Binary Exponentiation

$2^{13} \Rightarrow 2^{1101}$ $\quad (13)_{10} = (1101)_2$

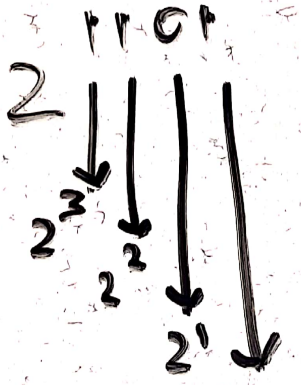
$\Rightarrow 1 \; 1 \; 0 \; 1$

$\Rightarrow 1000 \rightarrow 8 \rightarrow 2^3$

$100 \rightarrow 4 \rightarrow 2^2$

$10 \rightarrow 2 \rightarrow 2^1$

$1 \rightarrow 1 \rightarrow 2^0$



$\Rightarrow 2^{13} = 2^{1(8)+1(4)+0(2)+1(1)}$

$\Rightarrow 2^{8+4+1}$

$\Rightarrow 2^8 \cdot 2^2 \cdot 2^0$

$7^{53} \rightarrow 7^{110101}$

1	1	0	1	0	1
↓	↓	↓	↓	↓	↓
2^5	2^4	2^3	2^2	2^1	2^0
↓	↓	↓	↓	↓	↓
32	16	8	4	1	1

$53 = 1(32) + 1(16) + 0(8) + 1(4) + 0(1) + 1(1)$

$= 32 + 16 + 4 + 1$

$53 = 32 + 16 + 4 + 1$

$7^{53} = 7$

$\Rightarrow 7^{32} \cdot 7^{16} \cdot 7^{4} \cdot 7^{1}$

. . . . . .

$2^{21} \Rightarrow 2$ 10101

1	0	1	0	1
↓	↓	↓	↓	↓
2^4	2^3	2^2	2^1	2^0
↓	↓	↓	↓	↓
16	8	4	2	1

$21 = 1(16) + 0(8) + 1(4) + 0(2) + 1(1)$

$= 16 + 4 + 1$

$2^{21} = 2^{16+4+1} = 2^{16} \cdot 2^4 \cdot 2^1$

⇒ from here There are 2 Approaches.

① Loop ② Recursive.

Loop

Take RES = 1

2^{13} = Base Power

Base = 2 ; Power = 13

⇒ Retriving LSB.

Power & 1 = LSB

Power >>= 1;

↳ Left shift

By ONE.

Loop

$5^{13} \Rightarrow 5^{1101}$

1	1	0	1
↓	↓	↓	↓
2^3	2^2	2^1	2^0
↓	↓	↓	↓
8	4	2	1

$13 = 1(8) + 1(4) + 0(2) + 1(1)$

$5^{13} = 5^{8+4+1}$

$= 5^8 \cdot 5^4 \cdot 5^1$

$5^{13} \Rightarrow 5^8 \cdot 5^4 \cdot 5^2 \cdot 5^1$

↓	↓	↓	↓
1	1	0	1

Loop

Take res = 1

```
while (Power) {
    Power & 1;  → LSB
    Power >>= 1;
}
```

This snippet produces LSB of Power

① if LSB is 1
multiply Base to res
and store it in res

② for Each iteration
Square the Base

Loop

```
res=1
while(Power){
    if(Power &1)
        res = res * Base
    Base = Base * Base;
}
return res;
```

$5^{13} \rightarrow 5^{1101}$

Power	1	1	0	1
Base	390625	625	25	5
Res	1220703125	3125	5	5

remains same

Power &1 = 0

2^{21} $(21)_{10} = (10101)_2$

Power	1	0	1	0	1
Base	65536 256^2	256 16^2	16 $(4)^2$	4 $(2)^2$	2
Res	2097 152	32	32	2	2

→ Res not changed.
Power & 1 = 0

$2^{21} = 2097152$

Constraints

Res and Base should be long long.

max value of long long is

$$2^{63} - 1$$

$$= 9223372036854775807$$

Time Complexity

b^p

Normal Approach : $O(P)$

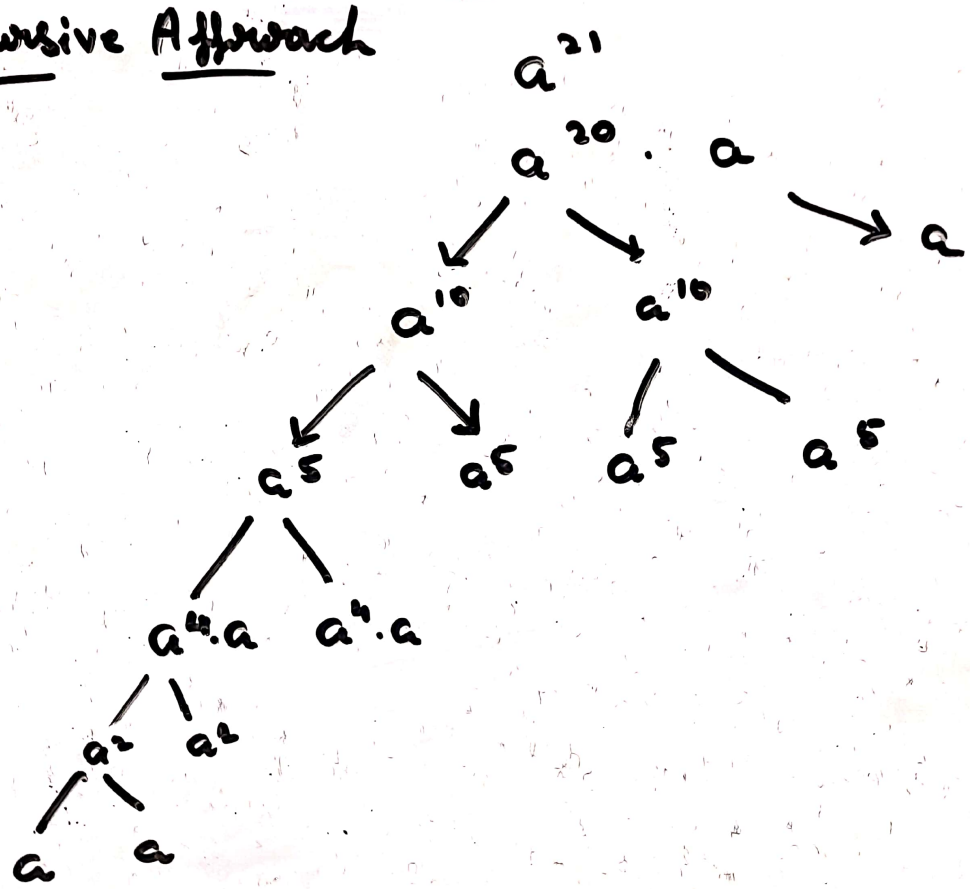
Continuous Multiplication

Binary Exponentiation : $O(\log_2 P)$

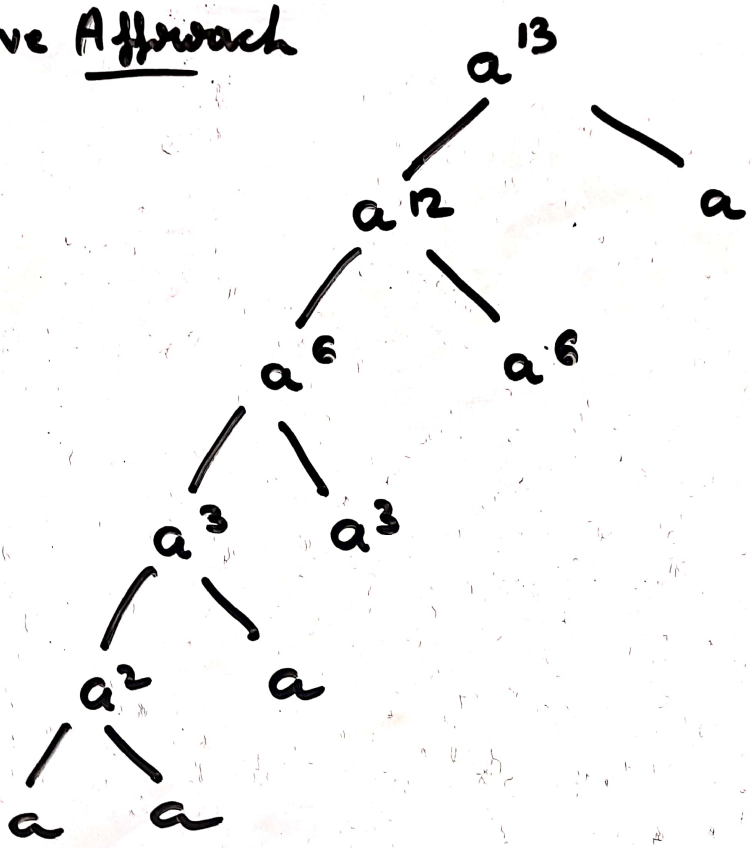
Multiplication takes $O(\log b)$ time

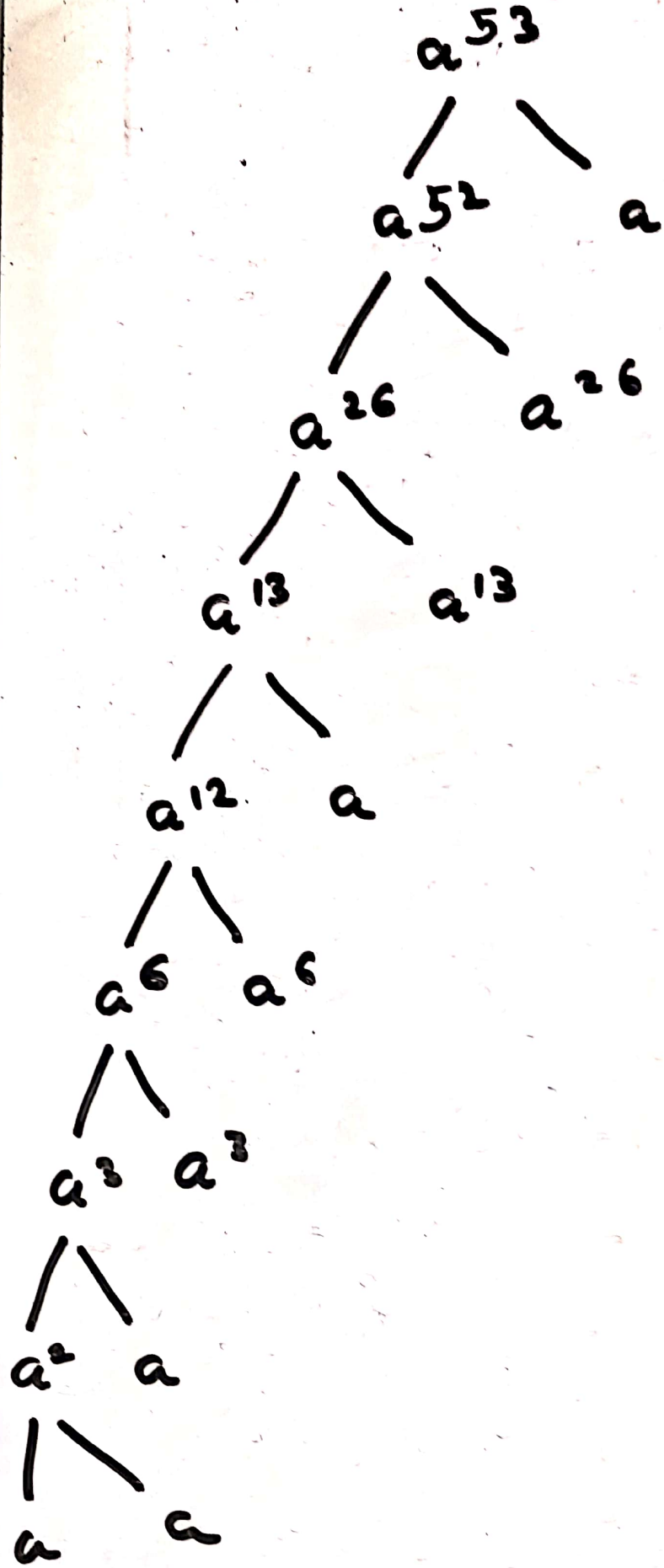
So, TC : $O(\log b \times \log_2 P)$

Recursive Approach



Recursive Approach




a^{53}
a^{52}
a
a^{26}
a^{26}
a^{13}
a^{13}
a^{12}
a
a^{6}
a^{6}
a^{3}
a^{3}
a^{2}
a
a
a

$a^0 \rightarrow 1$

$a^1 \rightarrow a$

$\boxed{a^n}$

① $n \rightarrow$ Even ; $a^n = a^{n/2} \cdot a^{n/2}$

$\Rightarrow a^n = (a^{n/2})^2$

---

② $n \rightarrow$ odd ; $a^n = a^{n-1} \cdot a$

if $n \rightarrow$ odd

$n-1 \rightarrow$ Even

$a^{n-1} \Rightarrow a^{n-1/2} \cdot a^{n-1/2}$

$\Rightarrow \left(a^{\frac{n-1}{2}}\right)^2$

$a^n = \left(a^{n-1/2}\right)^2 \cdot a$

Function long long BE(int b, int p)

b^p

Base Case

```
if (p == 0)
    return 1;
if (p == 1)
    return p;
```

Recursive Case

```
int res = BE(b, p/2)

if b → Even.
    return res * res.

if b → odd (else).
    return res * res * a.
```

NOTE

for a^n

if $n \rightarrow$ Even $\therefore a^n = (a^{n/2})^2$

if $n \rightarrow$ odd $\therefore a^n = (a^{\frac{n-1}{2}})^2 a$

But we took rs = BE(b, P/2)

dt $n \Rightarrow 4$

$n \rightarrow$ Even $\rightarrow$ $\boxed{n/2 = 2}$

dt $n = 7$

$n \rightarrow$ odd

$\left(\frac{n-1}{2}\right) = \left(\frac{7-1}{2}\right) = 6/2 = 3$

$n/2 \Rightarrow 7/2 = 3.5$

Decimal will be truncated.

So $n/2 = 3$ for $n = 7$

CODE b^p

```
long long BE(int b, int p) {
    if (p==0)
        return 1
    if (p==1)
        return p
    long long res = BE(b, p/2);
    if(p & 1)    // odd
        return res * res * a;
    else
        return res * res;
}
```

This can be integrated with DP for better Performance.