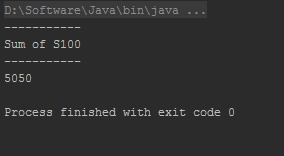
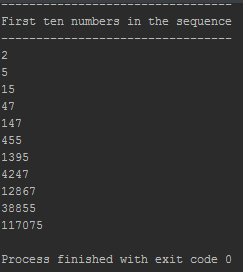
# Oppgave 1

|  |
| --- |
| **Main Klassen**  package Oppgave1.a;  /\*\*  \* Created by Martin on 19.03.2015.  \*/ public class main {   public static void main(String [] args){   Rekursjon r = new Rekursjon();   System.out.println("-----------");  System.out.println("Sum of S100");  System.out.println("-----------");  System.out.println(r.sum(100));   }  }  **Rekursive Java-Metoden**  package Oppgave1.a;  /\*\*  \* Created by Martin on 19.03.2015.  \*/ public class Rekursjon {   public int sum(int n){   int result;  if (n == 1){  result = 1;  } else{  result = n + sum(n-1);  }  return result;   }  } |

**Eksempel på kjøring av kode:**

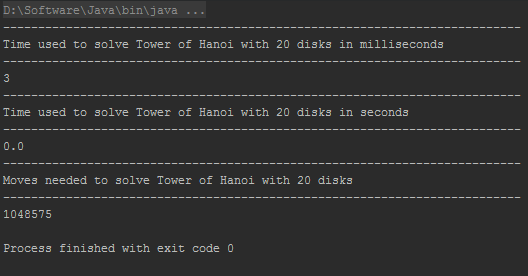
|  |
| --- |
| **Main Klassen**  package Oppgave1.b;  /\*\*  \* Created by Martin on 19.03.2015.  \*/ public class main {   public static void main (String [] args){   Sequence s = new Sequence();   System.out.println("---------------------------------");  System.out.println("First ten numbers in the sequence");  System.out.println("---------------------------------");   for (int i = 0; i < 11; i++){  System.out.println( s.sum(i));  }   }  }  **Rekursive Java-Metoden**  package Oppgave1.b;  /\*\*  \* Created by Martin on 19.03.2015.  \*/ public class Sequence {   public int sum(int n){   int result = 0;   if (n == 1){  result = 5;  } else if (n == 0){  result = 2;  } else if (n > 1){  result = (5 \* sum(n-1)) - (6 \* sum(n-2)) + 2;  }   return result;   }  } |

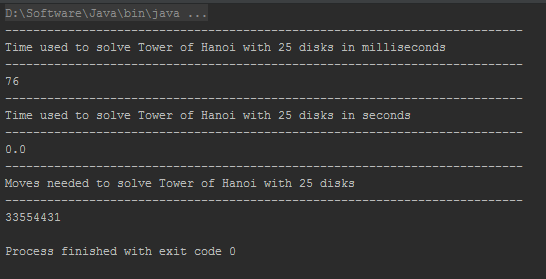
**Eksempel på kjøring av kode:**

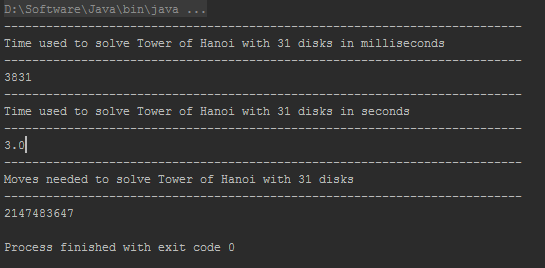


|  |
| --- |
| **Main Klassen**  package Oppgave1.c;  /\*\*  \* Created by Martin on 19.03.2015.  \*/ public class SolveTowers {   public static void main(String [] args){   int disks = 31;   TowersOfHanoi towers = new TowersOfHanoi(disks);   long start = System.currentTimeMillis();  towers.solve();  long elapsedTimeMillis = System.currentTimeMillis()-start;   float elapsedTimeSeconds = elapsedTimeMillis/1000;   System.out.println("--------------------------------------------------------------------------");  System.out.println("Time used to solve Tower of Hanoi with " + disks + " disks in milliseconds");  System.out.println("--------------------------------------------------------------------------");  System.out.println(elapsedTimeMillis);  System.out.println("--------------------------------------------------------------------------");  System.out.println("Time used to solve Tower of Hanoi with " + disks + " disks in seconds");  System.out.println("--------------------------------------------------------------------------");  System.out.println(elapsedTimeSeconds);  System.out.println("--------------------------------------------------------------------------");  System.out.println("Moves needed to solve Tower of Hanoi with " + disks + " disks");  System.out.println("--------------------------------------------------------------------------");  System.out.println(towers.getMovement());  } }  **TowersOfHanoi Klassen**  package Oppgave1.c;  /\*\*  \* Created by Martin on 19.03.2015.  \*/ public class TowersOfHanoi {   private int totalDisks;  private int totalMoves = 0;   public TowersOfHanoi(int disks){  totalDisks = disks;  }   public void solve(){  moveTower(totalDisks, 1, 3, 2);  }   private void moveTower(int numDisks, int start, int end, int temp){  if (numDisks == 1){  moveOneDisk(start,end);  } else {  moveTower(numDisks-1, start, temp, end);  moveOneDisk(start, end);  moveTower(numDisks-1, temp, end, start);  }  }   private void moveOneDisk(int start, int end){  //System.out.println("Move one disk from " + start + " to " + end);  totalMoves++;  }   public int getMovement() {  return totalMoves;  } } |

**Eksempel på kjøring av kode med tre forskjellige eksempler**







For alle disse eksemplene så stemmer formelen med antall flyttinger som er registrert i selve programmet. Når det kommer til tidsforbruket av disse løsningene så hadde det nok vært mer nøyaktig å bruke «System.nanoTime()» istedenfor «System.currentTimeInMillis()». Men tiden er tilnærmet korrekt for disse tre eksemplene.

# Oppgave 2

# Oppgave 3

Binært søk i java kan oppnås ved å lage en rekursiv metode som for eksempel tar inn en heltallsarray, minstelengde på arrayen, makslengden på arrayen og det tallet som man ønsker å finne i arrayen. Algoritmen blir da som følger:

1. Sett variablene min og maks til en int-verdi som blir bestemt av lengden på data-arrayen. I tillegg settes målverdien som ønskes å finnes i arrayen.
2. Et midpunkt som søket begynner ifra blir satt til min-verdien+maks-verdien dele på 2.
3. Man bruker så en if-setning som sjekker om midpunktet i arrayen stemmer overens med målverdien, hvis den gjør dette så skal posisjonen(indeksen) returneres.
4. Derimot hvis den ikke stemmer så skal midtpunktsverdien bli vurdert om den er større eller mindre enn målverdien.
5. Hvis den er større så blir indeksen redusert med 1 og metoden blir kalt igjen helt til målverdien er funnet. Hvis den er mindre så blir indeksen økt med 1 og metoden blir kalt igjen. Hvis målverdien blir funnet så skal posisjonen returneres.
6. Hvis målverdien ikke finnes i arrayen så skal -1 returneres til kallet.

b)