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;
}
```

Цикл while будет выполняться, пока $e > 0$. На каждой итерации e становится в 2 раза меньше. Тогда цикл выполнится $\lfloor \log_2 n \rfloor + 1$ раз.

Когда $e \% 2$, выполняется 2 умножения в цикле, иначе только 1.

"Переменная e не делится на 2" можно представить в двоичном виде. Т.е. если в двоичной СС число e имеет n единиц, то при делении на 2 число e не будет иметь остаток 0 n раз. Следовательно, наибольшее число умножений будет, если $e = 2^k - 1$ $k \in \mathbb{N} \cup \{0\}$.

Тогда операций умножения будет $2 \cdot (\lfloor \log_2 n \rfloor + 1) = 2 \lfloor \log_2 n \rfloor + 2$.

Иной способ возведения в степень — умножать число само на себя n раз. Поэтому асимптотика такого алгоритма $O(n)$.

Расчитаем, что наиболее выгодно:

n	1	2	3	4	5	6	7	8	9	10	11	12
$2 + 2\lfloor \log_2 n \rfloor$	2	4	4	6	6	6	6	8	8	8	8	8

То есть при $n > 6$ выгоднее алгоритм быстрого возведения.

Инвариант P: после каждой итерации $x^n = r \cdot p^e$.

INIT: на входе $r = 1$, $p = x$, $e = n \Rightarrow r \cdot p^e = 1 \cdot x^n = x^n$.

MNT: 1) если $e \% 2 \neq 1$: $p \to p^2$, $e \to \frac{e}{2} \Rightarrow r \cdot p^e = r \cdot (p^2)^{\frac{e}{2}} = r p^e = x^n$.

2) если $e \% 2 = 1$: $p \to p^2$, $r \to r \cdot p$, $e \to \left\lfloor \frac{e}{2} \right\rfloor$

$\Rightarrow r \cdot p^e = r \cdot p \cdot p^{2 \cdot \left\lfloor \frac{e}{2} \right\rfloor} = r \cdot p^e = x^n$

TRM: На выходе получим $e = 0$, $r = p^n$, тогда $r \cdot p^e = p^n \cdot p^0 = p^n = x^n$.