Matthew Oakley
Algorithm analysis and design
Dr. Rivas
2/6/18

```java
import java.util.Scanner;

public class fib{

  public static int posEfibonacci(int num){
    if(num <= 3)
      return 1;
    else
      return posEfibonacci(num - 1) + posEfibonacci(num - 2) + posEfibonacci(num - 3);
  }
  public static int negEfibonacci(int num){
    if(num >= -3)
      return -1;
    else
      return negEfibonacci(num + 1) - negEfibonacci(num + 2) - negEfibonacci(num + 3);
  }
  public static void main(String[] args){
    Scanner input = new Scanner(System.in);
    int num = input.nextInt();
    int output = 0;
    if(num > 0)
      output = posEfibonacci(num);
    else
      output = negEfibonacci(num);
    System.out.println(output);
  }
}

```

My code will check if what was put in was positive or negative. Based on that it runs two different efibonacci methods. The positive adds the numbers together. The negative minus the numbers. Efib of 10 is 105, while -10 is 1.
My efibonacci methods are decent for smaller input but as the input gets larger problems could arise. One major problem is that because it is recursive it could cause a stack overflow error. But there are some tradeoffs from this like the coding being easier to read. These two methods also make progress towards the base cases each time.
Growth Rates:

Slowest
37
2/n
sqrt(n)
nlog(log(n))
n

nlog(n)
nlog(n^2)
nlog(n)^2
n^1.5
n^2
n^2 log(n)
2^(n/2)
n^3
2^n
Fastest

A. (2^(2^days - 1)) B. amount = 2^(2^n - 1))

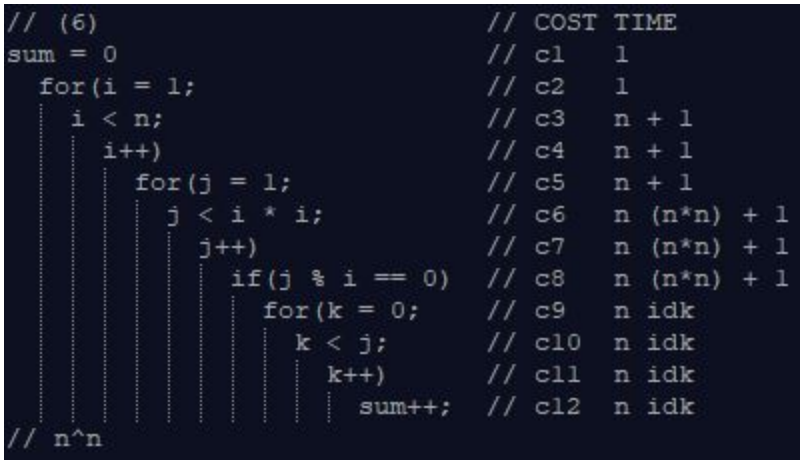
```
// (1)          // COST TIME
sum = 0;        // c1   1
  for(i = 0;    // c2   1
    i < n;      // c3   n + 1
      i++)      // c4   n + 1
    sum++;      // c5   n
// n + (n + 1) + (n + 1) => (n)


// (2)              // COST TIME
sum = 0;            // c1   1
for(i = 0;          // c2   1
  i < n;            // c3   n + 1
    i++)            // c4   n + 1
      for(j = 0;    // c5   n + 1
        j < n;      // c6   n * n + 1
          j++)      // c7   n * n + 1
            sum++;// c8   n * n
// (n + 1) + (n + 1) + (n*n) + (n*n + 1) + (n*n + 1) + (n*n) => n^2


// (3)                // COST TIME
sum = 0;              // c1   1
for(i = 0;            // c2   1
  i < n;              // c3   n + 1
    i++)              // c4   n + 1
      for(j = 0;      // c5   n + 1
        j < n * n;    // c6   n * n^2 + 1
          j++)        // c7   n * n^2 + 1
            sum++;    // c8   n * n^2
// (n + 1) + (n + 1) + (n + 1) + (n * n^2 + 1) + (n * n^2 + 1) + (n * n^2) => n^3
```

```
// (4)              // COST TIME
sum = 0;            // c1   1
for(i = 0;          // c2   1
  i < n;            // c3   n + 1
    i++)            // c4   n + 1
      for(j = 0;    // c5   n + 1
        j < i;      // c6   n * (n + 1) / 2
          j++)      // c7   n * (n + 1) / 2
            sum++;  // c8   n * (n + 1) / 2
// (n + 1) + (n + 1) + (n + 1) + (n(n + 1) / 2) + (n(n + 1) / 2)  +  (n(n + 1) / 2) => (n^n / 2 )


// (5)                      // COST TIME
sum = 0;                    // c1   1
for(i = 0;                  // c2   1
  i < n;                    // c3   n + 1
    i++)                    // c4   n + 1
      for(j = 0;            // c5   n + 1
        j < i * i;          // c6   n (n^n) + 1
          j++)              // c7   n (n^n) + 1
            for(k = 0;      // c8   n (n^n) + 1
              k < j;        // c9   n (n^n) * n + 1
                k++)        // c10  n (n^n) * n + 1
                  sum++;    // c11  n (n^n) * n
// n^n
```



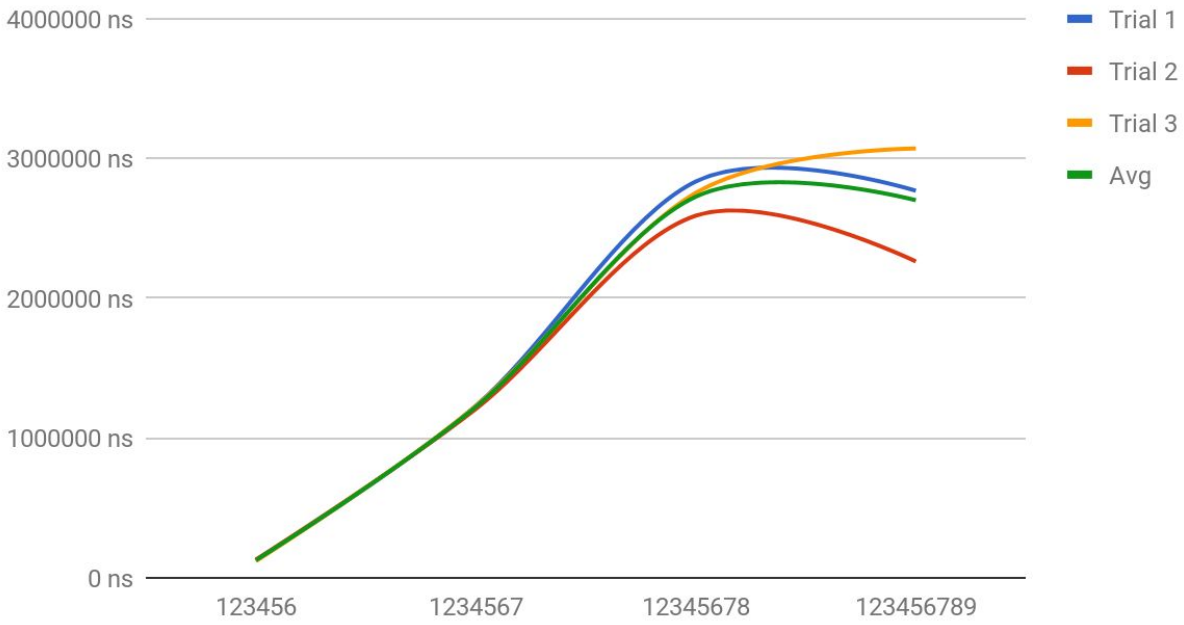
```
// (6)                          // COST TIME
sum = 0                         // c1   1
  for(i = 1;                    // c2   1
    i < n;                      // c3   n + 1
    i++)                        // c4   n + 1
      for(j = 1;                // c5   n + 1
        j < i * i;              // c6   n (n*n) + 1
        j++)                    // c7   n (n*n) + 1
          if(j % i == 0)        // c8   n (n*n) + 1
            for(k = 0;          // c9   n idk
              k < j;            // c10  n idk
              k++)              // c11  n idk
                sum++;          // c12  n idk
// n^n
```

Question 1					
Input	Trial 1	Trial 2	Trial 3	Avg	Unit of Time
12345	127210	129967	120099	125759	ns
123456	1230617	1208889	1229433	1222980	ns
1234567	2836149	2591606	2754767	2727507	ns
12345678	2769779	2263704	3072001	2701828	ns
123456789	3973927	2640988	3339063	3317993	ns

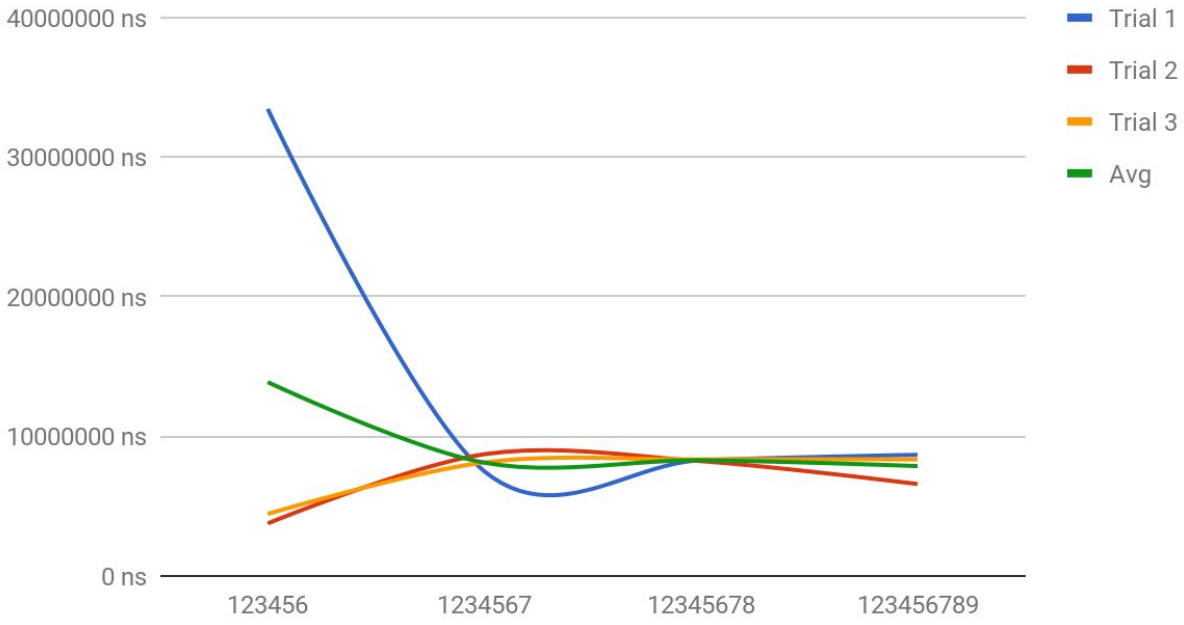
Question 2					
Input	Trial 1	Trial 2	Trial 3	Avg	Unit of Time
12345	33457791	3764150	4446026	13889322	ns
123456	7496299	8723361	8133139	8117600	ns
1234567	8287608	8224793	8348448	8286950	ns
12345678	8679905	6586866	8349238	7872003	ns
123456789	7909534	6772546	6733830	7138637	ns
Question 3					
Input	Trial 1	Trial 2	Trial 3	Avg	Unit of Time
12345	151551664	8736004	8771164	56352944	ns
123456	2148347	1728396	2106075	1994273	ns
1234567	3373828	3316940	3402668	3364479	ns
12345678	7804843	8551114	8267855	8207937	ns
123456789	3358421	4120891	3311013	3596775	ns
Question 4					

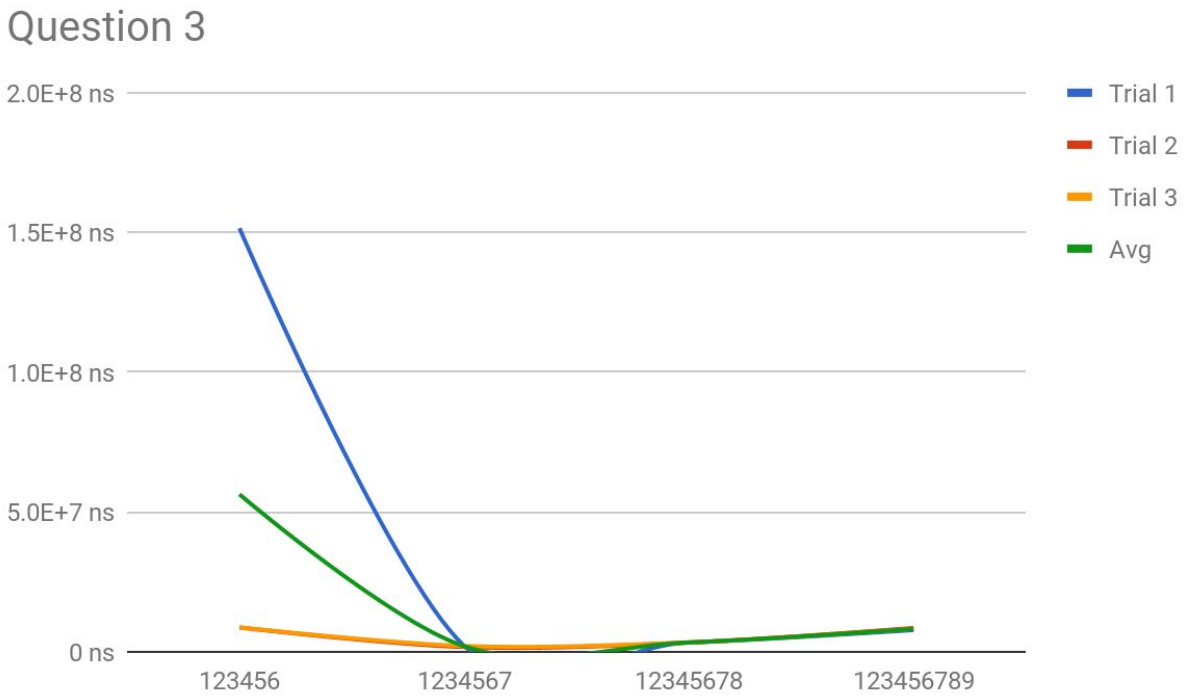
Input	Trial 1	Trial 2	Trial 3	Avg	Unit of Time
12345	4230718	4380841	5185582	4599047	ns
123456	3827359	3727014	3760990	3771788	ns
1234567	5062323	5174916	4114569	4783936	ns
12345678	12854524	12935906	11557536	12449322	ns
123456789	80698106	77125656	78768624	78864129	ns

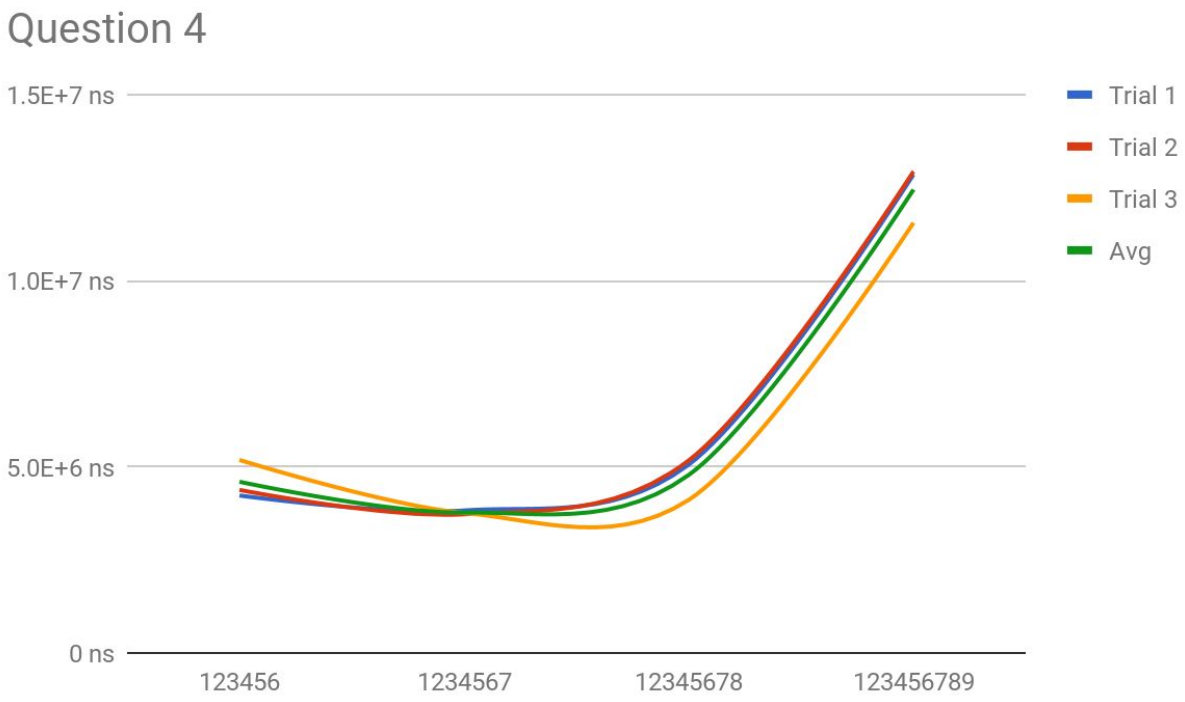
Question 1



Question 2







6. f(n) = O(log(n)) f(n) = O(3^n) f(n) = O(n) f(n) = Θ(n)
7. The best running time would be n!*n because it needs to go through each element of the array twice

8.

A. 1 to do a simple adding problem it can be done in constant time

B. 1 to do a multiplication problem it can be done in constant time

C. 1 to do a dividing problem this can be done in constant time

All of these can be done for any integer except C when the bottom number is 0