1 Эту зададу наиболее эффективным способом можно решить, используя рассмотренный на лекции №5 алгоритм "Radix Sort":

1. Сам алгоритм: наши исходные данные уже разбиты нужным нам образом: n элементов, в каждом из которых k разрядов (в нашем случае разряд == латинский символ) и каждый разряд принадлежит ограниченному множеству $0, ..., 25$ (порядковое значение i-той буквы - порядковое значение буквы 'a' в таблице кодировки). Сортируем "справаналево т.е. от младшего разряда к старшему, используя !обязательно! устойчивый алгоритм. Тогда при сортировке более старших разрядов будет учтен правильный порядок младших. Алгоритмически эффективно в данном случае сортировать массив из i-тых разрядов с помощью алгоритма "Counting Sort расмотренного на лекции №5.

2. Сложность: наш алгоритм <u>линеен</u> по входным данным. Их размер это $n * k$. Почему? Потому что мы k раз сортируем разряды, при этом каждый раз используя сортировку со сложностью $\Theta(n + d)$. Тогда общая сложность следующая: $\Theta(k * (n + d))$ Учитывая, что d константа, и в общем случае она меньше $n * k \Rightarrow$ сортировка линейна.

2 Рассмотрим предложенный мною алгоритм:

1. Сам алгоритм: смотрим значение элемента с позицией $\frac{a+b}{2}$, где a и b - левая и правая границы массива на каком-то шаге. Если он больше левого и правого одновременно $\Rightarrow$ мы нашли ответ, иначе если он только больше левого $\Rightarrow a_{new} = \frac{a+b}{2} b_{new} = b$ (идем в правую половину), аналогично с ситуацией, когда больше только правого соседа. Ищем элемент рекурсивно, постоянно уменьшая диапазон индексов массива, где может храниться искомый элемент $\Rightarrow$ алгоритм вернем нам правильный ответ.

2. Доказательство сложности поиска: на каждом этапе мы разбиваем отрезов поиска на 2, таким образом уменьшая его в практически ровно 2 раза (зависит от того, является ли длина исходного массива четной или нечетной). Тогда, в худшем случае нам понадобится $O(\log_2(n))$ шагов рекурсии.

3. В данной задаче нам необходимо дать верхнюю оценку на сложность поиска фальшивой монеты.

1. Приведем алгоритм. Делим исходную кучу на 3 кучи по $\lceil \frac{n}{3} \rceil$ монет в каждой (если не делится ровно на 3, то третья куча просто будет на 1 или 2 монеты меньше). Сравниваем первую и вторую. Если они одинаковые, значит фальшивая монета точно в третьей нерассмотренной куче, иначе в той, что легче. Итак, на этом шаге (как и на всех последующих) мы точно можем сказать, в какой конкретно куче из 3 находится фальшивая монета. Далее продолжаем дробление куч на 3 по нашему алгоритму. Дойдя до того момента, когда останется сравнить кучу, в которой от 1 до 5 монет (это зависит от исходного количества монет). Далее уже за константу определяем, какая из оставшихся монет фальшивая.

2. Теперь давайте поймем, почему при таком алгоритме сложность будет равна именно $O(\log_3(n)$. Каждый раз мы делим кучку на 3 равные, уменьшая ее объем в 3 раза. Это значит, что когда мы дойдем до последних монет в делении кучек, мы выполним $O(\log_3(n)$ делений и сравнений, соответсвенно.

4 . Теперь нам нужно дать уже не верхнюю оценку, а нижнюю. Давайте докажем, почему нельзя за меньшее количество сравнений найти нужную монету. Воспользуемся "Методом противника". На каждом шаге при разбиении на кучки будем "помещать"фальшивую монету в ту кучу, в которой больше всего монет (это ситуация, когда исходное число монет не является степенью тройки. В случае степени кучки всегда будут равны, и точно придется пройтись в глубину до последней тройки монет за $\log_3(n)$). Тогда, на каждом шаге количество монет будет уменьшаться чуть меньше, чем в 3 раза. Это означает, что для просмотра всех монет нам понадобиться делить каждый раз на 3 кучки, брать самую большую из них и делить уже ее. Так появится константа c. Но для просмотра всех монет нам понадобится опять же пройтись в глубину $\log_3(n)$ раз.

5. Заведем два указателя: на первый и второй массивы соответственно, а также вспомогательную переменную-счетчик $k = 1$. Теперь будем идти по массивам следующим образом: идем по первому массиву, пока по первому указателю элемент меньше, чем по второму, при этом при каждом сдвтге указателя увеличиваем счетчик k. Таким образом двигаемся по массиву до тех пор пока $k == n$. Тогда медиана будет находиться по последнему передвинутому указателю.

1. Сложность этого алгоритма это $O(n)$, потому что мы пройдемся ровно по n элементам суммарно из двух массивов, выполнив при этом $n - 1$ сравнение.

2. Этот алгоритм корректен, потому что массивы отсортированы, и мы, двигаясь по массиву таким образом, идем от самого меньшего элемента объединенного массива (это либо 0-й элемент первого массива, либо 0-й элемент второго) по возрастанию до ровно n-ного порядкового элемента, что и будет медианой объединенного.

6. Все коэффициенты натуральны, а значит с увеличением x увиличивается и $f(x)$. Возьмем $l = 0, r = y$. По алгоритму бинарного поиска будем искать решение между l и r. Таким способом, на каждом шаге мы суживаем область поиска в 2 раза, а значит сложнсть будет $O(\log_2(n))$. За счет арифметических операций на каждом шаге общая сложность домножит на n