3. Рассмотрим алгоритм реализации $f(x) = 2x$ на МТ. В общем случае мы должны идти к правому концу входной последовательности, удалять одну единицу (если она есть), затем идти к левому концу последовательности, делать отступ и писать две единицы. И так до тех пор, пока мы не сотрем всю входную последовательность. Таким образом, один повтор данных действий мы сделаем за $\sim 2n$ тактов. При увеличении входных данных в k раз, один повтор мы будем делать за $\sim 2(kn)$ раз. Отсюда можно заключить, что асимптотика алгоритма линейная.
В случае пк, очевидно, что асимптотика – $o(1)$, поскольку операция умножения выполняется компьютером почти мговенно и почти не зависит от входных данных ($1 * 2$ умножится почти также быстро, как $10000 * 2$ и $1e10 * 2$).

4. Существует множество алогоритмов вычисления функции $f(x) = x!$. Если рассматривать "наивную"итеративную реализацию по определению, то сложность такого алогоритма будет $o(n)$. Функция сложности $g(x) = x$, поскольку чтобы вычислить $x!$, нам нужно сделать x шагов, последовательно умножая $1 * 2 * \cdots * x$.

5. Для подсчета суммы всех элементов стандартным итеративным алгоритмом, мы должны пройтись по всему массиву, т.е. $g(x) = x$, асимптотика алгоритма линейная, т.е. $o(n)$.

6. В этом случае сложность будет $o(n * m)$.

7. В этом случае, как и в предыдущем, сложность будет $o(n * m)$.

8. Представим, что у нас есть матрица размера (n, m), тогда задача сводится к печати всевозможных индексов элементов этой матрицы. Тогда сложность такого алгоритма будет $o(n * m)$.

9. В этом случае, как и в предыдущем, сложность будет $o(n * m)$, где m, n – размеры таблицы.

10. Вытянем всю таблицу в массив. Тогда размер такого массива будет составлять $m * n$. Бинарный поиск в таком массиве будет иметь сложность $o(\log(n * m))$. Можно пойти и по-другому: допустим, мы меняем индексы сразу по двум размерностям, но тогда сложность такого поиска будет $o(log(m) + log(n))$, что в силу тождественности расложения логарифма произведения на сумму логарифмов будет эквивалентно $o(\log(m * n))$.

11. В такой задаче нам должны быть известны два параметра: область поиска корня и точность, с которой мы хотим получить корень. Возьмем абсолютную величину разности границ области поиска и поделим ее на точность. Тогда мы получим количество вариантов для нашего корня, или то самое n. Очевидно, что этот алгоритм – тот же самый бинарный поиск, поэтому и сложность у него будет логарифмическая: $o(\log(n))$.

12. Для начала приведем x в область $[0; 2\pi]$. Для этого будем из x вычитать (прибавлять) 2π, пока он не окажется в этой области. Затем задача сводится к предыдущей: множество возможных корней – это $n = \frac{\pi}{\mathcal{E}}$. А дальше мы можем бинарным поиском искать значение x. Таким образом, сложность данного алгоритма – $o(\log(\frac{\pi}{\mathcal{E}}))$.