Next Power of 2

Write a function that, for a given no n, finds a number p which is greater than or equal to n and is a power of 2.

```
IP 5
OP 8

IP 17
OP 32

IP 32
OP 32
```

There are plenty of solutions for this. Let us take the example of 17 to explain some of them.

Method 1(Using Log of the number)

```
1.  Calculate Position of set bit in p(next power of 2):
    pos =  ceil(lgn)  (ceiling of log n with base 2)
2.  Now calculate p:
    p   = pow(2, pos)
```

Example

```
Let us try for 17
        pos = 5
        p   = 32
```


Method 2 (By getting the position of only set bit in result)

```
   /* If n is a power of 2 then return n */
   1  If (n & !(n&(n-1))) then return n
   2  Else keep right shifting n until it becomes zero
       and count no of shifts
       a. Initialize: count = 0
       b. While n ! = 0
              n = n>>1
              count = count + 1

    /* Now count has the position of set bit in result */
   3  Return (1 << count)
```

Example:

```
   Let us try for 17
              count = 5
              p     = 32
```

```
unsigned int nextPowerOf2(unsigned int n)
{
  unsigned count = 0;

  /* First n in the below condition is for the case where n is 0*/
  if (n && !(n&(n-1)))
    return n;

  while( n != 0)
  {
    n  >>= 1;
    count += 1;
  }

  return 1<<count;
}

/* Driver program to test above function */
int main()
{
```


Popular Posts

```
  unsigned int n = 0;
  printf("%d", nextPowerOf2(n));

  getchar();
  return 0;
}
```

Method 3(Shift result one by one)

Thanks to coderyogi for suggesting this method . This method is a variation of method 2 where instead of getting count, we shift the result one by one in a loop.

```
unsigned int nextPowerOf2(unsigned int n)
{
    unsigned int p = 1;
    if (n && !(n & (n - 1)))
        return n;

    while (p < n) {
        p <<= 1;
    }
    return p;
}

/* Driver program to test above function */
int main()
{
  unsigned int n = 5;
  printf("%d", nextPowerOf2(n));

  getchar();
  return 0;
}
```

Time Complexity: O(lgn)

Method 4(Customized and Fast)

```
1. Subtract n by 1
   n = n -1


2. Set all bits after the leftmost set bit
```

```
2. Set all bits after the leftmost set bit.

/* Below solution works only if integer is 32 bits */
            n = n | (n >> 1);
            n = n | (n >> 2);
            n = n | (n >> 4);
            n = n | (n >> 8);
            n = n | (n >> 16);
3. Return n + 1
```

Example:

```
Steps 1 & 3 of above algorithm are to handle cases
of power of 2 numbers e.g., 1, 2, 4, 8, 16,

   Let us try for 17(10001)
   step 1
      n = n - 1 = 16 (10000)
   step 2
      n = n | n >> 1
      n = 10000 | 01000
      n = 11000
      n = n | n >> 2
      n = 11000 | 00110
      n = 11110
      n = n | n >> 4
      n = 11110 | 00001
      n = 11111
      n = n | n >> 8
      n = 11111 | 00000
      n = 11111
      n = n | n >> 16
      n = 11110 | 00000
      n = 11111
```

```
   step 3: Return n+1
    We get n + 1 as 100000 (32)
```

Program:

```
# include <stdio.h>

/* Finds next power of two for n. If n itself
   is a power of two then returns n*/

unsigned int nextPowerOf2(unsigned int n)
{
    n--;
    n |= n >> 1;
    n |= n >> 2;
    n |= n >> 4;
    n |= n >> 8;
    n |= n >> 16;
    n++;
    return n;
}

/* Driver program to test above function */
int main()
{
    unsigned int n = 5;
    printf("%d", nextPowerOf2(n));

    getchar();
    return 0;

}
```

Time Complexity: O(lgn)

References:

http://en.wikipedia.org/wiki/Power_of_2

Related Tpoics:

- Check if a number is multiple of 9 using bitwise operators
- How to swap two numbers without using a temporary variable?
- Divide and Conquer | Set 4 (Karatsuba algorithm for fast multiplication)
- Find position of the only set bit
- Swap all odd and even bits
- Add two bit strings
- Write your own strcmp that ignores cases
- Binary representation of a given number

Writing code in comment? Please use **ideone.com** and share the link here.