CS325- Homework -1

Q1.

Insertion Sort beats Merge sort. Which means Merge sort takes more time than insertion sort.

$8n^2 < 64n\lg n$

Solving the equation

$n < 8\lg n$

By trial and error or drawing a graph, this is true for n>1 and n<43.56.

Q2.

Function	1 Second	1 Minute	1 Hour	1 Day	1 Month	1 Year	1 Century
lg n	2^10^6	2^60.10^6	2^60.60.10^6	2^24.60.60*10^6	2^30.24.60.60.10^6	2^12.30.24.60.60.10^6	2^100.12.30.24.60.60.10^6
√ n	(10^6)^2	(60.10^6)^2	(24.60.60*10^6)^2	(24.60.60*10^6)^2	(30.24.60.60.10^6)^2	(12.30.24.60.60.10^6)^2	(100.12.30.24.60.60.10^6)^2
n	10^6	60.10^6	60.60.10^6	24.60.60*10^6	30.24.60.60.10^6	12.30.24.60.60.10^6	100.12.30.24.60.60.10^6
n lg n	62746	2801417	133378058	2755147513	71870856404	797633893349	68654697441062
n^2	√(10^6)	√(60.10^6)	√(24.60.60*10^6)	√(24.60.60*10^6)	√(30.24.60.60.10^6)	√(12.30.24.60.60.10^6)	√(100.12.30.24.60.60.10^6)
n^3	^{3}√(10^6)	^{3}√(60.10^6)	^{3}√(24.60.60*10^6)	^{3}√(24.60.60*10^6)	^{3}√(30.24.60.60.10^6)	^{3}√(12.30.24.60.60.10^6)	^{3}√(100.12.30.24.60.60.10^6)
2^n	19.9	25.8	31.7	36.3	41.2	44.8	51.4
n!	9.4	11.1	12.7	13.9	15.2	16.1	17.7

The n lgn, 2^n, n! values were calculated using wolfram Alpha (Rounded off).

Q3.

Base Step:

$F(1)=T(2)=2=2\lg 2=2^1 \lg 2^1$

Induction Step:

$F(k)=2^k \lg 2^k$

We prove it for k+1:

$F(k+1)=T(2^{k+1})=2T(2^{k+1}/2)+2^{k+1}$

$== 2\ T(2^k)+2^{k+1}$

$== 2\cdot 2^k \lg 2^k + 2^{k+1}$

$==2^{k+1}(\lg 2^k + 1)$

$==2^{k+1}(\lg 2^k + \lg 2)$ because $\lg 2 = 1$

$==2^{k+1}\lg 2^{k+1}$

Hence proved.

Q4.

I am using below limit method to solve these questions.

$$\text{If } \lim_{n\to\infty} \frac{f(n)}{g(n)} = \begin{cases} 0 & \text{then } f(n) = O(g(n)) \\ \text{some finite, non-zero, positive constant} & \text{then } f(n) = \Theta(g(n)) \\ \infty & \text{then } f(n) = \Omega(g(n)) \end{cases}$$

a. lt n-> ∞ f(n)/g(n)
Apply LHopital Rule
lt n -> ∞ f'(n)/g'(n)
= lt n -> ∞ 1/(2n-100)
= 0
f(n) = O(g(n)), n>=101

b. lt n-> ∞ f(n)/g(n)
Apply LHopital Rule
lt n -> ∞ f'(n)/g'(n)
= lt n -> ∞ $(1/4x^{0.75})/(1/2\sqrt{x})$
= 0
f(n) = O(g(n)), n>1

c. lt n-> ∞ f(n)/g(n)
Apply LHopital Rule
lt n -> ∞ f'(n)/g'(n)
= lt n -> ∞ $1/\ (2\cdot\ln(x)/\ln^2(10)\cdot x)$
Again L hospital rule
lt n -> ∞ $((2.\ln 10x/x)/(2/x))$

$= \infty$
$f(n) = \Omega(g(n))$, $n>0$

d. lt n-> ∞ f(n)/g(n)
Apply LHopital Rule
lt n -> ∞ f'(n)/g'(n)
= lt n -> ∞ $(1/n.\ln 2)/((2. \ln 3n)/(\ln^2 10).n)$
= lt n -> ∞ $(\ln^2 10)/(2. \ln 2. \ln 3n)$
= 0
$f(n) = O(g(n))$, $n>185$

e. lt n-> ∞ f(n)/g(n)
Apply LHopital Rule
lt n -> ∞ f'(n)/g'(n)
= lt n -> ∞ $(1/n. \ln 10)/(1/n. \ln 2)$
= 0.3010 C1<0.3010 and C2>0.3010
$f(n) = \Theta(g(n))$

f. lt n-> ∞ f(n)/g(n)
Apply LHopital Rule
lt n -> ∞ f'(n)/g'(n)
= lt n -> ∞ $(2^n \ln 2)/(20n)$
Again LHopital
= lt n -> ∞ $(2^n \ln^2 2)/(20)$
= ∞
$f(n) = \Omega(g(n))$, $n>=10$

g. lt n-> ∞ f(n)/g(n)
Apply LHopital Rule
lt n -> ∞ f'(n)/g'(n)
= lt n -> ∞ $(e^x)/(2^n \ln 2)$
Always getting ∞/∞
Try substitution
e = 2.17
so e^n is > 2^n for n>=0
$f(n) = \Omega(g(n))$, $n>=0$

h. Cant prove by limits. Because always gives ∞/∞.
By substitution, we know that $2^n < 2^{n+1}$ for n >=0

Because $2^n < 2^n . 2$
$f(n) = O(g(n))$, $n>0$

i. Cant prove by limits. Because always gives ∞/∞.
By substitution, we know that $2^n < 2^{2^n}$ for n >=1
$f(n) = O(g(n))$, n>=1

j. Cant prove by limits. Because always gives ∞/∞.
By substitution, we know that $n.2^n > 2^n$ for n >=1
Because a value multiplied with any positive number greater than 1 is always greater than that number itself.
$f(n) = \Omega(g(n))$, n>=1

k. Cant prove by limits. Don't know derivative of n!
By substitution, we know that $2^n < n!$ for n >=4
Usually n will be very large
There fore $2^n < n!$
$f(n) = O(g(n))$, n>=4

l. Cant prove by limits. Don't know derivative of n!
By substitution, we know that $(n+1)! > n!$ for n >0
Because n+1 >n always for positive n
Therefore $(n+1)! > n!$ for n >0
$f(n) = \Omega(g(n))$, n>=0

Q5.

a. $f_1(n) = O(f_2(n)) \rightarrow f_2(n) = O(f_1(n))$
No, because $f_1(n) = O(f_2(n)) \Rightarrow f_1(n) \le C.\ f_2(n)$ C >0
And $f_2(n) = O(f_1(n)) \Rightarrow f_2(n) \le C.\ f_1(n)$
The first one says $f_1 < f_2$ and second one says $f_2 < f_1$ which is not possible. Therefore this is not true.

b. $f_1(n) = O(g_1(n))$ and $f_2(n) = O(g_2(n)) \rightarrow f_1(n) * f_2(n) = O(g_1(n) * g_2(n))$
$f_1(n) = O(g_1(n)) \Rightarrow f_1(n) \le C_1.g_1(n)$
$f_2(n) = O(g_2(n)) \Rightarrow f_2(n) \le C_2.g_2(n)$
$f_1(n) * f_2(n) \Rightarrow C_1.g_1(n) * C_2.g_2(n)$
$= C_1.\ C_2\ (g_1(n)\ .\ g_2(n))$
$= C_3.(\ g_1(n)\ .\ g_2(n))$ Assume, $C_3 = C_1.\ C_2$
$= O(g_1(n) * g_2(n))$ $C_1, C_2, C_3 > 0$ and $n > n_0$

c. $\max(f_1(n), f_2(n)) = \Theta(f_1(n) + f_2(n))$

g(n)
n₀

$\max(f_1(n),f_2(n)) = \Theta(f_1(n) + f_2(n)) \Rightarrow C_1 . (f_1(n) + f_2(n)) <= \max(f_1(n),f_2(n)) <= C_2 . (f_1(n) + f_2(n))$
$C_1, C_2 > 0, \; n > n_0$
This can be true in following cases:

1. If $f_1(n) = f_2(n)$, Then $C_1 <= 0.5$, $C_2 >= 0.5$
2. If $f_1(n) != f_2(n)$, Then $C_1 <= 0.5$, $C_2 >= 1$

Q6.

a. Code:

Fibonacci Recursive code:

```
#include<stdio.h>
#include<time.h>

int fib(int n)
{
   if ( n == 0 )
      return 0;
   else if ( n == 1 )
      return 1;
   else
      return ( fib(n-1) + fib(n-2) );
}
int main()
{
     clock_t start,end, duration;
     start = clock();
   int n = 75;
   int i;
   for ( i = 0 ; i < n ; i++ )
   {
      fib(i);
      //printf("%d\n", fib(i));
   }
   end = clock();
   duration = (end - start)/CLOCKS_PER_SEC;
   printf("Duration: %Lf\n",(long double)duration );
   return 0;
}
```

Fibonacci_Iterative Code:

```
#include<stdio.h>
#include<time.h>

int main()
{
   clock_t start,end, duration;
   start = clock();
   int n = 10000000 ;
   int first = 0, second = 1, next,i;
   for ( i = 0 ; i < n ; i++ )
   {
      if ( i <= 1 )
         next = i;
      else
      {
         next = first + second;
         first = second;
         second = next;
      }
      printf("%d\n",next);
   }
   end = clock();
   duration = (end - start)/CLOCKS_PER_SEC;
   printf("Duration: %Lf\n",(long double)duration );
   return 0;
}
```

b. Run Times:

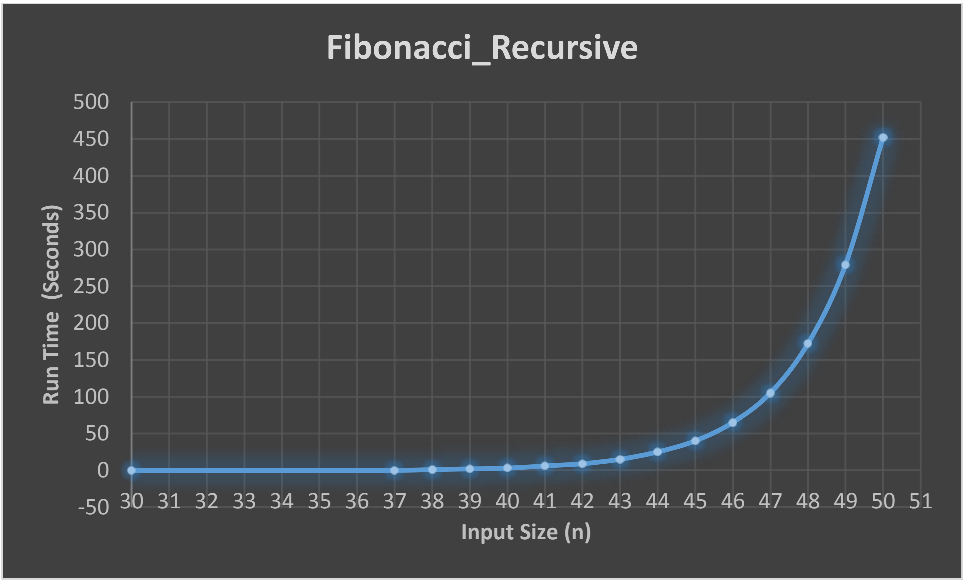
Fibonacci_Recursive:

Input Size	Run Time (seconds)
5	0
10	0
15	0
20	0
30	0
37	0
38	1
39	2
40	3
41	6
42	9
43	15
44	25
45	40
46	65
47	105
48	172
49	279
50	452

Fibonacci_Iterative:

Input Size (Million)	Run Time (seconds)
0.5	0
0.6	0
0.7	1
1	2
2	2
3	8
4	10
5	13
10	27
20	60
30	98
40	141
50	144

c. Run time Plot:

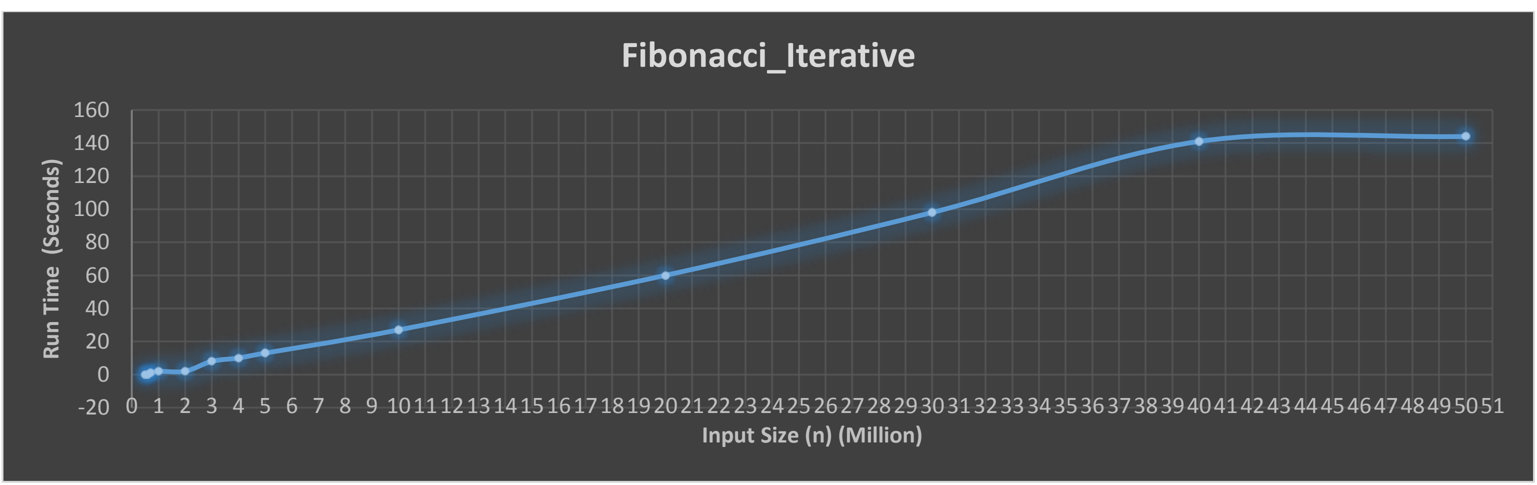


The graph is exponential. While using recursive method, The Run time increases exponential after certain input size (> 37) which is because of recursively calculating and storing the value and using it again.

Here I didn't include 100, 1000, 2000, 10000 because, my system is taking a lot of time, sometimes hangs while executing such larger n size. So, I am restricting to these values.

We know that Time complexity of Fibonacci_Iterative is T(n) = T(n-1) + T(n-2) + c which is 2^n.

But My graph looks like 2^n. But A small deviation from actual.



The Graph goes linear upto some extent. Looks like 3x graph. We know that time complexity of Fibonacci Iterative series is O(n). So My code is almost near to linear time complexity. The run time till 0.7 million is always 0 seconds. So plotted the graph by taking n size from 0.7 million to 50 million (intervals of 10 million).

The Run time is less when compared to iterative one because the operation here are just series of addition from variables where as in recursive, we need to store the previous recursion values and sum them.

Q7.

1. Sort the elements of array using merge sort.
2. For each element of array S, set var1 = S[i] – x;
3. Search for var1 in array S. If exists, return S[i] and var1.
4. If not, terminate saying no values found.
5. We know that Merge sort has $\Theta(n \lg n)$ complexity (proved in class).
6. Binary search also has $\Theta(n \lg n)$ complexity.
7. So, Overall complexity of our algorithm is $T(n) = \Theta(n \lg n) + \Theta(n \lg n)$.
8. Which is nothing but $2^* \Theta(n \lg n) = \Theta(n \lg n)$

Submitted By:

Lakshman Madhav Kollipara

For CS325_Homework 1