Algorithms and Data Structures

Arnav Singh

March 9, 2020

5 Sheet 5

5.1 Fibonacci Numbers

5.1.1 Implemented all 4 methods, in zip file check fib.cpp file

5.1.2 Table of value for increasing n

No. of fib elements	Naïve	Closed	Bottom Up	Matrix
1	1,00E-06	1.1e-05	9,00E-06	2,00E-06
2	0,00E+00	1,00E-06	1,00E-06	1,00E-06
5	0,00E+00	1,00E-06	0	0
12	2,00E-06	1,00E-06	1,00E-06	0
30	0.004321	0	0	1,00E-06
75	Reached limit	1,00E-06	0	0
187	Reached limit	0	1,00E-06	2,00E-06
467	Reached limit	1,00E-06	1,00E-06	2,00E-06
1167	Reached limit	0	3,00E-06	5,00E-06
2917	Reached limit	0	7,00E-06	1,00E-05
7292	Reached limit	1,00E-06	2.2e-05	2.3e-05
18230	Reached limit	1,00E-06	4.8e-05	5.8e-05
45575	Reached limit	2,00E-06	0.000122	0.000146
113937	Reached limit	0	0.000308	0.000367
284842	Reached limit	0	0.000773	0.000916
712105	Reached limit	1,00E-06	0.001918	0.00229
1780262	Reached limit	0	0.005491	0.005866
4450655	Reached limit	0	0.015028	0.014028
11126637	Reached limit	0	0.034345	0.036595
27816592	Reached limit	1,00E-06	0.083827	0.088135
69541480	Reached limit	0	0.19936	0.226127
173853700	Reached limit	1,00E-06	0.508536	0.576505
434634250	Reached limit	1,00E-06	Reached limit	Reached limit

Figure 1: Running time of all 4 methods with respect to increasing value of n in seconds , time limit taken as 1 second

5.1.3 Value of n

For all the methods the value of n is not the same. For the closed form where we implement the golden ratio, the Fibonacci number would be different because as the the value of increases the larger n gets, the compiler will give rounding off approximation errors, hence will lead a small error.

5.1.4 Plot of Graph

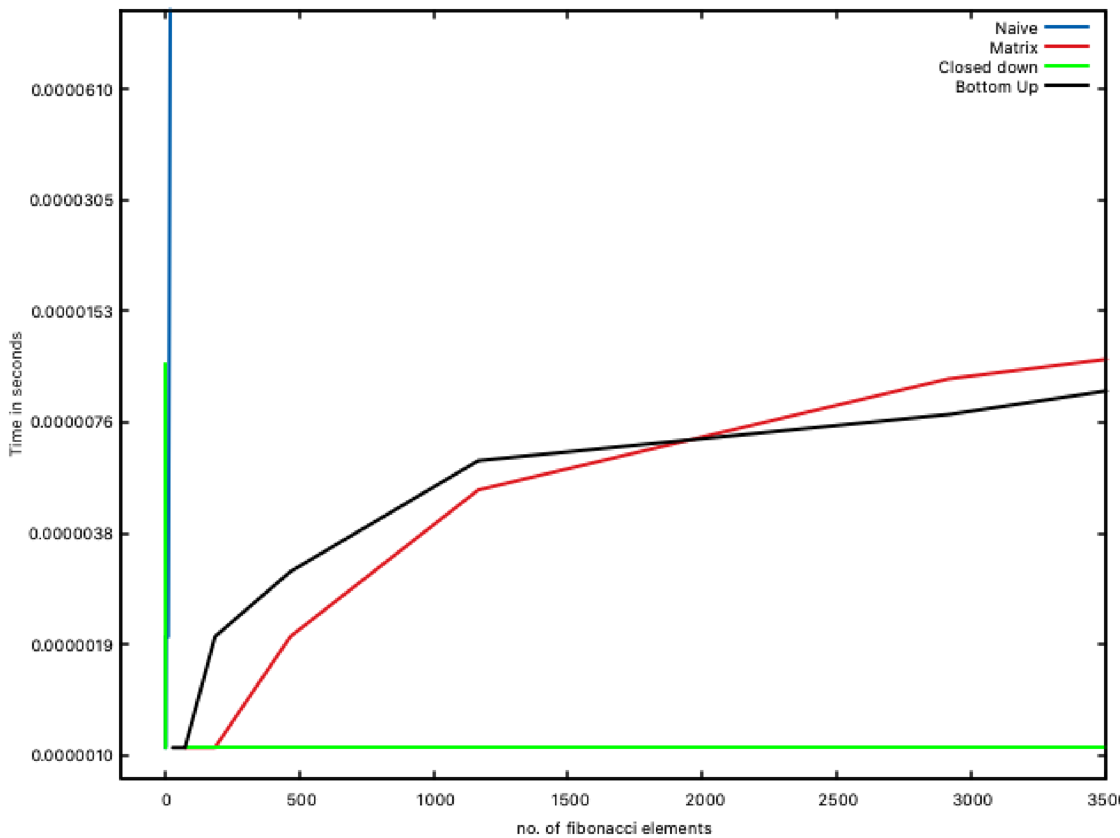


Figure 2: Plot of the running time of all methods with respect to n number of elements,(fixed time taken as 1 second)

5.2 Divide/Conquer and Solving Recurrences

5.2.1 Derive time complexity

Assume two numbers x, y each of n bits.
Multiplying each bit of a number x by each bit of a number y and doing this for n times is n^2
Shifting and adding the result is n
Therefore total time complexity is: $n^2 + n$

$$T(n) = \Theta(n^2) \tag{1}$$

5.2.2 Derive the algorithm

Consider two numbers x and y each of n bits
First we divide x into two parts a and b such that
$x = a * 2^{\frac{n}{2}} + b$ and similarly for $y = c * 2^{\frac{n}{2}} + d$
Therefore xy=$(a * 2^{\frac{n}{2}} + b)$*$(c * 2^{\frac{n}{2}} + d)$

On expanding we get,

$$\text{ac}\cdot 10^n + 10^{\frac{n}{2}}(\text{ad}+\text{bc})+\text{bd}$$

Here notice that we have 4 multiplications altogether at the highlighted parts However we can decrease the number of multiplications by rewriting the middle term i.e

$$(ad + bc) = (a + b) \cdot (c + d) - ac - bd \tag{2}$$

Thus we finally have,

$$\underline{ac} \cdot 10^n + 10^{\frac{n}{2}}[\underline{(a + b) \cdot (c + d)} - ac - bd)] + \underline{bd} \tag{3}$$

Now we have 3 distinct multiplications which are underlined. A Pseudo code for the algorithm is given in the last page.

5.2.3 Derive Recurrence relation

Since the algorithm requires three distinct multiplications of size n/2 for every subdivision the adding and shifting bits take n time recurrence becomes
$T(n) = 3T(\frac{n}{2}) + n$

5.2.4 Solve the recurrence relation using the tree method

Done in the bonus file

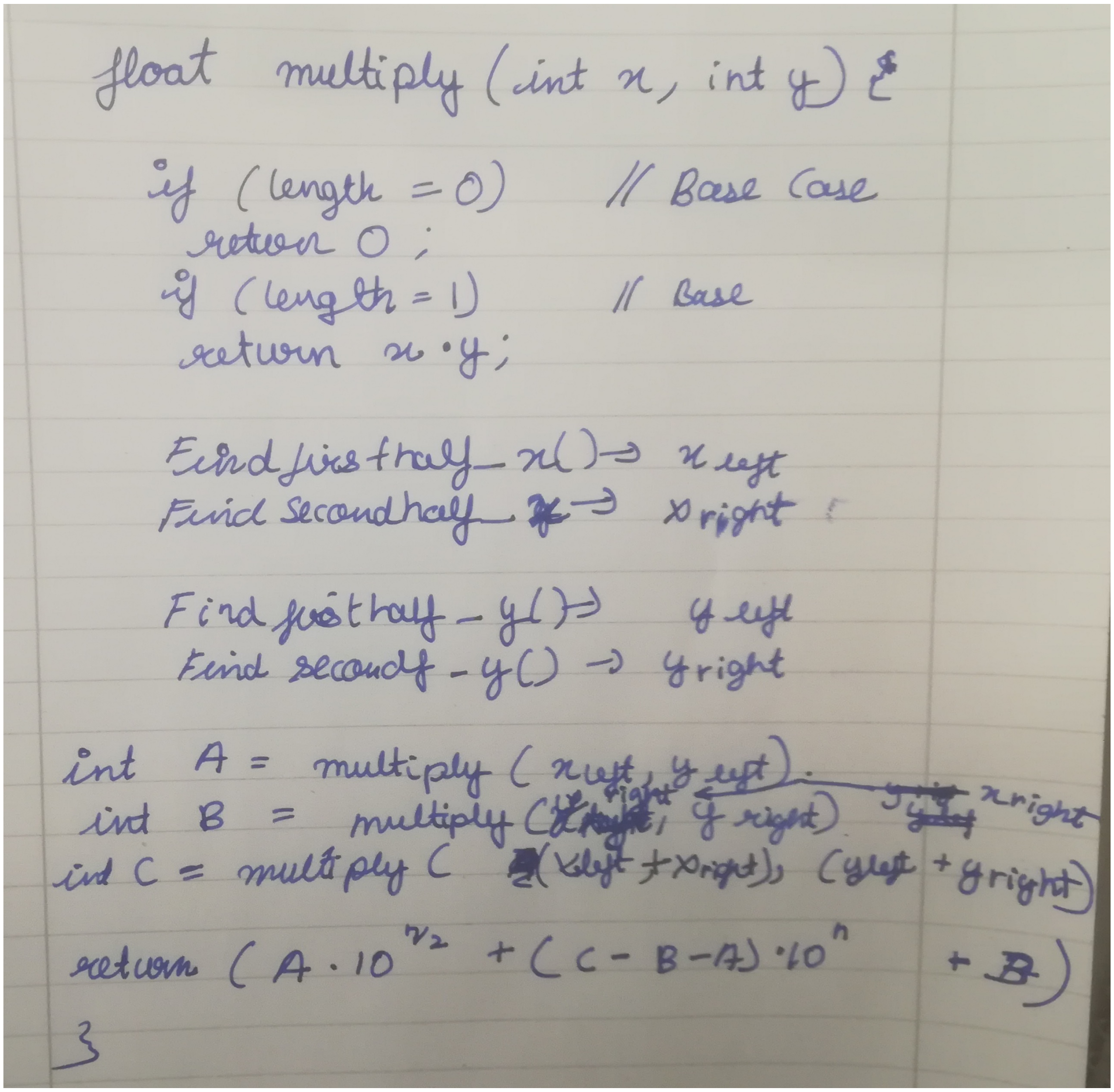
5.2.5 Solve it using the master method

$T(n) = 3T(\frac{n}{2}) + n$

$$n^{\log_b a = n^{\log_2 3 = n^{1.58}}} \text{ and } f(n) = n$$

This is Case 1 as $f(n)$ is polynomially smaller than $n^{1.58}$.
$f(n) = O(n^{1-e})$ for $e = 0.58$
Thus by the master theorem: $T(n) = \Theta(n^{\log_2 3)}$

Check the next page



```
float multiply (int x, int y) {

  if (length = 0)        // Base Case
   return 0;
  if (length = 1)        // Base
   return x·y;

  Find firsthalf_x() → x left
  Find secondhalf_x → x right

  Find firsthalf_y() → y left
  Find secondf_y() → y right

int A = multiply (x left, y left).
 int B = multiply (x right, y right)
int C = multiply ( (xleft + xright), (yleft + yright))
return (A·10^(n/2) + (C - B - A)·10^n + B)
}
```

Figure 3: Pseudo Code for the algorithm