**Bit Tricks for Competitive Programming**

* **Clear all bits from LSB to ith bit**

mask = ~((1 << i+1 ) - 1);

x &= mask;

**Logic:** To clear all bits from LSB to i-th bit, we have to AND x with mask having LSB to i-th bit 0. To obtain such mask, first left shift 1 i times. Now if we minus 1 from that, all the bits from 0 to i-1 become 1 and remaining bits become 0. Now we can simply take complement of mask to get all first i bits to 0 and remaining to 1.  
Example-  
x = 29 (00011101) and we want to clear LSB to 3rd bit, total 4 bits

mask -> 1 << 4 -> 16(00010000)  
mask -> 16 – 1 -> 15(00001111)  
mask -> ~mask -> 11110000  
x & mask -> 16 (00010000)

#include<bits/stdc++.h>

using namespace std;

int main(){

int n , i ;

cin >> n >> i ;

n &= (~( ( 1 << (i+1) ) -1 ) ) ;

cout << n << endl;

return 0;

}

* **Clearing all bits from MSB to i-th bit**

mask = (1 << i) - 1;

x &= mask;

**Logic:** To clear all bits from MSB to i-th bit, we have to AND x with mask having MSB to i-th bit 0. To obtain such mask, first left shift 1 i times. Now if we minus 1 from that, all the bits from 0 to i-1 become 1 and remaining bits become 0.  
Example-  
x = 215 (11010111) and we want to clear MSB to 4th bit, total 4 bits  
mask -> 1 << 4 -> 16(00010000)  
mask -> 16 – 1 -> 15(00001111)  
x & mask -> 7(00000111)

#include<bits/stdc++.h>

using namespace std;

int main(){

int n , i ;

cin >> n >> i ;

n &= ( ( 1 << (i+1) ) -1 ) ;

cout << n << endl;

return 0;

}

* **Divide by 2**

x >>= 1;

**Logic:** When we do arithmetic right shift, every bit is shifted to right and blank position is substituted with sign bit of number, 0 in case of positive and 1 in case of negative number. Since every bit is a power of 2, with each shift we are reducing the value of each bit by factor of 2 which is equivalent to division of x by 2.  
Example-  
x = 18(00010010)  
x >> 1 = 9 (00001001)

* **Multiplying by 2**

x <<= 1;

**Logic:** When we do arithmetic left shift, every bit is shifted to left and blank position is substituted with 0 . Since every bit is a power of 2, with each shift we are increasing the value of each bit by a factor of 2 which is equivalent to multiplication of x by 2.  
Example-  
x = 18(00010010)  
x << 1 = 36 (00100100)

* **Upper case English alphabet to lower case**

ch |= ' ';

**Logic:** The bit representation of upper case and lower case English alphabets are –

A -> 01000001 a -> 01100001

B -> 01000010 b -> 01100010

C -> 01000011 c -> 01100011

. .

Z -> 01011010 z -> 01111010

As we can see if we set 5th bit of upper case characters, it will be converted into lower case character. We have to prepare a mask having 5th bit 1 and other 0 (00100000). This mask is bit representation of space character (‘ ‘). The character ‘ch’ then ORed with mask.

Example-  
ch = ‘A’ (01000001)  
mask = ‘ ‘ (00100000)  
ch | mask = ‘a’ (01100001)  
Please refer [Case conversion (Lower to Upper and Vice Versa)](https://www.geeksforgeeks.org/case-conversion-lower-upper-vice-versa-string-using-bitwise-operators-cc/) for details.

* **Lower case English alphabet to upper case**

ch &= '\_’ ;

**Logic:** The bit representation of upper case and lower case English alphabets are –

A -> 01000001 a -> 01100001

B -> 01000010 b -> 01100010

C -> 01000011 c -> 01100011

. .

. .

Z -> 01011010 z -> 01111010

As we can see if we clear 5th bit of lower case characters, it will be converted into upper case character. We have to prepare a mask having 5th bit 0 and other 1 (10111111). This mask is bit representation of underscore character (‘\_‘). The character ‘ch’ then AND with mask.  
Example-  
ch = ‘a’ (01100001)  
mask = ‘\_ ‘ (11011111)  
ch & mask = ‘A’ (01000001)  
#include<bits/stdc++.h>

using namespace std;

int main(){

char a ;

cin >> a ;

a |= (1<<5); // Upper to Lower // (1<<5) is equal “ “

a &= (~(1<<5)); // Lower to Upper// (~(1<<5)) is equal “\_“

cout << (char)(a)<<endl;

return 0;

}

* **Count Set Bits :**

int countSetBits(int x){

int count = 0;

while (x){

x &= (x-1);

count++;

}

return count;

}

* **Find log base 2 of 32 bit integer**

int log2(int x)

{

    int res = 0;

    while (x >>= 1)

        res++;

    return res;

}

**Logic:** We right shift x repeatedly until it becomes 0, meanwhile we keep count on the shift operation. This count value is the log2(x).

**9) Checking if given 32 bit integer is power of 2**

int isPowerof2(int x){

    return (x && !(x & x-1));

}

**Logic:** All the power of 2 have only single bit set e.g. 16 (00010000). If we minus 1 from this, all the bits from LSB to set bit get toggled, i.e., 16-1 = 15 (00001111). Now if we AND x with (x-1) and the result is 0 then we can say that x is power of 2 otherwise not. We have to take extra care when x = 0.

Example  
x = 16(000100000)  
x – 1 = 15(00001111)  
x & (x-1) = 0  
so 16 is power of 2