Analysis and Design of Algorithms Final Project Report

Two codes have been implemented.
Towers with Matrix multiplication and Towers with Strassen Matrix Multiplication.

Dynamic Programming:
The algorithm used works similar to Fibonacci series.
The general recursion formula is:
$f_n = 0$ for $n<0$
$f_0 = 1$
$f_n = h_1 * f_{n-1} + h_2 * f_{n-2} + h_3 * f_{n-3} + \ldots + h_{15} * f_{n-15}$ for $n>0$

If we have bricks of height i, set h_i to 1, else 0.
Implementation Explanation:
Matrix Multiplication:
Consider we have bricks of size 2 and 3 and we have to build a tower of height 20.
Hence, we need to consider the M matrix.
The M matrix for this example will be:

0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	1	1	0

V =

f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}

R=

f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}	f_{16}

Thus, $R = M^{(n-15)}V$
Thus, the final answer will be in R[R.length-1].

For TowersFinal.java:

After initializing the variables, we calculate the values of arrayH.
After that, we calculate values of f_n using the above mentioned formula.
Then if n>15, we calculate n-15, and apply Exponentiation by squaring on it.
We use matrix multiplication on these matrices, calculate value of R matrix and output the answer as the last value of R matrix * 2. (Since time to build each tower is 2 minutes)

For TowersWithStrassenFinal.java:

We do same as above, only instead of matrix multiplication, we perform Strassen matrix multiplication.

Functions:
TowersFinal.java

```
/*
Matrix Multiplication
 */
 static long[][] matrixMultiply(long[][] a, long [][] b)
 {
   long c[][] = new long[a.length][a.length];
   for(int i=0;i<a.length;i++)
   {
     for(int j=0;j<b.length;j++)
     {
       for(int p=0;p<b.length;p++)
       {
         c[i][j]+=a[i][p]*b[p][j];
       }
       c[i][j] = c[i][j] % (long)(Math.pow(10, 9)+7);
     }
   }
   return c;
 }

 /*
   After Matrix multiplication, to multiply it to several powers, for
   exponentiation by squaring, like (x^2)^8. So x^2 is done to a power
   of 8 times.
 */
 static long[][] powerMatrixMultiply(long[][] a, long power)
 {
```

```
        long[][] result = a;
        for(int i=1;i<power;i++)
        {
            result = matrixMultiply(result, a);
        }

        return result;
    }
```

TowersWithStrassenFinal.java:

```
// Strassen Multiplication
    static long[][] RecurMatMul(long a[][],long b[][])
    {
        //s1 is starting pos of
        if(a.length==2)
        {
            long ans[][]=new long[2][2];
            long m1=((a[0][0]+a[1][1])%(long)(Math.pow(10, 9)+7)
                    *(b[0][0]+b[1][1])%(long)(Math.pow(10, 9)+7))%(long)(Math.pow(10, 9)+7);
            long m2=((a[1][0]+a[1][1])%(long)(Math.pow(10, 9)+7)
                    *b[0][0]%(long)(Math.pow(10, 9)+7))%(long)(Math.pow(10, 9)+7);
            long m3=(a[0][0]%(long)(Math.pow(10, 9)+7)
                    *(b[0][1]-b[1][1])%(long)(Math.pow(10, 9)+7))%(long)(Math.pow(10, 9)+7);
            long m4=(a[1][1]%(long)(Math.pow(10, 9)+7)
                    *(b[1][0]-b[0][0])%(long)(Math.pow(10, 9)+7))%(long)(Math.pow(10, 9)+7);
            long m5=((a[0][0]+a[0][1])%(long)(Math.pow(10, 9)+7)
                    *b[1][1]%(long)(Math.pow(10, 9)+7))%(long)(Math.pow(10, 9)+7);
            long m6=((a[1][0]-a[0][0])%(long)(Math.pow(10, 9)+7)
                    *(b[0][0]+b[0][1])%(long)(Math.pow(10, 9)+7))%(long)(Math.pow(10, 9)+7);
            long m7=((a[0][1]-a[1][1])%(long)(Math.pow(10, 9)+7)
                    *(b[1][0]+b[1][1])%(long)(Math.pow(10, 9)+7))%(long)(Math.pow(10, 9)+7);
            ans[0][0]=(m1+m4-m5+m7)%(long)(Math.pow(10, 9)+7);
            ans[0][1]=(m3+m5)%(long)(Math.pow(10, 9)+7);
            ans[1][0]=(m2+m4)%(long)(Math.pow(10, 9)+7);
            ans[1][1]=(m1-m2+m3+m6)%(long)(Math.pow(10, 9)+7);
            return ans;
        }
        else
        {
            int d=a.length/2;
            //make total 4 subarrays of a and 4 subarrays of b
            long a11[][]=make(a,0,0,d);
            long a12[][]=make(a,0,d,d);
            long a21[][]=make(a,d,0,d);
```

```
        long a22[][]=make(a,d,d,d);

        long b11[][]=make(b,0,0,d);
        long b12[][]=make(b,0,d,d);
        long b21[][]=make(b,d,0,d);
        long b22[][]=make(b,d,d,d);


        long m1[][]=RecurMatMul(add(a11,a22,d), add(b11,b22,d));
        long m2[][]=RecurMatMul(add(a21,a22,d), b11);
        long m3[][]=RecurMatMul(a11,sub(b12,b22,d));
        long m4[][]=RecurMatMul(a22,sub(b21,b11,d));
        long m5[][]=RecurMatMul(add(a11,a12,d), b22);
        long m6[][]=RecurMatMul(sub(a21,a11,d), add(b11,b12,d));
        long m7[][]=RecurMatMul(sub(a12, a22, d),add(b21,b22,d));

        long c11[][]=sub(add(add(m1,m4,d),m7,d), m5, d);
        long c12[][]=add(m3,m5,d);
        long c21[][]=add(m2,m4,d);
        long c22[][]=sub(add(add(m1,m3,d),m6,d), m2, d);
        long c[][]=new long [a.length][a.length];
        c=merge(c,c11,0,0,d);
        c=merge(c, c12, 0, d, d);
        c=merge(c, c21, d, 0, d);
        c=merge(c, c22, d, d, d);
        return c;
    }
}
```

Implementation Tricks:

- Used modulo (10^9 + 7) in both the codes since matrix multiplication sometimes gives long numbers, or signed numbers.
- Added extra rows and columns of 0s to matrix for Strassen. (Static padding).
- Exponentiation by squaring done recursively till smallest value is obtained.

Snapshots:

Test cases.

Build time is inclusive of the time taken to enter the input as well.

Output – Others (run)

```
run:
1000000
2
2 3
874548080
BUILD SUCCESSFUL (total time: 2 minutes 49 seconds)
```

Output – Others (run)

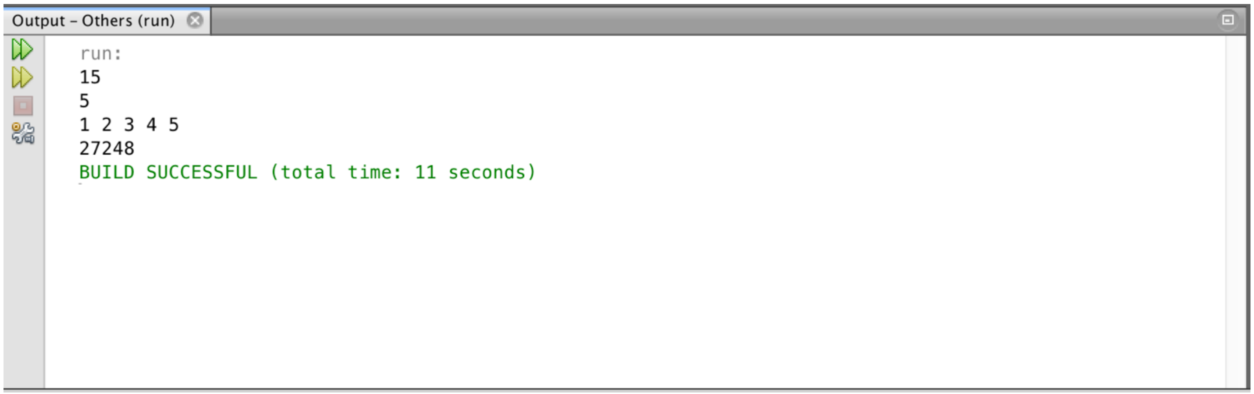
```
run:
10
1
1
2
BUILD SUCCESSFUL (total time: 8 seconds)
```

Output – Others (run)

```
run:
5
2
2 3
4
BUILD SUCCESSFUL (total time: 8 seconds)
```

Output – Others (run)

```
run:
19
2
4 5
8
BUILD SUCCESSFUL (total time: 11 seconds)
```



Time analysis:
TowersFinal.java:

When we get values till n<=15, we calculate the matrix based on Dynamic Programming using the function setDPMatrix(). Thus complexity is, O(n) when n<=15
When we multiply the matrix using Matrix Multiplication, we use the function matrixMultiply(). Thus complexity is $O(n^3)$
After applying exponentiation by squaring, $O(n^3 * \log_2 n)$
Hence, time complexity is: $O(n^3 * \log_2 n)$

Likewise,
TowersWithStrassenFinal.java:

setDPMatrix(): O(n) when n<=15
matrixMultiply(): $O(n^{2.8})$
After applying exponentiation by squaring, $O(n^{2.8} * \log_2 n)$
Hence, time complexity is: $O(n^{2.8} * \log_2 n)$

Space analysis:
TowersFinal.java:

setDPMatrix(): O(n) when n<=15
matrixMultiply(): $O(n^2)$
After applying exponentiation by squaring, $O(n^2 * \log_2 n)$
Hence, space complexity is: $O(n^2 * \log_2 n)$

TowersWithStrassenFinal.java:

setDPMatrix(): O(n) when n<=15
matrixMultiply(): $O(n^2)$
After applying exponentiation by squaring, $O(n^2 * \log_2 n)$
Hence, space complexity is: $O(n^2 * \log_2 n)$