: каждая строка = **(предыдущий mod N)² mod N**

степень	вычисление	mod N	бит D
$H(m)^1$	$H(m)$	…	бит 2^0
$H(m)^2$	(mod N выше)² mod N	…	бит 2^1
$H(m)^4$	(mod N выше)² mod N	…	бит 2^2
$H(m)^8$	(mod N выше)² mod N	…	бит 2^3

1. Взять строки где **бит = 1**, перемножить их «mod N» — после каждого умножения → mod N

Пример: P=5, Q=11, D=13, H(m)=6 — $N = 55$, $\varphi(55) = 4 \cdot 10 = 40$, $13 = 1101_2$

Евклид:

i	x	y	z
0	40	—	0
1	13	—	1
2	1	3	−3

$C = -3 + 40 = \mathbf{37}$

Пров.: $13 \cdot 37 = 481 = 12 \cdot 40 + 1$ ✓

Степени ($6^{13} \bmod 55$, $13 = \mathbf{1101}_2$):

ст.	вычисл.	mod 55	бит
6^1	6	**6**	1 ✓
6^2	$6^2 = 36$	36	0
6^4	$36^2 = 1296$	**31**	1 ✓
6^8	$31^2 = 961$	**26**	1 ✓

$s = 26 \cdot 31 \cdot 6 \bmod 55$

$26 \cdot 31 = 806$, $806 \bmod 55 = 36$

$36 \cdot 6 = 216$, $s = \mathbf{216 \bmod 55 = 51}$

Пров.: $51^{37} \bmod 55 = 6$ ✓

⚠ **Подвохи:**
- Строка 0 = $\varphi(N)$, строка 1 = **D** (не N, не φ!)
- Таблица степеней: **квадрат предыдущего mod N** — числа всегда < N, калькулятор справится
- $13 = 1101_2$ → справа налево биты: 1,0,1,1 → берём $6^1, 6^4, 6^8$ (где бит=1)
- При умножении шаг за шагом → **mod N после каждого умножения**

Задача 7: Шифр Шамира — 2 балла

1. D_a = обратный к C_a по mod **(P–1)** — Евклид
2. D_b = обратный к C_b по mod **(P–1)** — Евклид
3. $x_1 = m^{C_a} \bmod P$ — А шлёт Б
4. $x_2 = x_1^{C_b} \bmod P$ — Б шлёт А
5. $x_3 = x_2^{D_a} \bmod P$ — А шлёт Б
6. $x_4 = x_3^{D_b} \bmod P = m$ — Б получает

Отличия от RSA: модуль в Евклиде = **P–1** (не φ(N)); в возведении в степень = **P** (не N). Таблицы те же.

Пример: P=23, C_a=5, C_b=7, m=8. P–1 = **22**

Da: $(5 \cdot D_a) \bmod 22 = 1$

i	x	y	z
0	22	—	0
1	5	—	1
2	2	4	−4
3	1	2	9

$D_a = 9$. Пров.: $5 \cdot 9 = 45 = 2 \cdot 22 + 1$ ✓

Db: $(7 \cdot D_b) \bmod 22 = 1$

i	x	y	z
0	22	—	0
1	7	—	1
2	1	3	−3

$D_b = -3 + 22 = \mathbf{19}$. Пров.: $7 \cdot 19 = 133 = 6 \cdot 22 + 1$ ✓

Гоняем сообщение (все mod 23):

шаг	формула	вычисление	итог
x_1	$8^5 \bmod 23$	биты $5 = 101$: степени $8^1 = 8$, $8^2 = 18$, $8^4 = 2$. Берём $8 \cdot 2 = 16$	**16**
x_2	$16^7 \bmod 23$	биты $7 = 111$: $16^1 = 16$, $16^2 = 3$, $16^4 = 9$. $16 \cdot 3 = 2$, $2 \cdot 9 = 18$	**18**
x_3	$18^9 \bmod 23$	биты $9 = 1001$: $18^1 = 18$, $18^2 = 2$, $18^4 = 4$, $18^8 = 16$. $18 \cdot 16 = 12$	**12**
x_4	$12^{19} \bmod 23$	биты $19 = 10011$: $12^1 = 12$, $12^2 = 6$, $12^4 = 13$, $12^8 = 8$, $12^{16} = 18$. $12 \cdot 6 = 3$, $3 \cdot 18 = 8$	**8** = m ✓

⚠ **Подвохи:**
- В Евклиде модуль = **P–1**, в возведении в степень = **P** (не путать!)
- В задаче могут дать всё (P, Ca, Cb, Da, Db, m) — тогда Евклид не нужен, сразу к протоколу
- Если дали только Ca и Cb — **считать** Da и Db через Евклид
- Если коэф в Евклиде вышел отрицательный → **прибавить P–1** (не P!)
- Финальная проверка: x_4 обязан совпасть с m. Не совпал — ошибка где-то

Шифрование Эль-Гамаля — 2 балла

Параметры: p (простое), g (первообразный корень mod p), x (закрытый ключ Боба), $y = g^x \bmod p$ (открытый)

1. Алиса берёт случайное k, считает $a = g^k \bmod p$
2. Считает $b = y^k \cdot m \bmod p$
3. Шифротекст: пара **(a, b)**
4. Боб дешифрует: $m = b \cdot (a^x)^{-1} \bmod p$ — обратный по mod **p** через Евклид

Пример: p=23, g=5, x=6, m=10, k=3

Подготовка: $y = 5^6 \bmod 23$. $6 = 110_2$, степени: $5^1 = 5$, $5^2 = 2$, $5^4 = 4$. Перемножаем биты 1,2: $2 \cdot 4 = \boldsymbol{y = 8}$

Шифрование:
- $a = 5^3 \bmod 23 = 125 \bmod 23 = \mathbf{10}$
- $b = 8^3 \cdot 10 \bmod 23$. $8^3 = 512$, $512 \bmod 23 = 6$. $6 \cdot 10 = 60$, $60 \bmod 23 = \mathbf{14}$
- Шифротекст: **(10, 14)**

Дешифрование:
- $a^x = 10^6 \bmod 23$. $10^1 = 10$, $10^2 = 8$, $10^4 = 18$. Биты 1,2: $8 \cdot 18 = 144$, $144 \bmod 23 = \mathbf{6}$
- Обратный к 6 по mod 23 (Евклид):

i	x	y	z
0	23	—	0
1	6	—	1
2	5	3	−3
3	1	1	4

$(a^x)^{-1} = 4$
- $m = 14 \cdot 4 \bmod 23 = 56 \bmod 23 = \mathbf{10}$ ✓

⚠ **Подвохи:**
- Передаётся **пара** (a, b), а не одно число
- В обратном элементе модуль = **p** (не p–1, в отличие от Шамира/RSA)
- k обычно даётся в условии — **не выбираешь сам**
- y = $g^x \bmod p$ можно считать заранее или взять из условия

Подпись Эль-Гамаля — 2 балла

Параметры: p (простое), g (первообр. корень), x (закрытый), $y = g^x \bmod p$ (открытый), H(m) (хэш)

1. Выбрать k, **взаимно простой** с p–1
2. $r = g^k \bmod p$
3. $s = (H(m) - x \cdot r) \cdot k^{-1} \bmod (p-1)$ — обратный к k по mod **(p–1)**!
4. Подпись = пара **(r, s)**
5. **Проверка:** $y^r \cdot r^s \bmod p$ должно равняться $g^{H(m)} \bmod p$

Пример: p=23, g=5, x=6, H(m)=7, k=3

Сначала y = $5^6 \bmod 23 = 8$ (как в шифровании выше)

Подпись:
- $r = 5^3 \bmod 23 = 125 \bmod 23 = \mathbf{10}$
- Обратный к k=3 по mod (p–1) = mod **22** (Евклид):

i	x	y	z
0	22	—	0
1	3	—	1
2	1	7	−7

$k^{-1} = -7 + 22 = 15$. Пров.: $3 \cdot 15 = 45 = 2 \cdot 22 + 1$ ✓
- $s = (7 - 6 \cdot 10) \cdot 15 \bmod 22 = (-53) \cdot 15 \bmod 22$
- $-53 \bmod 22$: $-53 + 3 \cdot 22 = -53 + 66 = 13$
- $s = 13 \cdot 15 \bmod 22 = 195 \bmod 22 = 195 - 8 \cdot 22 = \mathbf{19}$
- Подпись: **(10, 19)**

Проверка: $y^r \cdot r^s = 8^{10} \cdot 10^{19} \bmod 23$, должно = $5^7 \bmod 23$
- $8^{10} \bmod 23$: $8^1 = 8$, $8^2 = 64 \bmod 23 = 18$, $8^4 = 18^2 = 324 \bmod 23 = 2$, $8^8 = 4$. $10 = 1010_2$, биты 1,3: $18 \cdot 4 = 72 \bmod 23 = 3$
- $10^{19} \bmod 23$: $10^1 = 10$, $10^2 = 8$, $10^4 = 18$, $10^8 = 2$, $10^{16} = 4$. $19 = 10011_2$, биты 0,1,4: $10 \cdot 8 \cdot 4$. $10 \cdot 8 = 80 \bmod 23 = 11$, $11 \cdot 4 = 44 \bmod 23 = 21$
- $3 \cdot 21 = 63 \bmod 23 = \mathbf{17}$
- $5^7 \bmod 23$: $5^1 = 5$, $5^2 = 2$, $5^4 = 4$. $7 = 111_2$: $5 \cdot 2 \cdot 4 = 40 \bmod 23 = \mathbf{17}$ ✓

⚠ **Подвохи:**
- Обратный к k ищем по mod **(p–1)** — как в Шамире, **не** по mod p
- $H(m) - x \cdot r$ может быть **отрицательным** — прибавлять (p–1) пока не станет положительным
- Подпись = **пара** (r, s), не одно число

- Финальная проверка: $y^r \cdot r^s \bmod p = g^{H(m)} \bmod p$. Если нет — ошибка
- k должен быть взаимно прост с p−1 (иначе обратного не существует)

Задача 5: Пролог (трассировка предиката) — 2 балла

Что вообще такое Пролог? Язык, где программа = набор **правил**. Правило: «если такой-то шаблон аргументов — сделай такое-то тело». Запустили `p(что-то)` — Пролог ищет правило, шаблон которого подходит, и выполняет его тело.

Синтаксис правила:
`имя(шаблон1, шаблон2, ...) :- тело.`
`:-` читается как «если» (тело справа должно выполниться). Точка в конце обязательна.

Базовый принцип — унификация: когда вызываешь `p([2,5,6], [], L)`, Пролог сверху вниз перебирает правила и пытается **подставить** конкретные значения вместо переменных в шаблоне. Если подходит — берёт это правило.

Главное про списки в Прологе:
- `[]` — пустой список
- `[H|T]` — H = первый элемент (голова), T = всё остальное (хвост, тоже список)
- `[1,2,3]` это то же что `[1|[2,3]]` = `[1|[2|[3]]]` = `[1|[2|[3|[]]]]`
- `_` (подчёркивание) — «что угодно, мне не важно»
- Заглавные буквы (X, L, L1) — **переменные**, могут принимать любое значение
- Строчные буквы / цифры — **конкретные** значения (атомы)

Таблица шаблонов списков (учить!):

Шаблон	Какие списки подходят	Что во что попадёт
`[]`	только пустой	—
`[X]`	**ровно** 1 элемент	X = этот эл.
`[X,Y]`	**ровно** 2 элемента	X=1-й, Y=2-й
`[_,_]`	**ровно** 2 элемента, значения не важны	—
`[X|Y]`	≥ 1 элемента	X=голова, Y=хвост (список)
`[X,Y|Z]`	≥ 2 элементов	X=1-й, Y=2-й, Z=остаток
`[_,_|_]`	≥ 2 элементов, значения не важны	—

Что такое cut `!`: когда Пролог встречает `!` в теле правила, он **фиксирует** выбор этого правила и больше не возвращается перебирать альтернативы. На практике в этих задачах: `!` в конце базы → «база сработала окончательно, дальше не пытайся».

—

Образец билета:
```
p([_,_],L,L):-!.
p([X|L1],L2,L):-p(L1,[X,X|L2],L).
```
Расшифровываем по словам:

Правило 1 (база): `p([_,_],L,L):-!.`
- Шаблон: 1-й аргумент = список **ровно из 2 элементов**, 2-й и 3-й аргументы — **одна и та же** переменная L
- Тело: пусто (только cut)
- **Смысл:** «если 1-й список из 2 эл. — **вернуть** 2-й аргумент как 3-й. Точка.»
- Иначе говоря: `p([a,b], X, Y)` → `Y = X` (Y получает то же что X)

Правило 2 (шаг рекурсии): `p([X|L1],L2,L):-p(L1,[X,X|L2],L).`
- Шаблон: 1-й аргумент = непустой список (X — голова, L1 — хвост), 2-й аргумент — какой-то L2, 3-й — какой-то L
- Тело: вызвать p заново с новыми аргументами:
  - 1-й → L1 (хвост — на 1 короче)
  - 2-й → `[X,X|L2]` — это значит «новый список, где голова это X, потом ещё X, потом старый L2 целиком» (т.е. вставили **две копии X в начало** L2)
  - 3-й → тот же L
- **Смысл:** «взять голову исходного списка, дописать её **дважды в начало** накопителя, и рекурсивно повторить с хвостом»

Полная трассировка `p([2,5,6,3,4],[],L)`:

шаг	текущий вызов	что происходит
1	`p([2,5,6,3,4], [], L)`	Список 5 эл. — правило 1 не подходит (там надо ровно 2). Берём правило 2. Унификация: X=2, L1=`[5,6,3,4]`, L2=`[]`. Тело → вызов с L1 и `[2,2|[]] = [2,2]`
2	`p([5,6,3,4], [2,2], L)`	4 эл. — правило 2. X=5, L1=`[6,3,4]`, L2=`[2,2]`. Новый L2 = `[5,5|[2,2]] = [5,5,2,2]`
3	`p([6,3,4], [5,5,2,2], L)`	3 эл. — правило 2. X=6, L1=`[3,4]`, L2=`[5,5,2,2]`. Новый L2 = `[6,6,5,5,2,2]`
4	`p([3,4], [6,6,5,5,2,2], L)`	**2 эл.! правило 1 сработало.** Унификация: 2-й=`[6,6,5,5,2,2]`, 3-й=L. Из шаблона `p([_,_],L,L)` следует: L = `[6,6,5,5,2,2]`. Cut — больше не ищем

Финальный ответ: `L = [6,6,5,5,2,2]`

—

Мнемоника «на пальцах»:

Программа **удваивает** каждый элемент входного списка кроме **двух последних** и **переворачивает** порядок (т.к. кладёт в начало накопителя).

Алгоритм для решения за 30 секунд:
1. Возьми список: например `[a, b, c, d, e]`
2. Отбрось **2 последних**: `[a, b, c]`
3. Удвой каждый: `a a b b c c`
4. **Переверни порядок:** `c c b b a a`
5. Это ответ: `[c,c,b,b,a,a]`

Второй пример (потренироваться): `p([1,7,3,2], [], L)`
- Список: `[1, 7, 3, 2]`, отбрасываем 2 последних → остаётся `[1, 7]`
- Удваиваем: `1 1 7 7`, переворачиваем: `7 7 1 1`
- **Ответ:** `L = [7,7,1,1]`

Третий пример: `p([a,b,c,d,e,f], [], L)`
- Отбрасываем 2 последних: `[a,b,c,d]`
- Удваиваем: `a a b b c c d d`, переворачиваем: `d d c c b b a a`
- **Ответ:** `L = [d,d,c,c,b,b,a,a]`

—

Если 2-й аргумент не пустой (например `p([2,5,6,3,4], [99], L)`):
- Накопитель **уже содержит** `[99]`, и мы кладём перед ним
- Результат: `[6,6,5,5,2,2,99]` (старое содержимое в конце)

⚠ **Подвохи:**
- Правила перебираются **сверху вниз**. Если оба подходят — берётся **первое**. Поэтому база (правило 1) пишется **первой**
- `[_,_]` — это **ровно** 2 элемента, не «хотя бы 2». Для «хотя бы 2» — `[_,_|_]`
- В правиле `p(...,L,L)` — **одинаковое имя L** в шаблоне означает «это одна и та же переменная». То есть 2-й и 3-й аргументы должны быть **равны** — отсюда вывод L=L2
- `[X,X|L2]` создаёт **новый** список с двумя X в начале и L2 в конце. Это **не** конкатенация в смысле «склеить два списка целиком»

- В ответе скобки и запятые **обязательны**: пиши `[6,6,5,5,2,2]`, не `6 6 5 5 2 2`
- Если список во входе имеет **менее 2 элементов** (1 или 0), программа **зациклится** / упадёт — но в задачах из билета такого не дают
- При трассировке **писать столбиком**: каждый вызов на отдельной строке с подставленными значениями. Это и есть твой «показ работы» для баллов