A2

FAST_EXPONENT.cpp

```cpp
int fastExponent(int x, int n) {
    int r = 1;
    int p = x;
    int e = n;

    while (e > 0) {
        if (e % 2 != 0) r = r * p;
        p = p * p;
        e = e / 2;
    }

    return r;
}
```

1. Какое точное количество операций умножения требуется выполнить, чтобы вычислить x^n с помощью алгоритма FAST EXPONENT? Всегда ли данный алгоритм лучше наивного способа вычисления?

в цикле мы выполним $log_2 n + 1$ операций возведения p в квадрат и столько же операций деления e пополам и `std::popcount(n)` (количество единиц в бинарной записи числа) умножений `r=r*p`

итого в функции мы выполним

$X(n) = 2\lfloor log_2 n \rfloor + std :: popcount(n) + 5$

не всегда эта функция выгодна, когда у нас маленький $n = 2^k - 1$ выгоднее умножить обычным способом например при $n = 3$ обычным способом получится 2 операции

а функцией $X(3) = 2 + 2 + 5 = 9$ операций

2. Сформулируйте условие P, которое подходит в качестве инварианта цикла while. Представьте достаточное обоснование выбора инварианта. Выполните проверку выполнения найденного инварианта до входа в цикл (INIT), в каждой итерации цикла (MNT), а также при выходе из цикла (TRM).

$$P : p^e r = x^n$$

я выбрал этот инвариант, потому что по нему отчетливо видно, как при изменении p^e меняется r, к тому же он выглядит красиво и лаконично

INT

при входе в массив

$p = x$

$r = 1$

$e = n$

выходин инвариант выполняется:

$x^n = p^e r = x^n$

MNT

у нас известно что цикл выполнялся на предыдущей итерации

есть два варианта предыдущего условия

первый:

$\sqrt{p}^{2e} r = x^n$

тогда инвариант продолжает выполнятся

$p^e r = x^n$

второй вариант

$\sqrt{p}^{2e+1} \frac{r}{\sqrt{p}} = x^n$

тогда

$\sqrt{p}^{2e} r = x^n$

тогда инвариант продолжает выполнятся

$p^e r = x^n$

TRM

в конце у нас получвется что

$r = x^n$, а $e = 0$

тогда

$p^0 r = 1r = x^n$

инвариант выполняется!