**Rohit Singh**

* **Write a program to calculate using recursion x^y ( X power y) . X and y are input.**
* **Calculate time complexity of written code.**
* **Write a better solution in terms of time complexity.**
* **Calculate time complexity again for a better solution.**

**Solution :-**

**1st Approach(Basic Approach using recursion)**

|  |
| --- |
| // C++ program to calculate using recursion x^y ( X power y) . X and y are input.  #include<iostream>  **using** **namespace** std;  **class** RecursionPower  {  /\* Function for calculat x raised to the power y \*/  **public**:  **int** power(**int** x, unsigned **int** y)  {  **if** (y == 0)  **return** 1;  **else** **if** (y % 2 == 0)  **return** power(x, y / 2) \* power(x, y / 2);  **else**  **return** x \* power(x, y / 2) \* power(x, y / 2);  }  };    /\* To test above functions \*/  **int** main()  {      RecursionPower obj;  **int** x = 3;      unsigned **int** y = 4;        cout << obj.power(x, y);  **return** 0;  } |

**Output : 81**

**Time Complexity:** O(y), Since, We are hitting Power Function for each Iteration. So we can say that there are Y number of hits.  
**Space Complexity:** O(1)  
**Algorithmic Paradigm:** Divide and conquer.

**2nd Approach(Better Approach in terms of Time Complexity using recursion)**

**Solution** :

Above Programme can be optimized to O(log y) by calculating power(x, y/2) only once and storing it for further result.

Because each recursion ,we decrease y by half

|  |
| --- |
| // C++ program to calculate using recursion x^y ( X power y) . X and y are input.  #include<iostream>  **using** **namespace** std;  **class** RecursionPower  {  /\* Function for calculat x raised to the power y \*/  **public**:  /\* this Function to calculate x raised to the power y in O(logn)\*/  **int** power(**int** x, unsigned **int** y)  {  **int** temp;  **if**( y == 0)  **return** 1;      temp = power(x, y/2);  **if** (y%2 == 0)  **return** temp\*temp;  **else**  **return** x\*temp\*temp;  }  };  /\* To test above functions \*/  **int** main()  {      RecursionPower obj;  **int** x = 3;      unsigned **int** y = 4;        cout << g. obj (x, y);  **return** 0;  } |

**\*Time Complexity of optimized solution: O(log(y)), as we are now calling power function for every y/2. So our complexity will be like - y + y/2+ y/4+ y/8 …. -> and that gives log(y).**

**3rd Approach(Better Approach in terms of Time Complexity without recursion and using bitwise operation)**

**Solution** :

The recursive solutions are mostly not preferred as they required space on call stack & they involve function call overhead.

So that we can use bitwise operation for better approach for this Programme

// C++ program to implement pow(x, n)

#include<iostream>

**using** **namespace** std;

**class** RecursionPower

{

/\* Function for calculating x raised to the power y \*/

**public**:

/\* This Function to calculate x raised to the power y in O(logn)\*/

/\* Iterative Function to calculate (x^y) in O(logy) \*/

**int** power(**int** x, unsigned **int** y)

{

**int** res = 1; // Initialize result

**while** (y > 0) {

        // If y is odd, multiply x with result

**if** (y & 1)

            res = res \* x;

        // n must be even now

        y = y >> 1; // y = y/2

        x = x \* x; // Change x to x^2

    }

**return** res;

}

};

/\* To test above functions \*/

**int** main()

{

    RecursionPower obj;

**int** x = 3;

    unsigned **int** y = 4;

    cout << obj.power(x, y);

**return** 0;

}

**Time Complexity** : O(log y), Since We are Iterating over Y’s Bits.  
 **Space Complexity:** O(1)