COMPUTE X^N (X TO THE POWER N)

Write a C function to compute x^n which uses minimum number of multiplications.

```
1    unsigned int power(double x, unsigned int n)
```

Brute Force Solution: O(n) time

The brute force method for this is to use a for loop

```
1    double power(double x, unsigned int n)
2    {
3        double product = 1;
4        for(int i=0; i<n; i++)
5        {
6            product = product * x;
7        }
8        return product;
9    }
```

We can also write the function recursively. But recursion gives no advantage in terms of performance. In fact it is always less optimal than non-recursive version.

```
1    double power(double x, unsigned int n)
2    {
3        if (n == 0)
```

```
4            return 1;
5        return x * power(x, n-1);
6    }
```

Time Complexity: O (n)

Space Complexity: For non-recursive version it's O (1), for recursive version it's O (n).

Optimized Solution: O (lg(n)) time

A Better Solution is to do Exponentiation by squaring. In this method we take advantage of the basic formula:

$x^{n+m} = x^m * x^n$

Hence,

$x^n = x^{n/2} * x^{n/2}$ (If x is even)

$x^n = x * x^{n/2} * x^{n/2}$ (If x is odd)

```
1    double power(double x, unsigned int n)
2    {
3        if( n == 0)
4            return 1;
5
6        double retValue = 1;
7        while(n)
8        {
```

```
9           if(n%2)
10              retValue = retValue * x;
11          x = x * x;
12          n = n/2;
13       }
14      return retValue;
15  }
```

The recursive version of above code will be something like:

```
1   double power(double x, unsigned int n)
2   {
3       if( n == 0)
4           return 1;
5
6       double temp = power(x, n/2);
7
8       if (n%2 == 0)
9           return temp*temp;
10      else
11          return x*temp*temp;
12  }
```

Note that the above code is not same as the below code (though both are recursive and both does exponential multiplication)

```
1    unsigned int power(double x, unsigned int n)
2    {
3        if (n == 0)
4            return 1;
5
6        if(x%2 == 0)
7            return power(x, n/2) * power(x, n/2);
8        else
9            return x * power(x, n/2) * power(x, n/2);
10   }
```

In the above code we are calling power method twice and hence it will be much more time consuming. (if n=10) then power(x,5) will be called twice. Its a typical problem with recursion.

The approach in the first recursive function is that call power(x,5) once and store its value in a variable (temp) and for second time use that value. This approach is called Dynamic Programming.

Source: http://www.ritambhara.in/compute-xn-x-to-the-power-n/