Labb 3

Du ska analysera de empiriska värdena ovan och din implementation av laboration 3 genom att svara på följande frågor:

1. Baserat på de empiriska värdena ovan, vilken tidskomplexitet verkar de olika algoritmerna i laboration 3 tillhöra? Här är det viktigt att du för varje algoritm även redovisar hur du kommer fram till rimlig tidskomplexitet.
    - NumberSequenceIterative

$O(n)$ är tidskomplexiteten för tiden växer linjärt.

- NumberSequenceRecursive

$O(2^n)$eftersom funktionen fördubblas för varje element i inmatningen.

2. Är tidskomplexiteten baserad på de empiriska värdena ovan rimlig utifrån din implementation av de olika algoritmerna? Motivera ditt svar genom att resonera kring din implementation av algoritmerna.

Iterativ lösning:

```
if (n <= 0) {
        n = 0;
        return n;
    }
    int[] num = new int[(int) (3 + n)];
    num[0] = 0;
    num[1] = 1;
    num[2] = 2;
        for (int i = 3; i <= n; i++){
            num[i] = num[i - 1] + num[i - 3];
        }
    return num[(int) n];
 }
```

Rekursiv lösning:

```
long funk = 0;
 long funk2 = 0;
 long x = 0;
 if(n <= 0) {
    n = 0;
 }
```

```
  else if(n == 1){
      n = 1;
  }
  else if(n == 2){
      n = 2;
  }
  else if (n > 2) {
      x = n - 1;
      funk = calculate(x);
      x = n-3;
      funk2 = calculate(x);
      long sum = funk + funk2;
      return sum;
  }
  return n;
}
```

Tidskomplexiteten från tabellerna skulle jag säga går ihop med min kod. Den iterativa lösningen är ju lineär eftersom den bara går igenom en forloop och gör uträkningen för att hitta rätt nummer. Den rekursiva lösningen hoppar upp i sig själv fler gånger beroende på storleken på n så den ökar för varje gång n ändras. Med det sagt, den rekursiva lösningen ökar exponentiellt så den borde vara $O(2^n)$.