Exponential / Power function of order O(logN):

Exponential/Power function algorithms with O(logN) time complexity often involve efficient techniques like binary exponentiation. These algorithms reduce the number of operations by halving the problem size in each step.

Algorithm:

Binary exponentiation is a divide-and-conquer algorithm. Given a base x and an exponent n, it calculates x^n efficiently. The algorithm follows these steps:

1. Base Case:
   - If n=0, return 1.
   - If n is even, calculate y=x^n/2, and return y×y.
   - If n is odd, calculate y=x^(n−1)/2, and return x×y×y.
2. Divide and Conquer:
   - Break down the problem by recursively calculating the power of the base to half of the exponent in each step.

Code:

```
#include <iostream>
using namespace std;
double power(double x, int n) {
   if (n == 0) {
      return 1.0;
   }
   double halfPower = power(x, n / 2);
   double result = halfPower * halfPower;
   if (n % 2 == 1) {
      result *= x;
   }
   return result;
}

int main() {
   double number = 2.0;
   int power_1 = 10;
   cout<<"Enter Number :- ";
   cin >> number;
   cout<<"Enter Power :- ";
```

```
    cin >> power_1;
    double result = power(number, power_1);
    cout << number << " raised to the power of " << power_1 << " is: " << result << endl;
    return 0;
}
```

Output:

```
Enter Number :- 23
Enter Power :- 2
23 raised to the power of 2 is: 529
```

Space complexity: _____

Justification:_____
_____

Time complexity:

Best case time complexity: _____

Justification:_____
_____

Worst case time complexity: _____

Justification:_____