**Fast Algorithm:**  
  
result = 1  
If power is odd:  
    result = result\*value  
value = value\*value  
power = power/2  
  
Yeah, I know, above algorithm more looks like a code & was nothing but a bouncer. ;)  
  
  
  
Lets discuss an example of what exactly we are doing here. Same example 2^10.  
  
We know,  
  
    2^10  
=  (2\*2) \* (2\*2) \* (2\*2) \* (2\*2) \* (2\*2)  
=  4^5    // here power has become odd, so lets take out one 4 out of five 4's & multiply it with 'result' & store back in 'result' & hence we are left with four 4's  
  
So result = 4 and we are left with 4^4  
  
4^4  
= (4\*4) \* (4\*4)  
= 16^2 // since power is even no need to mess up our result variable  
  
16^2  
= 256^1 // here power has become odd, so lets take out one 256 & multiply it with 'result' & store back in 'result' hence we are left with zero 256's  
  
  
  
Viola, power became zero. Its the end. Now our result contains what we want.  
  
Following is C/C++ implementation of fast powering algorithm.  
  
Line #8: (power&1) is a better/faster way to check if a number if odd or not  
Line #14: I commented it out, but that too is an alternative of Line #13

|  |  |
| --- | --- |
|  | long long fast\_pow(int a,int n) |
|  | { |
|  | long long result = 1; |
|  | long long power = n; |
|  | long long value = a; |
|  | while(power>0) |
|  | { |
|  | if(power&1) |
|  | {result = result\*value; |
|  | result = result%1000000007;} |
|  | value = value\*value; |
|  | value = value%1000000007; |
|  | power /= 2; |
|  | //power >>= 1; |
|  | } |
|  | return result; |
|  | } |