Ответы, указания и критерии проверки

Описание критериев:

Критерии.

+1 Означает, что описанная в пункте часть решения стоит 1 балл.

-1 Означает, что за описанную в пункте ошибку снимается 1 балл.

2 По-умолчанию означает максимальный балл (2) за описанный случай. Иные трактовки поясняются.

Преамбула к контрольной

Необоснованные ответы не оцениваются! Если в задаче требуется построение алгоритма, то нужно построить оптимальный алгоритм (за неэффективность снижается оценка), доказать его корректность и оценить время работы (если не в условии не оговорено иное). Во всех задачах модель вычислений атомарная, т. е. арифметические операции стоят $O(1)$.

1(3). Рекуррентные соотношения часто решают методом подстановки: для разрешения $T(n) = 2T(n/2) + Cn$ – можно подставить $Dn \log n$ и подобрать D так, чтобы доказать индуктивный переход. Рассмотрим рекуррентное соотношение $T(n) = 2T(n/2) + 1$. Легко получить, что $T(n) = \Theta(n)$, но если подставить $T(n/2) = Dn/2$ то получим $T(n) = 2Dn/2 + 1 = Dn + 1 \not\leqslant Dn$, т. е. переход доказать не удается. Можно ли это исправить? Решите это рекуррентное соотношение методом подстановки.

Решение. Возьмем в качестве подставляемой функции $Dn + C$. Тогда

$$T(n) = 2\left(D\frac{n}{2} + C\right) + 1 = Dn + 2C + 1$$

Хотим

$$T(n) = Dn + 2C + 1 \leqslant Dn + C$$

При $C \leqslant -1$ это выполняется.
Кроме того, нужно достаточно большое D, чтобы $T(n) : \mathbb{N} \mapsto \mathbb{R}_+$ (например, $D = 2$ подходит).
Тогда подстановкой можем показать, что $T(n) = \Theta(n)$. ■

Критерии.

0 Если задача не решается подстановками функций вида $\Theta(n)$.

+1 Предложена функция вида $Dn + C$.

-0,5 Не доказано «достаточно большое D».

-0,5 $D = 1$.

-0,5 Арифметическая ошибка при подсчете C (либо ошибка в знаке).

-1 Предположение сформулировано в виде $T(n) \leqslant Dn + C$ (оценка только сверху).

-1 Частный случай предыдущего пункта, который часто встречается. В выводе используется $Dn - 1 \leqslant Dn$ и из этого делается вывод о Θ-асимптотике.

2 (2+4). В памяти хранится массив из n чисел со значениями от 1 до $n-1$.

1. Постройте алгоритм, который находит любое из чисел, которое встречается в массиве хотя бы два раза (число n известно).

Решение. **Алгоритм:** Обозначим исходный массив A. Создаем массив B размера $n-1$. Изначально инициализируем $B[i] = 0\ \forall i \in 1, \ldots, n-1$. Проходим по массиву A и для i-го элемента увеличиваем на 1 соответствующую ячейку в B, а именно делаем $B[A[i]] = B[A[i]] + 1$. После этого проходим по массиву B, и если в ячейке под номером i лежит $B[i] \geqslant 2$ возвращаем i.

Корректность: По принципу Дирихле в массиве существует повтор, т.е. какой-то элемент (обозначим его a) встретится минимум два раза. При подсчете мы таким образом увеличим соответствующее $B[a]$ опять же минимум два раза, значит последний поиск выдаст этот элемент a.

Сложность:

По времени: Мы совершаем три линейных прохода по массивам, это $O(n)$

По памяти: Мы выделяем массив на $n-1$ ячейку, это $O(n)$ (по числу ячеек в памяти, для числа битов можно было бы написать $O(n \log n$ из того что ни один счетчик не превысит n). ■

2. Решите ту же задачу, но при условии, что массив доступен только для чтения и разрешено использовать $O(\log n)$ битов памяти.

Решение. **Идея:** Бинпоиск по кол-ву элементов меньших $\left\lfloor \frac{n+1}{2} \right\rfloor$ – если их больше половины то там есть повтор, можно перейти только к этой половине массива значений. По памяти мы будем поддерживать только несколько переменных не превышающих $O(n)$, так что это даст $O(\log n)$ битов памяти.

Алгоритм: Обозначим исходный массив A. Введем переменные $L = 1$, $R = n$, $M = \left\lfloor \frac{L+R}{2} \right\rfloor$. Посчитаем K – количество элементов $A[i]$ массива таких, что $L \leqslant A[i] < M$, если $K > M - L$ то переходим $R := M$ иначе переход к $L := M$, затем опять вычисляем $M = \left\lfloor \frac{L+R}{2} \right\rfloor$ и повторяем пока $R - L > 1$. Если $R - L = 1$ – выводим L.

Корректность: По принципу Дирихле в массиве существует повтор, т.е. какой-то элемент встретится минимум два раза. Рассмотрим первую итерацию алгоритма. Заметим, что т.к. всего элементов $n > R - L$ то выполнено как минимум одно из двух: $K > M - L$ или $n - K > R - M$ (иначе, сложив, получим $n \leqslant R - L$). Алгоритм переходит к половине в которой условие выполнено, при это сохраняется тот факт что в этой половине элементов больше чем возможных значений, таким образом повтор все еще в ней есть.

Сложность:

По времени: Мы совершаем $\log n$ линейных проходов по массиву (для подсчета), это $O(n \log n)$.

По памяти: Мы выделяем константное кол-во переменных, каждая из которых в процессе работы алгоритма не превышает n, значит на запись каждой из них уходит не более $\lceil \log n \rceil$ бит, таким образом получаем $O(\log n)$ бит памяти. ■

Решение. **Идея:** для массива такого вида можно рассмотреть граф с ребрами $i \to a[i]$, и тогда задача можно переформулировать как поиск вершины в которую ведут хотя бы два ребра (это будет соответствовать повтору в массиве). Далее надо использовать то что в вершину с номером n по условию ребер не идет (в массиве числа от 1 до $n-1$), т.е. начав идти из нее

мы сможем найти вершину a в которую ведут два ребра (т.к. рано или поздно мы войдем в какой-либо цикл).

Алгоритм:

1. Найдем цикл в вышеописанном графе. Возьмем переменную $t := n$. В цикле n раз сделаем $t = A[t]$.

2. Найдем длину цикла. Запомним $s := t$ и введем счетчик $P = 0$ для длины цикла. В цикле повторяем $t = A[t]$ и $P := P + 1$ пока опять не получим $s = t$.

3. Положим $t := n$, $h := t$. В цикле P раз сделаем $h = A[h]$. После этого повторяем

$$h := A[h],\ t := A[t]$$

пока $h \neq t$. Когда это выполнено – выводим h.

Корректность:

1. Из любой вершины есть ребро, число вершин ограничено (n), поэтому за n шагов мы хотя бы раз пройдем через вершину которую уже проходили, т.е. мы окажемся в цикле.

2. Длина цикла не превосходит числа вершин, т.е. n, из предыдущего пункта мы находимся в цикле, значит не более чем за n шагов мы вернемся в вершину в которой начали.

3. Пусть длина пути от вершины n до первой вершины (обозначим ее a) цикла который мы рассматривали выше равно L. Указатель t за L шагов дойдет до a. Указатель h суммарно сделал $P + L$ шагов, т.е. дошел до первой вершины цикла и один раз прошел по циклу, т.е. опять же остановится в a. Таким образом алгоритм выведет a. При этом $L + P \leqslant n$.

Сложность:

По времени: На каждом этапе алгоритма мы совершаем не более $O(n)$ операций, таким образом итоговая сложность равна $O(n)$.

По памяти: Аналогично предыдущему решению, мы храним несколько переменных не превышающих n. Получаем $O(\log n)$ бит памяти. ■

Критерии.

- (0,5 балла за первый пункт). Алгоритм с сортировкой за $O(n \log n)$ (QSort, RadixSort).
- (2 балла за первый пункт). Решение за линейное время.
- (-0.5 балла за первый пункт). Проблемы с корректностью (не сказано, что существует пара по принципу Дирихле, алгоритм её находит, потому что...).
- (0.5 балла за второй пункт) Алгоритм, который проверяет каждое число от 1 до $n - 1$ (встречается ли один раз или больше).
- (1 балл за второй пункт). Дано что-то лучше $O(n^2)$ (к примеру модификация CountingSort с использованием только $\log n$ ячеек (и тогда получается $\frac{n}{\log n}$ проходов по массиву).
- (4 балла за второй пункт). Решение за $O(n \log n)$ или лучше по времени. К примеру делаем бинарный поиск по числу элементов, меньших k, начиная с $k = \frac{n}{2}$. Или решение с обходом циклов.

- (от -0.1 до -2.0 баллов за второй пункт). Различные недочеты, не полное/отсутствующее д-во корректности, нет явно выписанной сложности.

- (0.5 за второй пункт). Бинарный поиск с использованием поиска k-ых порядковых статистик (сравнивать номер статистики с ее значением) – есть алгоритм в среднем за $O(n)$ (был в 4-м задании как звездочка) который по идее можно было бы реализовать не рекурсивно и тогда вроде нормально, но в курсе давался только за $O(n)$ в худшем случае в котором создается массив медиан, это $O(n)$ доп. памяти что выходит за допустимые границы.

3 (2). Найдите асимптотику роста функций, полагая, что $T(n) = \Theta(1)$ при малых n:
а) $T(n) = 256T(\frac{n}{16}) + n^2$; **б)** $T(n) = 256T(\frac{n}{16}) + 2n^{1.99}\log n$.

Критерии.

каждый пункт оценивается отдельно, по одному баллу максимум

если применяется мастер-теорема, без ссылки на нее, то пункт оценивается максимум в 0.5 балла

если ответ в пункте неверный, то этот пункт оценивается в 0 баллов

Задача 4 (3). Решите уравнение в целых числах $11x + 17y = C$, где $C \equiv 50^{27} \pmod{101}$. При вычислениях вы обязаны использовать алгоритмы из курса (применение алгоритма должно быть проверяемо по протоколу вычисления).

Указания.

С помощью алгоритма быстрого возведения в степень, находим, что $50^{27} \equiv 53 \pmod{101}$.

Остаётся решить диофантово уравнение $11x + 17y = 53$, которое решаем расширенным алгоритмом Евклида. Общее решение диофантова уравнения:

$$x = -159 + 17t, \;\; y = 106 - 11t, \;\; t \in \mathbb{Z}.$$

Критерии.

- Решение делится на 2 независимые части - нахождение C и решение диофантова уравнения при найденном (возможно даже неправильно) C, за каждую часть ставится 1,5 балла.

- За правильное применение быстрого возведения в степень в первой части ставится +0,5 баллов (просто упоминание существование такого алгоритма не оценивается)

- За правильно полученное $C = 53$ ставится ещё +1 балл

- За правильное применение во второй части алгоритма Евклида или аналогичного ему метода решения диофантова уравнения ставится +0,5 баллов

- За правильное решение диофантова уравнения в виде $x = -159+17t, y = 106-11t, t \in \mathbb{Z}$ (ну или для неверных $C - x = -3 + 17t$, $y = 2C - 11t$, $t \in Z$) ставится ещё +1 балл.

- Всё, что выходит за рамки указанных выше пунктов, оценивается в 0 баллов за соответствующий пункт (краткий список частых ошибок - арифметическая ошибка при вычислении , неверное частное решение диофантова, не указано общее решение решение диофантова, зависящая от параметра часть общего решения имеет одинаковые знаки для x и y, коэффициенты перед t были больше чем нужно в C раз)

- Отдельно хочется упоминуть тот случай, когда студенты пытались решить диофантово уравнение по модулю 101, хотя модуль относился непосредственно к C. Это конечно же их проблема, так как у них была возможность задать вопрос по условию, но тем не менее - им проставлялись баллы за соответствующие пункты, если правильный ответ всё же присутствовал в работе. То есть, если вы сразу перешли от -159 и 106 к -58 и 5, то ответ не засчитывался, так как это не частное решение, а если было прописано сначала $x = -159 + 17t, y = 106 - 11t, t \in \mathbb{Z}$, а потом зачем-то начато вычисление по модулю 101, то ответ засчитывался.

5(3). На вход задачи подаются $2n$ чисел. Постройте алгоритм, который разбивает их на пары так, что максимальная из сумм чисел в парах минимальна.

Решение. **Алгоритм**. Отсортируем массив $A[1 \dots 2n]$, например, сортировкой MergeSort. Из получившегося массива $B[1 \dots 2n]$ составим массив $P[1 \dots n]$ пар вида (b_i, b_{2n-i+1}). Этот массив удовлетворяет условию задачи.

Оценка по времени. Сортировка MergeSort работает за $\Theta(n \log n)$, нахождение элемента в массиве по указателю — $O(1)$, арифметические операции над числами (такие как сумма) работают за $O(1)$ по условию контрольной. Итоговое время работы: $\Theta(n \log n)$.

Корректность алгоритма. При $n = 1$ имеется ровно одно разбиение, поэтому далее будем считать, что $2n \geq 4$.

Рассмотрим *произвольное* разбиение α массива из $2n$ элементов на n пар. Предположим, что max стоит не в паре с min, т.е. имеются пары $(\max, a)$ и $(\min, b)$. Максимальное значение суммы пары чисел в разбиении α обозначим за M_α. Тогда

$$\begin{aligned} \max + a &\leq M_\alpha \\ \min + b &\leq M_\alpha. \end{aligned}$$

Докажем, что $M_\beta \leq M_\alpha$, где разбиение β получено заменой пар $(\max, a)$, $(\min, b)$ на пары $(\max, \min)$, (a, b) (остальные пары остаются без изменений). Действительно,

$$\begin{aligned} \max + \min \leq \max + a &\leq M_\alpha \\ a + b \leq a + \max &\leq M_\alpha. \end{aligned}$$

Но это значит, что для любой пары (a_i^β, b_i^β) из разбиения β выполняется $a_i^\beta + b_i^\beta \leq M_\alpha$. Следовательно $M_\beta \leq M_\alpha$. Это значит, что если сопоставить max и min в пару, то значение максимальной суммы среди пар не увеличится.

Индуктивно применяя это рассуждение к массиву без двух элементов — максимума и минимума, получаем, что для разбиения ω вида $[(a_{(1)}, a_{(2n)}), (a_{(2)}, a_{(2n-1)}), \dots, (a_{(n)}, a_{(n+1)})]$ значение M_ω не больше оптимального. А значит, оно удовлетворяет условию задачи. ■

Критерии.

- 1 балл за правильный алгоритм (сортировка и построение пар $(a_{(i)}, a_{(2n-i+1)})$);

- 2 балла за правильное доказательство корректности;

- -0,5 балла за неверное время работы, худшее, чем $n \log n$, или его отсутствие;
- -2 балла, если отсутствует доказательство корректности;
- -2 балла, если доказательство корректности сводится к "это корректно, т.к. максимум должен стоять с минимумом, иначе будет плохо".
- -2 балла, если не доказывается почему локальные изменения *конкретного* разбиения, полученного алгоритмом, ведут к глобальному оптимуму;
- -2 балла за неверное доказательство по индукции (шаг индукционного перехода не доказан математически);
- -2 балла, если не доказано, что цепь локальных преобразований сходится и приводит к противоречию;
- -2 балла, если решение подразумевает, что пара $(\max, \min)$ даёт максимальную сумму;
- от -2 до -1 балла за ошибки в идейно верном решении, которые не позволяют доказать, что *любое* разбиение не лучше данного алгоритмом;
- от -0,5 до -0,2 за мелкие недочёты: строгие неравенства, неправильные индексы элементов и др.

6(3). В оперативной памяти хранится k односвязных списков, ключи в которых упорядочены по возрастанию; общее число ключей (суммарно во всех списках) равно n. Постройте алгоритм, который возвращает список, состоящий из ключей всех списков, в котором элементы также упорядочены по возрастанию. Сложность алгоритма $O(n \log k)$.

Решение. **Алгоритм 1**

Односвязные списки не пусты, поэтому $k \leqslant n$. Построим кучу с минимальным свойством на парах (a_i, i), где a_i - первый элемент i-го списка, за $O(k)$, используя a_i в качестве ключей. Заведём массив индексов текущих элементов во всех списках, заполнив его нулями. Заведём пустой список для ответа.

Далее в цикле а $O(\log k)$ находим и извлекаем из кучи пару с минимальным ключом и добавляем его в ответ. Затем с такой же асимптотикой добавляем в кучу следующий элемент списка, номер которого равен значению в извлечённой паре и увеличиваем номер текущего элемента для этого списка на 1. Если список пройден полностью, не добавляем в кучу ничего.

Корректность

На каждом проходе цикла, включая первый, в ответ добавляется минимальный элемент из ещё не добавленных. Следовательно, в ответе элементы будут отсортированы по возрастанию.

Сложность

Проходов по циклу будет n, таким образом суммарное время работы составляет $O(n \log k)$.

Алгоритм 2

Модифицируем $Mergesort$: будем попарно сливать списки. Суммарно уменьшение количества списков в два раза требует $O(n)$ операций. Следовательно, за $O(n \log k)$ можно получить один отсортированный список.

■

Критерии.

0 Алгоритм с асимптотикой $\geqslant O(nk)$

+0.5 Модификация $Mergesort$ с неверно оцененной асимптотикой

+0.5 Алгоритм работал бы правильно, если бы была указана структура данных, которая позволяет удалять и добавлять элемент за $O(\log k)$, поддерживая при этом порядок

+0.5 Правильное решение, но с асимптотикой $O(n \log n)$ (сортировка всех элементов)

+3 Правильное решение с асимптотикой $O(n \log k)$

-2 Не оценена асимптотика

7 (3). Известно, что в ходе работы алгоритма двоичное дерево поиска не будет меняться, поэтому было решено хранить его в массиве как почти-полное бинарное дерево (как кучу). Постройте алгоритм, который получив на вход массив $[k_1, k_2, \ldots, k_n]$ ключей строит массив с почти полным бинарным деревом поиска (на этих ключах).

Решение. **Алгоритм**

Сортируем полученный на вход массив, например, с помощью MergeSort за $O(n \log n)$. Далее находим (за $O(\log n)$) степень двойки $h : 2^h - 1 < n \leq 2^{h+1} - 1$. На последние $n - 2^h + 1$ мест пустого массива размера n кладем поочередно элементы с шагом 2 из отсортированного: k_1, k_3, k_5.... Мы заполнили последний уровень нашего дерева. Каждый предыдущий уровень заполняем аналогично. Например, для предпоследнего: на последние пустые 2^h мест кладем элементы из оставшихся в исходном отсортированном с шагом 2, начиная с первого оставшегося: k_2, k_6, k_{10}.... Для следующего уровня берем 2^{h-1} элементов из оставшихся в отсортированном и так далее.

Корректность

При построении выполняется соотношение $k_{2i} \leq k_i \leq k_{2i+1}$, значит, так как на месте i находится элемент, являющийся родителем для своего левого потомка в ячейке $2i$ и правого в ячейке $2i + 1$, то полученное дерево будет являться бинарным деревом поиска. Из того, что полностью заполнены все уровни, а на последнем уровне элементы заполняют его последовательно слева направо следует, что получено почти полное бинарное дерево поиска.

Сложность

Сортировка $O(n \log n)$ + поиск h $O(\log n)$ + заполнение массива $O(n)$. Суммарное время работы $O(n \log n)$

■

Критерии.

+3 Корректный алгоритм с асимптотикой $O(n \log n)$

-1 Арифметические ошибки

-2 Не доказана корректность или не оценена асимптотика

+1 Решение через сортировку массива и выбор в качестве верхушки медианы массива

8 (3). В оперативной памяти хранится массив a, в котором n элементов. i-й элемент массива называется пиком, если $a[i] \geqslant a[i + 1]$ и $a[i - 1] \leqslant a[i]$. При $i = 0$ выполняется только условие $a[0] \geqslant a[1]$, а при $i = n$ только $a[n - 1] \leqslant a[n]$. Постройте алгоритм, который находит пик массива.

Решение. **Алгоритм:**

Начало работы: Проверяем, являются ли пиками первый или последний элемент исходного массива размера n. Выводим пик, если он был найден, иначе переходим к основной части алгоритма.

Пусть l, r - номера текущих левой и правой границ подмассива. Если $r - l \leqslant 5$, за два сравнения находим пик массива среди оставшихся элементов, исключая граничные. В противном случае рассматриваем элемент под номером $\lfloor \frac{l+r}{2} \rfloor$ и его соседей. Если $a_{\lfloor \frac{l+r}{2} \rfloor - 1} \leqslant a_{\lfloor \frac{l+r}{2} \rfloor} \geqslant a_{\lfloor \frac{l+r}{2} \rfloor + 1}$, то $a_{\lfloor \frac{l+r}{2} \rfloor}$ является пиком алгоритма. Выводим элемент в качестве ответа. Иначе, если выполнено $a_{\lfloor \frac{l+r}{2} \rfloor - 1} > a_{\lfloor \frac{l+r}{2} \rfloor}$, то рекурсивно запускаем проверку на подмассиве с границами $l, \lfloor \frac{l+r}{2} \rfloor$. Если не выполнено ни одно из предыдущих условий, значит $a_{\lfloor \frac{l+r}{2} \rfloor} < a_{\lfloor \frac{l+r}{2} \rfloor + 1}$, в этом случае рекурсивно запускаем проверку на подмассиве с границами $\lfloor \frac{l+r}{2} \rfloor, r$.

Сложность по времени: В алгоритме совершается $\log n$ рекурсивных вызовов, при каждом вызове выполняется не более двух сравнений. Таким образом получаем $O(\log n)$ операций.

Корректность: На каждом шаге алгоритм работает с подмассивом, начинающимся с элементов, составляющих возрастающую непрерывную подпоследовательность и заканчивающийся элементами, составляющими убывающую непрерывную подпоследовательность. Соответственно, в каждом рассматриваемом подмассиве существует по крайней мере один пик: это элемент, который заканчивает возрастающую подпоследовательность (можно говорить и про тот, который начинает убывающую). С каждым рекурсивным вызовом число интересующих нас элементов уменьшается вдвое, поэтому алгоритм закончит работу обнаружив пик в подмассиве размером не более 5 элементов. ■

Критерии.

- 0 за проверку условия пика для каждого элемента массива.
- 0.1-0.2 балла за решение, использующее количество операций не превосходящее $n - 1$ (не сложнее поиска максимума)
- -1 при отсутствиии доказательства корректности
- -0.1 при отсутствии оценки сложности алгоритма
- -0.5 за некорректное описание выбора подмассива для выполнения рекурсивного вызова

9 (5). На олимпиаде по знанию алгоритмов сортировок было n (очень большое число) участников. Известно, что некоторый балл t набрало больше чем $\lceil \frac{n}{5} \rceil$ участников. Предложите эффективный алгоритм, который находит какой балл t получили участники (если подходящих t несколько, то алгоритм может вывести любой). t — неотрицательное целое число, верхних ограничений на балл нет.

Решение. **Алгоритм:** Т. к. количество участников, набравших балл t строго больше $\frac{n}{5}$, в отсортированном по баллам массиве оценка t представляет собой непрерывный "отрезок", который гарантированно пересекает одну из 5 порядковых стастистик с шагом $\frac{n}{5}$. Т. е. достаточно найти все пять статистик ($\frac{n}{5}$, $\frac{2n}{5}$, ..., $\frac{4n}{5}$, n) и для каждого балла (соответствующего статистике) подсчитать количество его вхождений в массив.

Сложность по времени: Подсчет 5 порядковых статистик — $5O(n)$. Подсчет количества вхождений в массив - $5O(n) = O(n)$.

■

Критерии.

+5 Подсчет 5 статистик с шагом $\frac{n}{5}$ за $O(n)$, подсчет повторений в массиве для каждого $O(n)$. Или подсчет 5 статистик с шагом $\frac{n}{10}$ - за $O(n)$ и последовательное попарное сравнение за $O(n)$

+2 Поиск 5 статистик с шагом $\frac{n}{5}$, но неверное предположение о том что две из них обязательно окажутся равны (и это будет ответ).

+1 Сортировка + однопроходный подсчёт (при полном и корректном описании описании подсчета).