1 Рассмотрим каждый из пунктов:

а) Распишем по определению : $\exists N, C > 0 : \forall n \geqslant N$ выполнено :

$$n \leqslant C * n \log n$$

Сокращаем на n и находим $N = 2, C = 1$.

Таким образом, мы нашли такие константы NC, при которых опр-е вып-но, а значит, утрв-е а) ВЕРНО.

б) Воспользуемся известным фактом из матанализа:

$$\lim_{n\to\infty} \frac{\log n}{n^{\varepsilon}} = 0$$

Распишем это по определению предела:

$\forall \varepsilon > 0, \forall C > 0\ \exists N \in \mathbb{N} : \forall n \geqslant N$ выполнено: $\log n < C * n^{\varepsilon}$ (1)

Теперь распишем по определению исходное выражение:

$\exists \varepsilon, N, C > 0 : \forall n \geqslant N$ выполнено $\log n \geqslant C'$ (2)

Таким образом, из (1) следует, что (2) не может быть выполнено, то есть определение не выполняется, а значит, утв-е б) НЕВЕРНО

2 Рассмотрим каждый пункт:

1. **а)** Может. Простейший пример: $f(n) = n * \log n = O(n^2), g(n) = 1 = \Omega(1) = O(n)$

б) Не может, т.к. наилучшая верхняя оценка это $O(n^2)$ (см.пункт 2)

2. Верхняя оценка:

$$h(n) = \frac{f(n)}{g(n)} \leqslant \frac{C_1 * n^2}{C_2} = C * n^2 = O(n^2)$$

Нижняя оценка:

$$h(n) = \frac{f(n)}{g(n)} \geqslant \frac{f(n)}{C * n}$$

Далее заметим, что нижней асимптотической оценки $\Omega(f(n))$ нет, а это значит, что мы всегда можем ограничить ф-ю $f(n)$ снизу ф-ей вида $\frac{1}{n^k}, \forall k$. Теперь докажем, что наилучшей нижней оценки не существует. Пойдем от обратного. Пусть наилучшая нижняя оценка сущестует. Тогда, мы всегда сможем подобрать такую k, что мы сможем ограничить нашу оценку снизу. Это противоречит определению наилучшей нижней оценки, а значит ее нет.

3 Да, эквивалентно. Из опр-я студента следует наше определение, потому что в нашем еще есть ограничения на n, а у него такая константа C $\exists$ вообще $\forall$ n. Теперь нужно доказать, что верно обратное. Распишем наше опр-е и выведем из него опр-е студента:

$$\exists N, C > 0 : \forall n \geqslant N \Rightarrow f(n) \leqslant C * g(n)$$

То есть $\exists$ такие n $\in [1; N)$, что f(n) > C * g(n). Но мы в праве "управлять"константой C, и поэтому можем выбрать ее так, что даже при минимальном значении min функции $g(n)$, она будет больше, чем максимальное значение max функции $f(n)$:

$min = min_{0<n<N} g(n), max = max_{0<n<N} f(n) \Rightarrow C_{new} = \dfrac{max}{min}$. Теперь для константы C_{new} определение выполняется, а значит мы получили эквивалентное определение.

4 Везде в этой задаче я буду называть циклы for в том порядке, в котором они идут с самого внешнего к самому внутреннему сверху вниз!

1. Заметим, что итераторы во внутренних циклах не зависят от внешних итераторов. Это значит, что мы можем посчитать количесво выведенных слов, выраженных непосредственно через n.

2. Итак, количество выведенных слов в третьем цикле в точности равно $[\dfrac{n+1}{2}]$. Эта формула выведена так: количество выполненных циклов увеличивается на 1 при увеличении n на 2 (на кадом нечетном числе)

3. Количество выведенных слов в четвертом цикле в точности равно $\lceil \log_2 n \rceil$. Эта формула выведена так: количество выполненных циклов увеличивается на 1 при n = $2^k + 1$, причем при n = 1 кол-во = 0, при n = 2 кол-во = 1.

4. Теперь заметим, что количество итераций внешнего цикла ровно такое же ($\lceil \log_2 n \rceil$), как и у четвертогг for.

5. Заметим, что количество итераций второго оператора for при фиксированном bound, равно в точности $2^{bound-1}$

6. Теперь посчитаем, сколько раз посчитаются внутренние циклы:

$$\sum_{k=1}^{\lceil \log_2 n \rceil} 2^{k-1}$$

7. Итого,

$$g(n) = \sum_{k=1}^{\lceil \log_2 n \rceil} 2^{k-1} * ([\frac{n+1}{2}] + \lceil \log_2 n \rceil) = ([\frac{n+1}{2}] + \lceil \log_2 n \rceil) * \sum_{k=1}^{\lceil \log_2 n \rceil} 2^{k-1} =$$

$$= ([\frac{n+1}{2}] + \lceil \log_2 n \rceil) * (2^{\lceil \log_2 n \rceil} - 1) = \Theta(n * (n + log_2 n))$$

5 (Разобрана).

6 Выполним 3 пункта описания алгоритма:

1. (Сам алгоритм) Находим два максимума из первых эл-тов трех массивов, запоминая при этом номера массивов, в кот-х находятся максимумы и минимум. Везде далее я буду называть массив с наименьшим текущим эл-том тот массив, не из которого мы взяли первый и второй максимумы . Затем бежим по массиву с

наименьшим первым эл-том до тех пор, пока либо 1) он не закончится, либо 2) пока не встретится эл-т $\geqslant$ чем второй максимум. При этом каждый раз увеличиваем счетчик count различных элементов. Далее у нас 3 ситуации: а) эл-т равен второму максимуму, б) эл-т > второго максимума, в) массив закончился. Рассмотрим каждую из ситуаций:

а) В таком случае, мы делаем count++, затем прыгаем на следующие эл-ты в массиве со вторым максимумом и в массиве с минимумом. Помещаем в переменную max_2 больший из них и дальше просто продолжаем идти по массиву с наименьшим новым эл-том и возвращаемся в начало алгоритма.

б) В этом случае мы сравниваем текущий эл-т из массива с наименьшим эл-том с первым максимумом, и если он больше, то во второй максимум помещаем текущий первый максимум, а в первый максимум - текущий элемент из первого массива. Иначе, помещаем текущий эл-т во второй максимум, а первый максимум оставляем. Далее просто продолжаем идти по массиву с наименьшим текущим эл-том и возвращаемся в начало алгоритма.

в) В этом случае мы идем по массиву со вторым максимумом, пока не встретим эл-т $\geqslant$ чем первый максимум или пока он не закончится. И проделываем процедуру, аналогичную пунктам **а)** и **б)**

2. (Корректность) Таким образом, мы 1 раз пробежимся по всем эл-там нашего

массива, не посчитаем дважды ни один из одинаковых эл-тов благодаря бункту **а)**, а также наш алгоритм будет корректен, потому что каждый из массивов отсортирован по возрастанию и когда мы идем по "наименьшему"массиву, все его элементы до первого превышающего второй максимум гарантированно будут меньше и различны (из условия задачи).

3. (Сложность) T(n) = $O(n)$, потому что мы пробежимся по всем трем массивам

ровно 1 раз выполнив при этом не более n сравнений, где n - общее кол-во эл-тов в массивах. S(n) = $O(1)$, потому что мы будем хранить в памяти лишь ограниченное число переменных (два первых максимума, номера массивов, в которых нах-ся максимумы и минимум, и еще пару необходимых переменных, чье количетсво никак не зависит от n).

7. Выполним 3 пункта описания алгоритма:

1. (Сам алгоритм) Введем две вспомогательные переменные: S_A и S_B - суммы всех

a_i и b_i сответственно. Изначально S, S_A и S_B обнуляем. При считывании первой пары эл-тов увеличиваем S_A и S_B на a_1 и b_1 сответственно. Далее на каждом шаге $S = S + S_A * b_i + S_B * a_i$ и так же увеличиваем S_A и S_B на a_i и b_i сответственно. Итого, у нас в каждый момент времени ответ будет лежать в S.

2. (Корректность) Корректность этого алгоритма проверяется через индукцию

следующим образом: пусть верно для k эл-тов, тогда чтобы получить искомую сумму нам надо прибавить к S_k $a_k * S_B + b_k * S_A$ (легко проверяется)

3. (Сложность) $T(n) = O(n)$, так как мы единожды пробегаемся по всем эл-там и

выполняем $O(n)$ операций сложения. $S(n) = O(1)$, потому что мы используем конечное число переменных при решении задачи, чье количество никак не зависит от n.