1. Высота дерева $h = log_4 n, T(N) = 3T(\frac{n}{4} + C \Rightarrow a = 3, b = 4 \Rightarrow d = log_4 3)$
$f(n) = \Theta(n^{log_4 3})$
$C = O(n^{log_4 3 - \varepsilon}) \Rightarrow T(n) = \Theta(n^{log_4 3})$

2. Пусть сумма всех записанных на доске чисел изначально равна S. Поскольку на каждом шаге мы берем два числа a и b $(a > b)$ и вычитаем из ab, то сумма $S = S - (a - b)$, то есть $S(n)$ – монотонно убывающая функция. Т.к. S монотонно убывает и ограничена снизу (т.к. все числа на доске положительны), то, очевидно, процесс когда-нибудь остановится, т.к. S обязательно достигнет значения S_{min}.

Оставшиеся числа будут равны $d =$НОД$(x_{1n}))$, т.к. разность любых двух чисел кратна d и сами числа кратны d.

3. 1) Складываем два числа – сложность $O(n)$
2) Возводим результат в квадрат – сложнсть $O(m + n) = O(2n) = O(n)$
3) Находим квадраты первого и второго чисел – сложность $O(m)$ и $O(n) = O(n)$
4) Вычитаем из (2) квадраты – сложность $O(m + n) = O(n)$
5) Делим результат на 2 – сложность $O(n)$
В итоге получаем сложность $O(n)$

4. Самый простой способ нахождения НОК через НОД с помощью формулы: НОК$(a, b) = \frac{a \cdot b}{\text{НОД}(a,b)}$.

Для получения НОД существует алгоритм Евклида, который работает за $\Theta(n^2)$ (за счет операции деления с остатком).

Далее нам остается произвести операции умножения и деления, которые также совершаются за $\Theta(n^2)$.

Таким образом, суммарная сложность алгоритма – $\Theta(n^2)$.

5. Еще из школы мы знаем формулу $(a_1 + a_2 + \cdots + a_n)^2 = a_1^2 + a_2^2 + \cdots + a_n^2$ и плюс все попарные произведения.

Запишем алгоритм:
1) Вычисляем сумму всех чисел – $O(n)$
2) Вычисляем сумму квадратов всех чисел – $O(n)$
3) Вычисляем квадрат суммы всех чисел – $O(1)$
4) Вычисляем полуразность (3) и (2) – $O(1)$

Полученный результат будет равен искомой сумме. Таким образом, мы нашли значение необходимой суммы за $O(n)$.

6. В этой задаче, очевидно, стоит использовать *master* теорему.
а) $a = 36, b = 6 \Rightarrow log_b a = 2 \Rightarrow F(n) = \Theta(n^2)$
$f(n) = n^2 = \Theta(n^2)$
В таком случае получаем ответ $T(n) = \Theta(n^2 log n)$
б) $log_b a = 1 \Rightarrow F(n) = \Theta(n)$
$f(n) = n^2 = \Omega(n^{log_b a + \varepsilon})$
В таком случае $T(n) = \Theta(n^2)$
в) $log_b a = 2 \Rightarrow F(n) = \Theta(n)$
$f(n) = \frac{n}{log n} = O(n^{log_b a - \varepsilon})$
В таком случае $T(n) = \Theta(n^2)$

7. По сути нам достаточно просто применить к массиву сортировку слиянием, считая каждую перестановку. (Соответственно если мы переставили сразу несколько элементов, то мы прибавляем к счетчику число этих элементов) $\sum 1kn_i$ (где k - количество перестановок, а n_i – количество элементов которые переставляют в $i-$той перестановке) будет равна искомому количеству инверсий.

8. Воспользуемся мастер-теоремой. Здесь будет 3 случая:

1) Начнем с того, что $aT_1\frac{n}{b} = \Theta(n^{log_b a}) = aT_2\frac{n}{b}$

2) Из (1) и из $f(n) = \Theta(g(n)) \Rightarrow T_1(n) = \Theta(T_2(n))$.

Задачу можно решать считая, что их нет, т.е. считать, что на вход $T(n)$ могут подаваться дробные параметры. Это облегчит задачу. Однако для того, чтобы лучше разобраться в теме, рекомендуем решать задачи с учётом округлений.

9. Найдите Θ-асимптотику рекуррентной последовательности $T(n)$, считая что $T(n)$ ограничено константой при достаточно малых n:

В этой задаче везде используется master-теорема, так что высказывания о том, что я использую ее, я, пожалуй, опущу

а) $T(n) = 3T(\lfloor n/4 \rfloor) + T(\lceil n/6 \rceil) + n$;
$T(n) = \Theta(n^{log_4 3}) + \Theta(n^{log_6 1}) + \Omega(n^{log_4 3+\varepsilon}) = \Theta(f(n)) = \Theta(n)$

б) $T(n) = T(\lfloor \alpha n \rfloor) + T(\lfloor (1-\alpha)n \rfloor) + \Theta(n) \quad (0 < \alpha < 1)$;
В силу симметрии и коммутативности суммы можем считать $\alpha < \frac{1}{2}$
В таком случае длина самой короткой ветви будет равна $log_\alpha n$. Длина самой длиной ветви – $log_{1-\alpha} n$.
Тогда справедливы неравенства $\log_\alpha n \cdot \Theta(n) \leqslant T(n) \leqslant log_{1-\alpha} n \cdot \Theta(n)$, т.е. $T(n) = \Theta(nlogn)$

в) $T(n) = T(\lfloor n/2 \rfloor) + 2 \cdot T(\lfloor n/4 \rfloor) + \Theta(n)$;
Для $T(\frac{n}{2})$ сложность $O(n^{log_2 1})$. Для $2T(\frac{n}{4})$ сложность $O(n^{log_4 2})$.
Т.к. $f(n) = \Theta(n) = \Omega(n^{log_4 2+\varepsilon_1} = \Omega(n^{log_2 1+\varepsilon_2}))$, то $T(n) = \Theta(nlogn)$.

г) $T(n) = 27T(\frac{n}{3}) + \frac{n^3}{\log^2 n}$.
Для $27T(\frac{n}{3})$ имеем $n^{log_3 27} = n^3$.
Т.к. $f(n) = O(n^{3-\varepsilon})$, то $T(n) = \Theta(n^3)$.

10. На вход подаются натуральные числа n, p, $n < p, p$ – простое. Предложите алгоритм, который за $O(n + \log p)$ арифметических операций вычисляет массив длины n (i пробегает значение от 1 до n):

а) `invfac[`i`]` $= (i!)^{-1} \pmod p$;

б*) `inv[`i`]` $= (i)^{-1} \pmod p$.

11*. Оцените трудоемкость рекурсивного алгоритма, разбивающего исходную задачу размера n на n задач размеров $\lceil \frac{n}{2} \rceil$ каждая, используя для этого $\Theta(n)$ операций.

1. Можно считать n степенью двойки.
Здесь мы имеем $T(n) = nT(\frac{n}{2})$.
Высота дерева равна $log_2 n$.
Количество вызовов на $i-$том уровне: $n(\frac{n}{2})^i$

В итоге имеем $n \cdot \sum_{0}^{log_2 n} (\frac{n}{2})^i = \Theta(n \cdot \frac{n^{log_2 n}}{2^{log_2 n}}) = \Theta(n^{log_2 n})$.

2. Решите для произвольного n.

Аналогично предыдущему пункту, но при делении на 2 просто всегда округляем вверх.