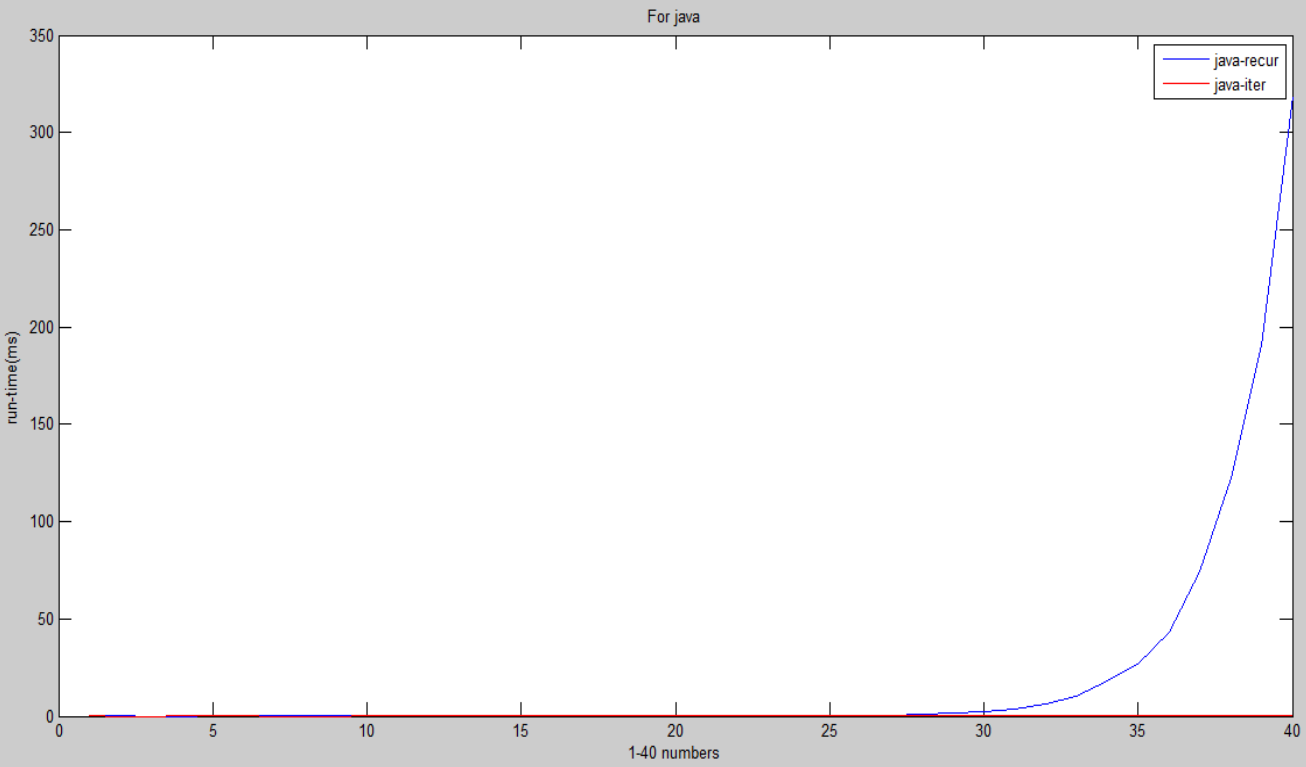
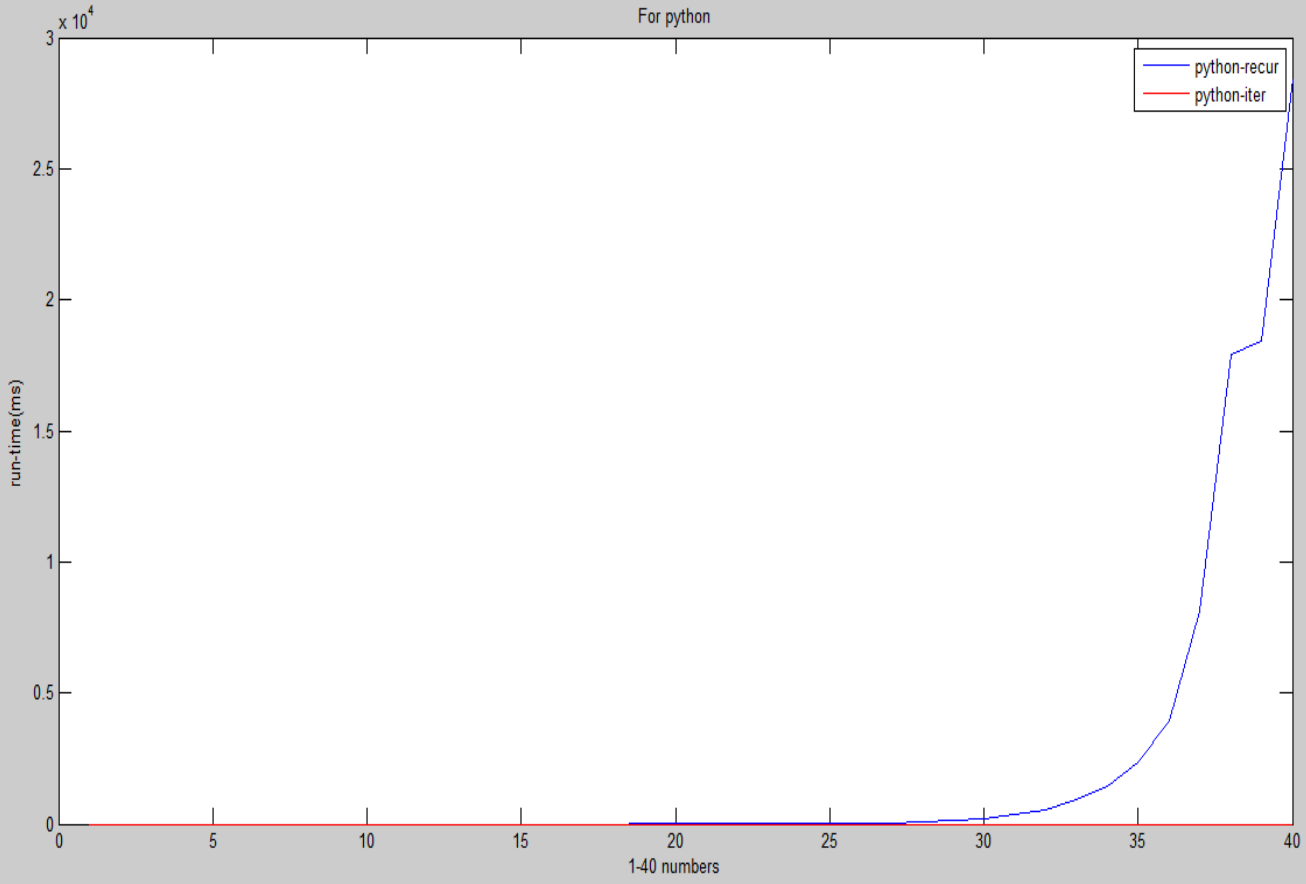

For java
java-recur
java-iter
run-time(ms)
1-40 numbers
0
50
100
150
200
250
300
350
0
5
10
15
20
25
30
35
40


For python
x 10^4
python-recur
python-iter
run-time(ms)
1-40 numbers
0
0.5
1
1.5
2
2.5
3
0
5
10
15
20
25
30
35
40

As you can see in the graph also, when the problem is small all the implementations run-time is almost 0ms (for about to 25). Although the actual values show different values we can considered them as 0ms.

When we compare java and python, java has the most advantage in run-time. Although the iteration graphs of java and python are much alike recursion graphs are much different. As the problem increases the run-time of python recursion code increase very rapidly, but it is less in the java code.

When the problem is small (1-25) both the implementations in java and python do not show much difference of run-time. So when the problem is small both the algorithms are useful.
But when the problem is large the recursion algorithm takes more run-time to show the answers.

The reason for the poor performance of recursion algorithm is heavy push-pop of the stack memory in each recursive call. This happens when the problem gets larger. Finding the Time complexity of Recursion is more difficult than that of Iteration.

The Iteration method would be the prefer and faster approach to solving our problem because we are storing the first two of our Fibonacci numbers in two variables (a, b) and using "fib" to store our Fibonacci number. Storing these values prevent us from constantly using memory space in the Stack. Thus giving us a time complexity of O(n).

From java and python, python recursion algorithm is the worst one for the use. So as a conclusion we can say that if the problem is large recursion is not useful.