```java
package uebung1111;

public class maintest {
    public static void main(String[] args) {
        FibonacciRekursive re = new FibonacciRekursive();
        FibonacciParallel pa = new FibonacciParallel();
        FibonacciDynamic dy = new FibonacciDynamic();
        FibonacciDynamicParallel dp = new FibonacciDynamicParallel();
        System.out.println("ergebnis recursive: " + re.calculate(16));
        System.out.println("ergebnis parallel: " + pa.calculate(16));
        System.out.println("ergebnis dynamic: " + dy.calculate(10));
        System.out.println("ergebnis dynamicparallel: " + dp.calculate(10));
    }
}

```

```java
package uebung1111;

public abstract class Fibonacci {

    public abstract long calculate(int n);

}

```

```java
package uebung1111;

import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.assertEquals;

class FibonacciTest {
    //Warum gibt es Fehler ab number 16 und höher????
    private static final int[] NUMBERS = new int[] { 3, 5, 8, 12, 9, 15, 10, 7, 11 };

    @Test
    void test() {

        for(int i = 0 ; i < NUMBERS.length; i++) {
            FibonacciRekursive recursive = new FibonacciRekursive();
            FibonacciParallel parallel = new FibonacciParallel();
            FibonacciDynamic dynamic = new FibonacciDynamic();
            FibonacciDynamicParallel dynamicParallel = new FibonacciDynamicParallel();

            long recursiveResult = recursive.calculate(NUMBERS[i]);
            long parallelResult = parallel.calculate(NUMBERS[i]);
            long dynamicResult = dynamic.calculate(NUMBERS[i]);
            long dynamicParallelResult = dynamicParallel.calculate(NUMBERS[i]);

            assertEquals(recursiveResult, parallelResult);
```

```
            assertEquals(recursiveResult, dynamicResult);
            assertEquals(recursiveResult, dynamicParallelResult);


        }
    }

}

```

```java
package uebung1111;

public class FibonacciRunner extends Thread{


    private long n;

    public long getResult() {
        return result;
    }

    private long result;
    public FibonacciRunner(long n){
        this.n = n;
    }
    @Override
    public void run() {
        result = calculate(n);
    }

    private long calculate(long n){
        long result;
        if(n == 0){
            return 0;
        }
        if(n == 1){
            return 1;
```

```java

        }
        else{
            FibonacciRunner nminus1 = new FibonacciRunner(n-1);
            FibonacciRunner nminus2 = new FibonacciRunner(n-2);

            nminus1.start();
            nminus2.start();

            try{
                nminus1.join();
                nminus2.join();
            }
            catch( InterruptedException e){

            }

            result =  nminus1.getResult() +  nminus2.getResult();
            return result;
        }
    }
}

```

```java
package uebung1111;

import java.util.HashMap;
import java.util.Map;

public class FibonacciDynamic extends Fibonacci {
    private Map<Integer, Long> memo;

    public FibonacciDynamic() {
        memo = new HashMap<>();
    }

    @Override
    public long calculate(int n) {
        if(n == 0){

            return 0;
        }
        if(n <= 1){

            return 1;

        }
        if (memo.containsKey(n)) {
            return memo.get(n);
        }

```

n < 0 fehlt

```
        long result = calculate(n -1) + calculate( n -2);
        memo.put(n, result);

        return result;
    }

}

```

```java
package uebung1111;

public class FibonacciParallel extends Fibonacci {
    @Override
    public long calculate(int n) {
        long result;
        if(n == 0){
            return 0;
        }
        if(n == 1){
            return 1;

        }
        else{
            FibonacciRunner nminus1 = new FibonacciRunner(n-1);
            FibonacciRunner nminus2 = new FibonacciRunner(n-2);

            nminus1.start();
            nminus2.start();

            try{
                nminus1.join();
                nminus2.join();
            }
            catch( InterruptedException e){

            }
```

```
            result =  nminus1.getResult() + nminus2.getResult();
            return result;
        }



    }
}
```

```java
package uebung1111;

public class FibonacciRekursive extends Fibonacci {


    @Override
    public long calculate(int n) {
        long result;
        if(n == 0){

            return 0;
        }
        if(n <= 1){

            return 1;

        }
        //Gibt Zahlen in der Reihenfolge des Fibonacci-Baums von links nach rechts al slong zurück
        result = calculate(n -1) +calculate( n -2);
        return result;
    }
}

```

```java
package uebung1111;

import java.util.HashMap;
import java.util.Map;

public class FibonacciDynamicParallel extends Fibonacci {
    private Map<Integer, Long> memo;

    public FibonacciDynamicParallel() {
        memo = new HashMap<>();
    }

    @Override
    public long calculate(int n) {
        long result;
        if(n == 0){

            return 0;
        }
        if(n <= 1){

            return 1;

        }
        if (memo.containsKey(n)) {
            return memo.get(n);
        }
```

Dynamic?

n < 0 fehlt

```
        else{
            FibonacciRunner nminus1 = new FibonacciRunner(n-1);
            FibonacciRunner nminus2 = new FibonacciRunner(n-2);

            nminus1.start();
            nminus2.start();

            try{
                nminus1.join();
                nminus2.join();
            }
            catch( InterruptedException e){

            }

            result =  nminus1.getResult()  + nminus2.getResult();

            memo.put(n, result);
        }

        return result;
    }
}
```

Wartezeit = 3

x = BSFPLWB ,

x_{best} = BFLPSWB

q(BFLPSWB) = 4859

Tabu - Liste:

BSLPFWB, 1

BSFWPLB, 2

BSFPLWB, 3

N(x) = BSFWLPB, BFSPLWB

BSFPWLB, BPFSLWB, BSPFLWB

BLPFSWB, ~~BSLPFWB~~, BSFLPWB

BSWLPFB

q(BSFWLPB) = 4847 ✓ 4927

q(BFSPLWB) = 5076 ✓ 5417

q(BSFPWLB) = 5464 ✓

q(BPFSLWB) = 5743 ✓

q(BSPFLWB) = 5535 ✓

q(BLPFSWB) = 4642 ✓

q(BSFLPWB) = 4864 ✓

q(BSWLPFB) = 4526 ✓

~~q(BSLPFWB)~~ ✓

b) je nachdem, wie die Nachbarschaft definiert ist.

Entweder die erste (4847), die beste (4526) oder zufällig eine auswählen.