## №1

Реализуем сортировку по типу RadexSort. У нас будет n строк длины k по 25 вариантов букв в каждой позиции. Сортировку будем производить по стобцам, содержащим i-тый элемент каждой строки в порядке следования строк, двигаясь от конца к началу строк (т.е. если строки записать в матрицу n×k, то идем по ее столбцам справа налево). Нам понадобится массив длины n - buffer, где временно будем хранить слова в нужном порядке и массив длины 25 - count, где будем подсчитывать число слов с конкретной буквой из рассматриваемого столбца. Опишу действия на i-ом шаге, когда по (i+1) столбцу уже отсортировали. Пройдемся по этому столбцу, подсчитывая для каждой буквы, сколько слов ее содержат в этом столбце, увеличивая на 1 значение в соответствующей ей ячейке массива count. Далее пройдемся по массиву count и i-му элементу поставим в ссответствие сумму значений всех предыдущих ячеек (подсчитывать частичные суммы). Теперь проходясь по нашему столбцу с начала,строку с буквой под номером i поставим на место count[i] в buffer и инкрементируем count[i].
Т.к. для каждого из k столбцов мы реализем линейный алгоритм сортировки (всего 3 прохода длины n), то асимптотика в итоге - $\Theta(nk)$.
Корректность следует из того, что описанный шаг правильно сортирует строки: когда по правому от рассматриваемого столбцу уже сортировали, то при сортировке по текущему столбцу сортировка будет устойчивой - если в текущем столбце есть две одинаковых буквы, то они расположаться в buffer в правильном порядке(строка с предшествующей по алфавитному порядку буквой будет выше,той у которой буква справа встречается в алфавите позже).

## №2

Реализуем идею бинарного поиска. Будем делить рассматриваемый кусок массива пополам и если средний элемент :
1)больше обоих соседей, остановимся и вернем его значение
2)меньше правого, то продолжим то же с элементами справа от среднего (левые забываем)
3)больше правого, то продолжим то же с элементами слева от среднего (правые забываем)
Т.к. за $\log_2 n$ делений пополам массива длины n останется максимум 1 элемент, то он и будет ответом (т.е. больше сравнений не надо).

## №4

Мы можем сравнивать только равное число монет. Пусть оно равно x, тогда остается еще куча из (C-2x) монет, где C - общее текущее число монет. Чтобы алгоритм работал дольше, предположим, что дальше он продолжает работать с большей из 2-х куч x и (n-2x), т.е. с max(x,(n-2x)) монет, но max(x,(n-2x))$\geqslant \frac{C}{3}$. Значит, если $n \geqslant 3^k$, то после первого шага $\lceil \frac{n}{3} \rceil \geqslant 3^{k-1}$ и по индукции нужно хотя бы k итераций, чтоб осталась 1 монета. При этом константа: $3^k < n < 3^{k+1} \Rightarrow k < \log_3 n < k+1 \Rightarrow c = 1$

## №3

Рассмотрим случай –когда n -это степень 3-ки. В этом случае при каждом делении всех монет на три кучи надо сделать всего лишь одно взвешивание - если две выбранные (а берем любые две) кучи равны, то берем третью и продолжаем с ней то же. Если же разные, то берем ту, что легче и продолжаем.

В итоге, у нас останутся 3 монеты и из них одним взвешиванием находим легчайшую. В итоге - $\log_3 n$ взвешиваний. При делении на 2 кучи и взятии той, что легче мы в лучшем случае(даже если n - степень 2-ки) сделаем $\log_2 n$ сравнений, что больше $\log_3 n$. Если делить на 4,5,6 куч, то получим сосответственно, $2\log_4 n = \log_2 n$,$2\log_5 n$ и $2\log_6 n$, что больше, чем $\log_3 n$. Т.е. рассмотренный случай действительно похож на самый лучший (т.е. самый быстрый).Значит, $\log_3 n$ - необходимое число (если сделаем меньше взвешиваний, то не сможем делить до получения 1 нужного элемента - самого легкого - а иначе имея несколько кандидатов, не сможем из них выбрать наилегчайший).

## №5

Решение основывается на следующей идее:
Обратимся к средним элементам массивов $a[n/2] = m1$ и $b[n/2] = m2$(с произволным округлением до целого). Сравним их, тогда,если:
1)m1=m2, то m1 - искомая общая медиана.
2)если m1>m2, то отбросим все элементы из массива a с индексом >[n/2]

и из b с индексом $<$[n/2] у нас останется два отсортированных массивах длины [n/2] каждый (соответственные половинки), а цель все та же - найти в их обьединении медиану (т.к. выкинули слева и справа равное число элементов, то мединана в обьединении этих половинок - та же, что и для начального условия). 3)аналогично,при m1$<$m2 выбросим все слева от m1 и справа от m2.
Таким образом, за $\log_2 n$ шагов (число шагов равно числу сравнений, т.е. асимптотика логарифмическая, т.к. обращениий к элементам - $2\log_2 n$) у нас останется не более 1 элемента, и этот элемент будет медианой, т.к. переход верен.

## №6

Целыми корями многочлена являются делители его свободного члена $a_0 - y$. Т.к. мы решаем уравнение в натуральных числах, то переберем все положительные делители числа $a_0 - y$ от 1 до $min[a_0 - y, \ \lfloor \sqrt[n]{y} \rfloor]$ Если значение многочлена в какой-то точке равно 0, то ответ - да, существует.

## №7

1)Разобьем все монеты на пары, при взвешивании каждой пары откладываем более легкие и продолжаем то же с более тяжелыми. Через n-1 взвешивание останется самая тяжелая монета.
Чтобы найти самую легкую монету мы соберем все монеты, которые "вылетели в первом раунде т.к. только они не тяжелее никакой из остальных, прошедших дальше. Их $\lceil n/2 \rceil$. По аналогии, среди них найдем минимальную за $\lceil n/2 \rceil$-1 взвешивание. В итоге, взвешиваний $\frac{3}{2}n + O(1)$
2)Если сделать меньше указанного числа взвешиваний, то кандидат на самую легкую или самую тяжелую будут сравнен не со всеми монетами, т.е. заключить, что это искомая монета не сможем.

## №6(семинар)

Нет такого алгоритма. Достаточно не проверить один из n битов, и в худшем случае рядом с ним будет стоять 1 справа или 0 слева, т.е. возможно, что искомая пара есть, а наверняка мы не знаем.

## №7(семинар)

Если можно в ответе на мое письмо опишите пункт б) 12 монет и 3 взвешивания (в одной из всех ветвей нужно 4 взвешивания у меня)