Бабушкин А.

1 Обязательная часть

1.

a) $\sum_{k=1}^{n} \frac{1}{k^2} = \Theta(\int_1^n \frac{1}{k^2} dk) = \Theta((-1) \cdot \frac{1}{k}\Big|_1^n) = \Theta(\frac{-1}{n} - \frac{-1}{1}) = \Theta(1 - \frac{1}{n}) = \Theta(1).$ #.

b) $\sum_{k=1}^{n} \frac{1}{k^{\frac{1}{2}}} = \Theta(\int_1^n \frac{1}{k^{\frac{1}{2}}} dk) = \Theta((\frac{1}{2}) \cdot k^{\frac{1}{2}}\Big|_1^n) = \Theta(\frac{n^{\frac{1}{2}}}{2} - \frac{1}{2}) = \Theta(\sqrt{n}).$

2.

a) Программа выполняет $\Theta(\sum i)$ действий. Посмотрим, как изменяется i: $1, 2, 4, ..., k, k \geq \frac{n}{2}$. Сумма всех $i = 2k - 1 \leq 2n - 3 \to \sum i = \Theta(n)$.
b) Программа выполняет $\Theta(\sum \log i)$ действий. $\sum \log i = \log_2 3 \sum \log(i/3) = \log_2 3 \cdot \log((n/3)!) = \Theta(\log_2 3 \cdot ((n/3) \cdot (\log(n/3)))) = \Theta(n \log n)$.

3.

a) Если предположить, что в $a[i], b[i]$ лежат числа $\leq 1e9$, тогда за одну итерацию sum может увеличиться не более чем на $1e18$, а в long long влезает примерно $9e18$, тогда по модулю можно брать не каждый раз, а только когда $sum \geq 8e18$.
b) Есть хвостовая рекурсия – её можно переделать в цикл. А ещё можно не высчитывать каждый раз pow(x, -dep), потому что основание всегда одно и то же, а степень уменьшается на единицу, то есть можно завести переменную cur $= x^{0.5}$ и каждый раз делить её на x. Или умножать на заранее вычисленную величину $1/x$, если в double, как и в int, умножение быстрее деления. А если понять, что делает код, то можно вообще считать за $O(1)$ через формулу суммы геометрической прогресси.
c) cout долгий, можно выводить быстрее. Ещё to_string тоже наверняка долгая, ведь это вызов функции + внутри себя она выделяет память для строки, которую затём вернёт. Можно поддерживать строку cur, где будет лежать текущее число, и руками увеличивать его на 1 каждую итерацию, так мы избежим вызова функции, выделения памяти и ещё, возможно, деления, если оно используется в to_string.

4. Рассмотрим префиксные суммы pr[i]. Так как числа натуральные, они возрастают. Тогда сделаем два указателя: один – i – идёт от 0 до $n - 1$, второй – pn – изначально указывает на 0 и на каждой итерации первого указателя сдвигается вправо, пока pr[i] - pr[pn] $>$ S, затем делает проверку pr[i] - pr[pn] $=$ S. Суммарно оба пройдут не более 2n.

2 Дополнительная часть

1. Заведём хеш-таблицу Q[i], гду Q[i] – позиция, где последний раз встретилась префиксная сумма i. Тогда давайте идти вдоль массиваб поддерживая текущую префиксную сумму cur. Придя в позицию i, мы должны проверить, есть ли в хеш-таблице ключ cur - S, и если есть, то мы нашли отрезок с суммой S, а если

нет, то затем присвоить Q[cur] = i и идти дальше. Очевидно, $O(n)$, т.к. хеш-таблицы работают за $O(1)$.

2. Давайте поддерживать две переменных текущий элемент cur и некое число cn. Изначально cur = a_1, cn = 1. Затем перебираем элементы массива от 2 до n. Если $a_i = cur$, то делаем cn++, иначе cn– и если после этого cn < 0, то cur = a_i и cn = 1. Утверждается, что в итоге в cur будет лежать именно нужный нам элемент. Почему? Давайте считать, что у нас всегда есть множество из cn элементов, равных cur. Когда мы делаем cn–, это означает, что наш текущий не равен cur, мы делаем пару из двух разных элементов (cur и текущего) и отправляем их подальше. Если же мы делаем cn++, то мы просто добавили в множество ещё один элемент cur. Пусть в конце у нас cn = k. Тогда у нас есть (n - k) / 2 пар, "отправленных подальше", в каждой из которых элементы различны. Если cur != самому частому элементу, то все самые частые элементы "отправлены подальше". Но их > n / 2 штук, а пар ≤ n / 2, тогда в одной паре будет обязательно два таких элемента, а так быть не может. Противоречие. Значит, cur и есть самый частый элемент.