

```

algorithm1(A)
1 c = 0
2 ind = -1
3 for i = 0 to n
4     c1 = 0
5     for j = 0 to n
6         if A[i] = A[j]
7             c1 = c1 + 1
8     if c1 > c
9         c = c1
10    ind = i
11 if c > n / 2
12    return A[ind]

algorithm2(A)
c = 1
ind = 0
for i = 1 to n
    if A[ind] = A[i]
        c = c + 1
    else
        c = c - 1
if c = 0
    ind = i
c = 1
return A[ind]

algorithm3 (A)
if n = 1
    return A[0]
c = 1
sort(A)
for i = 1 to n
    if A[i - 1] = A[i]
        c = c + 1
    else
        if c > n / 2
            return A[i - 1]
c = 1

```

1. a. Утверждение неверно, потому что 3 алгоритма решают разные задачи.

Несмотря на то что все 3 алгоритма могут отыскать один и тот же.

Первый алгоритм ищет самое частотное значение в массиве (точнее отыскивает количество одинаковых значений "=".) в данном случае, если количество одинаковых значений будет больше половины общего количества элементов в массиве, то это значение является искомым.

Второй алгоритм ищет самое частотное значение в массиве, если количество одинаковых значений будет больше половины общего количества элементов в массиве, то это значение является искомым.

Второй алгоритм выделяет элемент с индексом ind , с которым будем сравнивать некоторые элементы, которые после него

на подмассиве $A[ind+1 \dots i]$, где $i \in \{1, (n-1)\}$, а также проверяет разность с каждым из элементов (ближайшим $A[ind]$, различным $A[ind]$) и каждым из элементов, не равных $A[ind]$ на подмассиве $A[ind \dots i]$. На каждом шаге из i удаляется единица, если $A[ind] == A[i]$, и увеличивается на 1 иначе. В конце каждого шага проверяется, равна ли разность с нулю. Если равна (такое можно придумать, если на некотором шаге i , т.е. $A[ind] \neq A[i]$), то $ind := i$, иначе менять алгоритм будем сравнивать элементы между $A[i]$, а с ближайшим до i (на некотором подмассиве $A[ind \dots i] = A[ind \dots ind]$ только один элемент $A[ind]$ равен $A[ind]$) и всех элементов не равных $A[ind]$.

Докажем, что если в A есть элемент max , то ему (ближайшему снизу) равно либо $\left\lfloor \frac{n}{2} \right\rfloor$ элементов, либо алгоритм вернет каким-либо из этих подрядко равных элементов.

Дадим исходный фрагмент работы m , бывшую (включая её самую) рабочую $\geq \lfloor \frac{n}{2} \rfloor$ элементов на входе \Rightarrow бывшая рабочая $\geq \lfloor \frac{n}{2} \rfloor + 1$ элементов и не рабочая $\leq n - \lfloor \frac{n}{2} \rfloor - 1$ элементов.

Если $A[0] \neq m$, то изначально алгоритм находит подогнавшую разность элементов, рабочих $A[0]$ и не рабочих $A[\ell_0]$. П.к. $\geq \lfloor \frac{n}{2} \rfloor$ элементов рабочие m , то $\leq \lfloor \frac{n}{2} \rfloor$ элементов рабочих $A[0] \Rightarrow$ на каком-то интервале разности с множеством рабочих Q (п.к. какое-то количество, рабочих m , сплошное множество количества элементов, рабочих $A[0]$), а элементы, не равные ни $A[0]$, ни m (если есть), будут только увеличивать эту разность) (с несмежем отрицательными, п.к. с изначальными и новой какого сдвигивающим рабочи i , а увеличивающим на каком-то интервале на ± 1 , с последующей проверкой на рабочесть).

При этом из-за сдвигов рабочий Q , т.е. на рассматриваемые портфелевые $A[\text{ind}_{-i}]$, $\text{ind} \neq i$ п.к. $A[\text{ind}] \neq A[i]$, рабочая половина элементов (помимо двух крайних граничных) рабочая $A[\text{ind}]$, а половина не рабочая $A[\text{ind}] \Rightarrow$ если в рассматриваемом портфеле есть k элементов рабочих $A[\text{ind}]$,

$m_0 \leq k$ элементов равны $m \Rightarrow$ когда при $c=0$
 в ind записано i , и в супортике $a[i..n-1]$,
 который будем рассматривать на следующих
 интервалах, будем на $\geq k$ первые элементы,
 ке равны m , и $k_0 \leq k$ первые элементы,
 равны $m \Rightarrow m_0$. Но утверждение $\text{ind} \in a[\text{ind}..n-1]$
 дальше находитъ элементы были равны m , то
 и в $a[i..n-1]$ дальше находятся элементы
 будем равно $m \Rightarrow$ ка какое-то количество
 элементов $a[\text{ind}..n-1]$ является
 равно m . С какими обновлением ind
 супортике $a[\text{ind}..n-1]$ утверждение \Rightarrow
 ка какое-то количество $A[\text{ind}]$ станет
 равно m , или в итоге останутся
 мало элементы, равные m (если некоторый
 супортике массив состоит только из элементов $=m$),
 но тогда ind станет указывать ка
 некоторую $k_0 \Rightarrow A[\text{ind}] = m$. Если утверждение
 $A[0] = m$ не сикогда не станет равным

$(m \leq k$ как-то элементов, равных $A[0]$, $\geq \lfloor \frac{n}{2} \rfloor + 1$, а
 как-то элементов, не равных $A[0]$, $\leq \lfloor \frac{n}{2} \rfloor \Rightarrow$
 \Rightarrow ind никогда не обновится и программа
 указывает ка $A[0] = m$ неизвестно сколько
 разомъ учитъ. Учите, если $A[0] \neq m$,

във всички случаи разгледани
доказано, че $A[\text{ind}] \neq m, m$

На какъв-то начин i идентичният
макар и, че $A[i] = m$. Тогава обновленето
идентичните $A[\text{ind}]$ може становището как
различни, така и различни m, k_0, m, k . Но тогава
коадресата $A[\text{ind}] \neq m$ (коадреса $A[\text{ind}] = m, m$
се не завръща чука $A[\text{ind}] = m$)



Най-доброто сортиране на масив и
да все букил прогодим по два

елемента, на коадреси че са идентични
и обновяват със себе си, като
между тях елементи разбиващи
и са разпределени, сколько различни елемента
това наредих по зони, и, една зона
макар подразделя също $\left\lfloor \frac{n}{2} \right\rfloor$,
във връзка с елементи че са подразделяни
чака със същите зони

Така също, 3-ти алгоритъм сортира
във връзка с елементи и последващите

представим его в виде суммы, состоящие
из разных элементов, возбраняя
первое набором (если есть). Однако
алгоритм не проверяет значение с
исле боксера из условия, и,
следовательно, если исходный
нормацав $\text{Lemn} > \left\lfloor \frac{n}{2} \right\rfloor$, состоящие
из разных элементов, окажется на
сумме, то алгоритм не рассмотрит ее.
Заметим, что если 1 из алгоритма
берут из элемента, то их сумма
будет однократной, т.к. в А каждому
элементу, разные возбраняются, $> \left\lfloor \frac{n}{2} \right\rfloor$
следовательно, в А в притупе не может
быть две одинаковые суммы
 $\text{Lemn} > \left\lfloor \frac{n}{2} \right\rfloor$, состоящих из попарно
разных элементов

b. Поясните $A := \{1, 2, 2\}; n = 3$

Первый алгоритм

i	начало итерации	конец итерации
0	$c = 0, c1 = 0, ind = -1, A[i] = 1$	$c1 = 1 > c \Rightarrow$ $\Rightarrow c := c1 = 1, ind := i = 0$
1	$c = 1, c1 = 0, ind = 0, A[i] = 2$	$c1 = 2 > c \Rightarrow$ $\Rightarrow c := c1 = 2, ind := i = 1$
2	$c = 2, c1 = 0, ind = 1, A[i] = 2$	$c1 = 2 \not> c$

Таблица 1: $c = 2 > \lfloor \frac{n}{2} \rfloor = 1 \Rightarrow$ алгоритм вернёт $A[ind] = 2$

Второй алгоритм

i	начало итерации	конец итерации
1	$c = 1, ind = 0$ $A[ind] = 1, A[i] = 2$	$c = 0 \Rightarrow ind := i = 1, c := 1$
2	$c = 1, ind = 1$ $A[ind] = 2, A[i] = 2$	$c = 2 \neq 0$

Таблица 2: алгоритм вернёт $A[ind] = 2$

Третий алгоритм

i	начало итерации	конец итерации
1	$c = 1, A[i - 1] = 1, A[i] = 2$	$A[i - 1] \neq A[i] \wedge c > \lfloor \frac{n}{2} \rfloor = 1 \Rightarrow$ $\Rightarrow c := 1$
2	$c = 1, A[i - 1] = 2, A[i] = 2$	$A[i - 1] = A[i] \Rightarrow c := c + 1 = 2$

Таблица 3: алгоритм не проверит значение с после завершения цикла по i и ничего не вернёт

1 и 2 алгоритмы вернут 2, 3 алгоритм ничего не вернёт.

Пусть $A := \{1, 1, 2\}$; $n = 3$

Первый алгоритм

i	начало итерации	конец итерации
0	$c = 0, c1 = 0, ind = -1, A[i] = 1$	$c1 = 2 > c \Rightarrow$ $\Rightarrow c := c1 = 2, ind := i = 0$
1	$c = 2, c1 = 0, ind = 0, A[i] = 1$	$c1 = 2 \not> c$
2	$c = 2, c1 = 0, ind = 0, A[i] = 2$	$c1 = 1 \not> c$

Таблица 1: $c = 2 > \lfloor \frac{n}{2} \rfloor = 1 \Rightarrow$ алгоритм вернёт $A[ind] = 1$

Второй алгоритм

i	начало итерации	конец итерации
1	$c = 1, ind = 0$ $A[ind] = 1, A[i] = 1$	$c = 2 \neq 0$
2	$c = 2, ind = 0$ $A[ind] = 1, A[i] = 2$	$c = 1 \neq 0$

Таблица 2: алгоритм вернёт $A[ind] = 1$

Третий алгоритм

i	начало итерации	конец итерации
1	$c = 1, A[i - 1] = 1, A[i] = 1$	$A[i - 1] = A[i] \Rightarrow c := c + 1 = 2$
2	$c = 2, A[i - 1] = 1, A[i] = 2$	$A[i - 1] \neq A[i] \wedge c > \lfloor \frac{n}{2} \rfloor = 1 \Rightarrow$ $\Rightarrow return A[i - 1] = A[1] = 1$

Таблица 3: алгоритм вернёт $A[1] = 1$

Все 3 алгоритма вернут 1.

2. В 1 алгоритме выполняется

внешний цикл по i от 0 до n ($0 \leq i \leq n$)

внутри которого выполняется

внутренний цикл по j от 0 до n ($0 \leq j \leq n$),

Быстро которого выполняемое операции, предующие комбинации отнесим к вклад на вычитание. Тогда быстренно укажем на выполнение операции, предующие комбинации отнесим к вкладу на вычитание. Таким образом, $T_{A_1}(n) = O(n^2)$.

Во 2 алгоритме в узле i он делает n ($0 \leq i < n$) выполнение операции, предующие комбинации отнесим к вкладу на вычитание. Таким образом, $T_{A_2}(n) = O(n)$.

В 3 алгоритме вызывается алгоритм сортировки массива A , после которого выполняется узел i ($0 \leq i < n$), на каждом израции которого выполняется операции, предующие комбинации отнесим к вкладу на вычитание, Таким образом,

$$T_{A_3}(n) = O(\text{sort}(n)) + O(n) = O(\max(O(\text{sort}(n)), O(n))),$$

$O(\text{sort}(n))$ — асимптотически верхняя граница временной сложности сортировки $\text{sort}(A)$.

Н.к. алгоритм любой сортировки делает однотактные комбинации не более 1 раз (н.е. обратимся $A[i] \forall i = \overline{1, (n-1)}$), то $O(\text{sort}(n)) \geq O(n) \Rightarrow$

$$\Rightarrow T_{A_3}(n) = O(\text{sort}(n))$$

Ошибки: $T_{A_1}(n) = O(n^2)$; $T_{A_2}(n) = O(n)$; $T_{A_3}(n) = O(\text{sort}(n))$, где

$O(\text{sort}(n))$ – асимптотический верхний граничный временный сложность алгоритма сортировки sort массива А длиной n .

3. Этот же алгоритм решает не задачу, что и первый, необходимо добавить проверку, что элемент $A[i:m]$ содержит все $\lfloor \frac{n}{2} \rfloor$ раз в масиве (то есть алгоритм всегда найдёт элемент, который (если его) равно более $\lfloor \frac{n}{2} \rfloor$ элементов, или такой элемент не есть, показано в конце пункта) на страницах [2;5] этого района).

Эту проверку можно сделать за $O(n)$, то есть изложим верхнюю границу $T_{A_2}(n)$.

Этот же алгоритм решает не задачу, что и первый, необходимо добавить проверку после выполнения цикла № i : если $c > \lfloor \frac{n}{2} \rfloor$, то алгоритм должен вернуть $A[1:n-1]$ (корректиность этого показана в примере 1. пункте а) на страницах [5;6] этого района).

Первый и третий алгоритмы не пригодны
из-за отсутствия возвращаемого значения.

III. Алгоритмы приведены на языке C++
(но крайней мере, по условию), то есть использует
мак `std::optional<T>`, их можно изложить так:

Алгоритм 1

```
1 #include <optional>
2
3 template <typename Container>
4 std::optional<typename Container::value_type> algorithm1(const Container& A) {
5     size_t n = A.size();
6     size_t c = 0;
7     ssize_t ind = -1;
8     for (size_t i = 0; i < n; i++) {
9         size_t c1 = 0;
10        for (size_t j = 0; j < n; j++) {
11            if (A[i] == A[j]) {
12                c1 = c1 + 1;
13            }
14        }
15
16        if (c1 > c) {
17            c = c1;
18            ind = static_cast<ssize_t>(i);
19        }
20    }
21
22    if (c > n / 2) {
23        return A[ind];
24    }
25
26    return std::nullopt;
27 }
```

4. а) Описание работы алгоритма: первое
алгоритмика определяет самое большое количество одинаковых
значений в массиве и возвращает его индекс в виде std::optional.
Если количество одинаковых значений не больше половины общего количества
 \Rightarrow мк. из первого алгоритма $T_{A_1}(n) = c_1 n^2$, то и
второй алгоритмик алгоритм
 $T_{A_1}(n) = c_1 n^2 + c_2 \Rightarrow T_{A_1}(n) = O(n^2) \Rightarrow$
 \Rightarrow увеличение времени выполнения не будет зависеть.

Алгоритм 2

```
1 #include <optional>
2
3 template <typename Container>
4 std::optional<typename Container::value_type> algorithm2(const Container& A) {
5     size_t n = A.size();
6     size_t c = 1;
7     size_t ind = 0;
8     for (size_t i = 1; i < n; i++) {
9         if (A[ind] == A[i]) {
10             c = c + 1;
11         }
12         else {
13             c = c - 1;
14         }
15
16         if (c == 0) {
17             ind = i;
18             c = 1;
19         }
20     }
21
22     size_t A_ind_count = 0;
23     for (size_t i = 0; i < n; i++) {
24         A_ind_count += A[ind] == A[i];
25     }
26
27     if (A_ind_count > (n >> 1)) {
28         return A[ind];
29     }
30
31     return std::nullopt;
32 }
```

4. б) Описание работы алгоритма
алгоритм ищет в массиве максимальное значение в конце подавлено
уникального элемента ($0 \leq i \leq n$), в месте которого
использован оператор, за которым
следует символ // Break, а также в конце подавлено
несколько операторов (строки [27; 31]), которые выполняются
также за комментарийом. Использовано 11 Break
 \Rightarrow в к. п. о. работы алгоритма $T_{A_2}(n) = c_1 n$, но и
нету языковых ограничений алгоритма
 $T_{A_2}(n) = (c_1 + c_2)n + c_3 \Rightarrow T_{A_2}(n) = O(n) \Rightarrow$
 \Rightarrow рабочая времени сложность не учитывается.

Алгоритм 3.

```
1 #include <algorithm>
2 #include <optional>
3
4 template <typename Container>
5 std::optional<typename Container::value_type> algorithm3(Container& A) {
6     size_t n = A.size();
7     if (n == 1) {
8         return A[0];
9     }
10
11    size_t c = 1;
12    std::sort(A.begin(), A.end());
13    for (size_t i = 1; i < n; i++) {
14        if (A[i - 1] == A[i]) {
15            c = c + 1;
16        }
17        else {
18            if (c > n / 2) {
19                return A[i - 1];
20            }
21            c = 1;
22        }
23    }
24
25    if (c > (n >> 1)) {
26        return A[n - 1];
27    }
28
29    return std::nullopt;
30}
31 }
```

4. с) Описание работы алгоритма и временных затрат

алгоритма определяет количество блоков и конечное значение

переменной $c > \lfloor \frac{n}{2} \rfloor$, $\text{return } A[n-1]$ и $\text{return std::nullopt}$.

Эти операции выполняются за константное время

Время выполнения \Rightarrow m_k .
Для каждого блока алгоритма

$$T_{A_3}(n) = O(\text{sort}(n)),$$
 но иforall языков

алгоритма ($m_k O(\text{sort}(n)) \geq O(n)$)

$$T_{A_3}(n) = O(\text{sort}(n)) + c_1 \implies T_{A_3}(n) = O(\text{sort}(n)) \Rightarrow$$

\Rightarrow постоянная временная сложность не учитывается.