

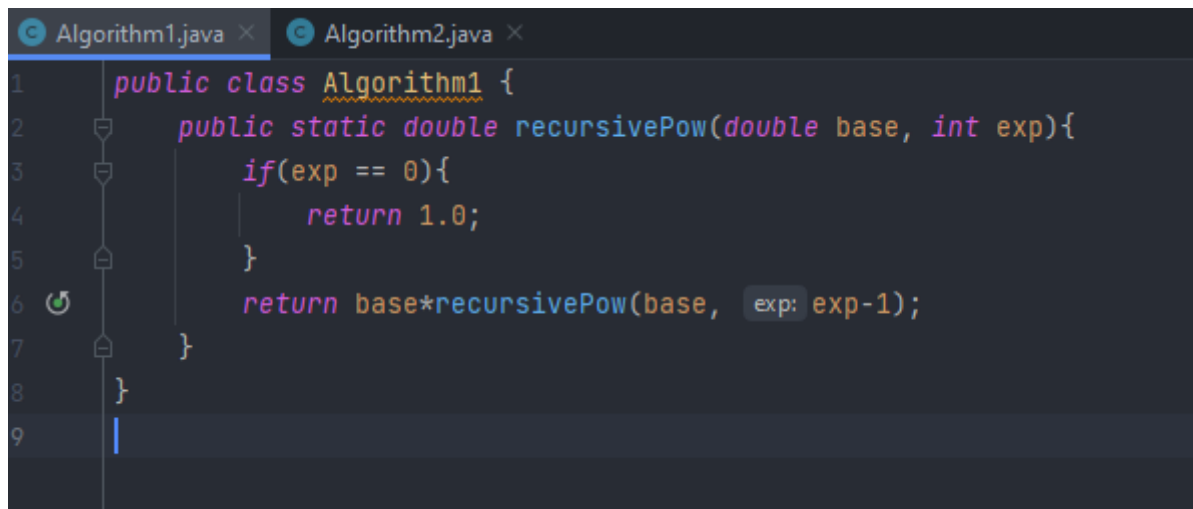
Algoritmer og datastrukturer

Øving 1

Brage Minge

## 2.1-1

På bilde 1 kan du se algoritmen tilhørende oppgave 2.1-1. På bilde 2 og 3 kan du se at den regner riktig



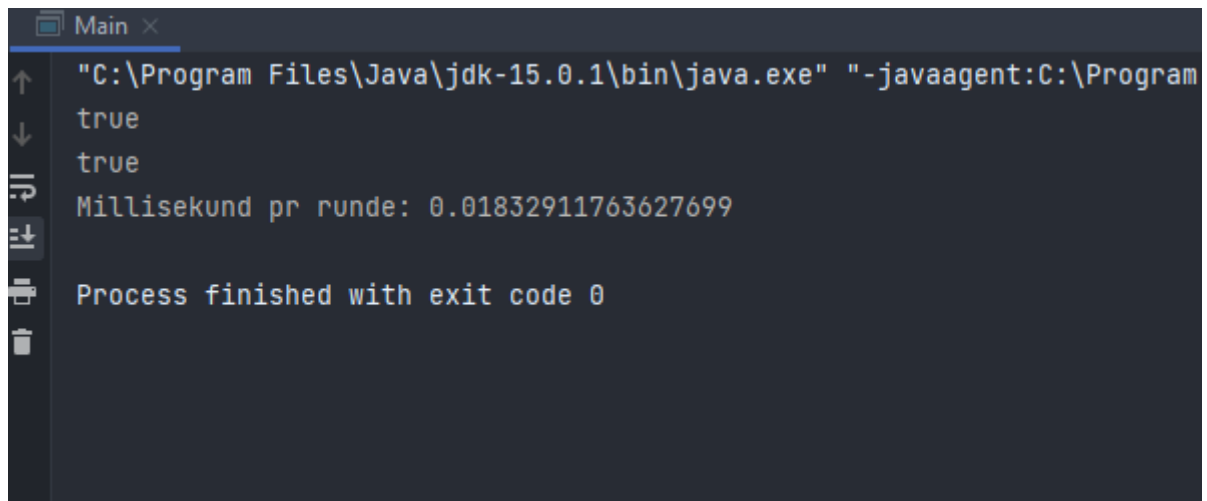
```
1 public class Algorithm1 {
2     public static double recursivePow(double base, int exp){
3         if(exp == 0){
4             return 1.0;
5         }
6         return base*recursivePow(base, exp-1);
7     }
8 }
9
```

Bilde 1



```
1 import java.util.Date;
2
3 public class Main {
4     public static void main(String[] args) {
5         //Klasse for tidtaking og verifisering av algoritme 1
6         Date start = new Date();
7         int runder = 0;
8         double tid;
9         Date slutt;
10
11         System.out.println(Algorithm1.recursivePow(base: 2, exp: 10)==1024.0);
12         System.out.println(Algorithm1.recursivePow(base: 3, exp: 14)==4782969);
13
14         do{
15             Algorithm1.recursivePow(base: 1.001, exp: 5000);
16             slutt = new Date();
17             ++runder;
18         }
19         while(slutt.getTime()-start.getTime()<1000);
20         tid = (double) (slutt.getTime()-start.getTime())/runder;
21         System.out.println("Millisekund pr runde: " + tid);
22     }
23 }
24
```

Bilde 2

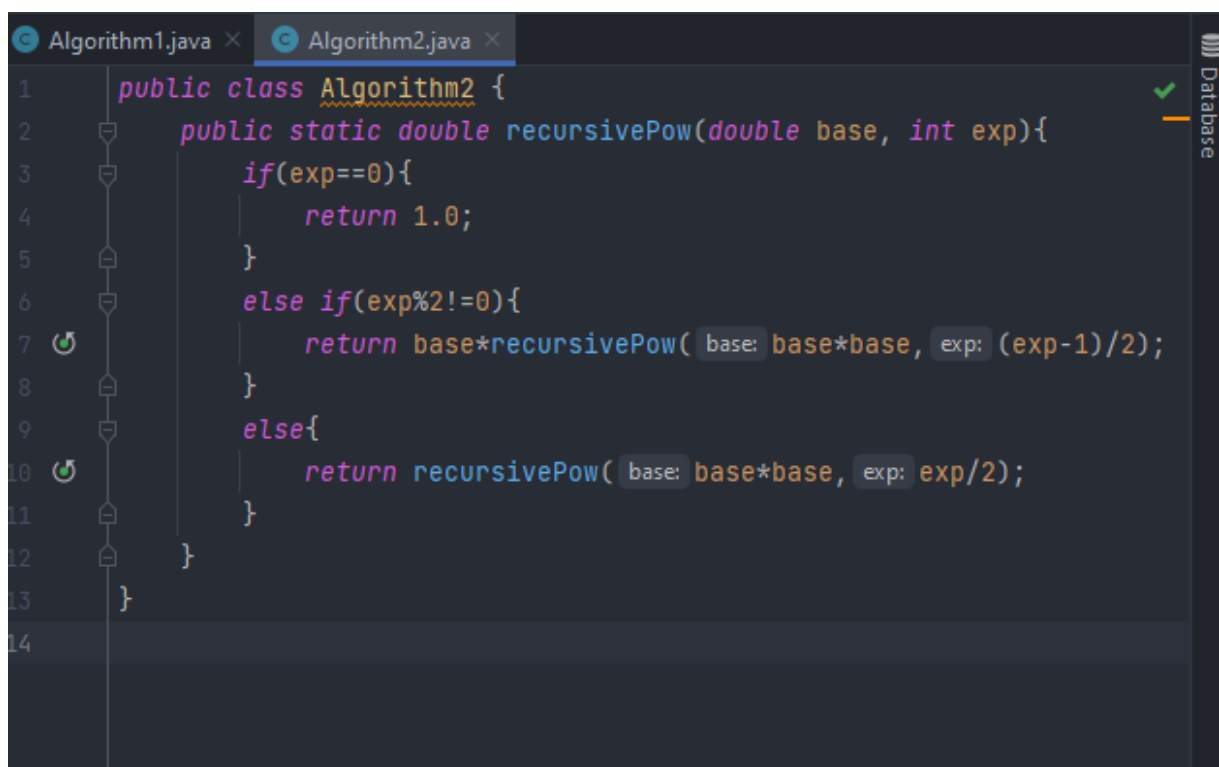


```
Main x
"C:\Program Files\Java\jdk-15.0.1\bin\java.exe" "-javaagent:C:\Program
true
true
Millisekund pr runde: 0.01832911763627699
Process finished with exit code 0
```

Bilde 3

2.2-3

På bilde 4 kan du se algoritmen tilhørende oppgave 2.2-3. På bilde 5 og 6 kan du se at den regner riktig.



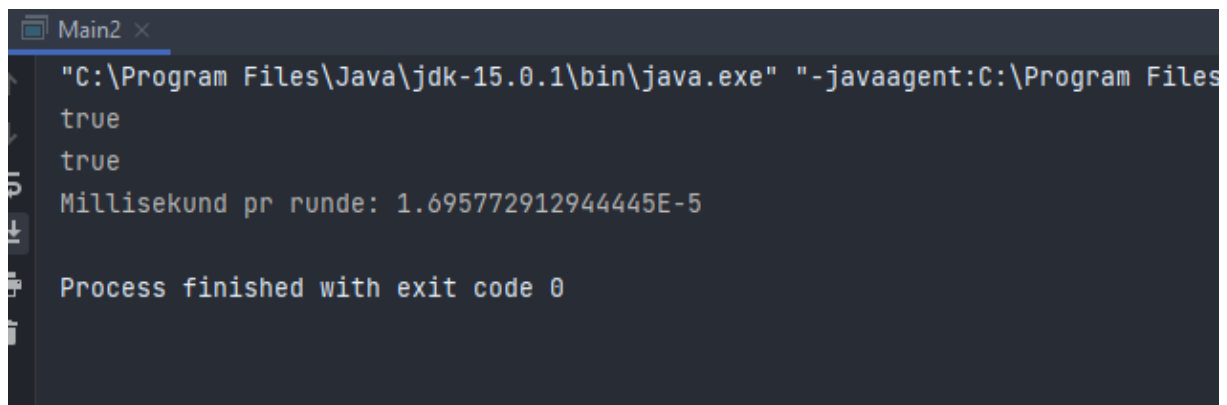
```
Algorithm1.java x Algorithm2.java x
1 public class Algorithm2 {
2     public static double recursivePow(double base, int exp){
3         if(exp==0){
4             return 1.0;
5         }
6         else if(exp%2!=0){
7             return base*recursivePow( base: base*base, exp: (exp-1)/2);
8         }
9         else{
10            return recursivePow( base: base*base, exp: exp/2);
11        }
12    }
13 }
14
```

Bilde 4



```
1  import java.util.Date;
2
3  public class Main2 {
4      public static void main(String[] args) {
5          //Klasse for tidtaking og verifisering av algoritme 2
6          Date start = new Date();
7          int runder = 0;
8          double tid;
9          Date slutt;
10
11          System.out.println(Algorithm2.recursivePow( base: 2, exp: 10)==1024.0);
12          System.out.println(Algorithm2.recursivePow( base: 3, exp: 14)==4782969);
13
14
15          do{
16              Algorithm2.recursivePow( base: 1.001, exp: 5000);
17              slutt = new Date();
18              ++runder;
19          }
20          while(slutt.getTime()-start.getTime()<1000);
21          tid = (double) (slutt.getTime()-start.getTime())/runder;
22          System.out.println("Millisekund pr runde: " + tid);
23      }
24  }
25
```

Bilde 5




```
Main2 x
"C:\Program Files\Java\jdk-15.0.1\bin\java.exe" "-javaagent:C:\Program Files
true
true
Millisekund pr runde: 1.695772912944445E-5

Process finished with exit code 0
```

Bilde 6

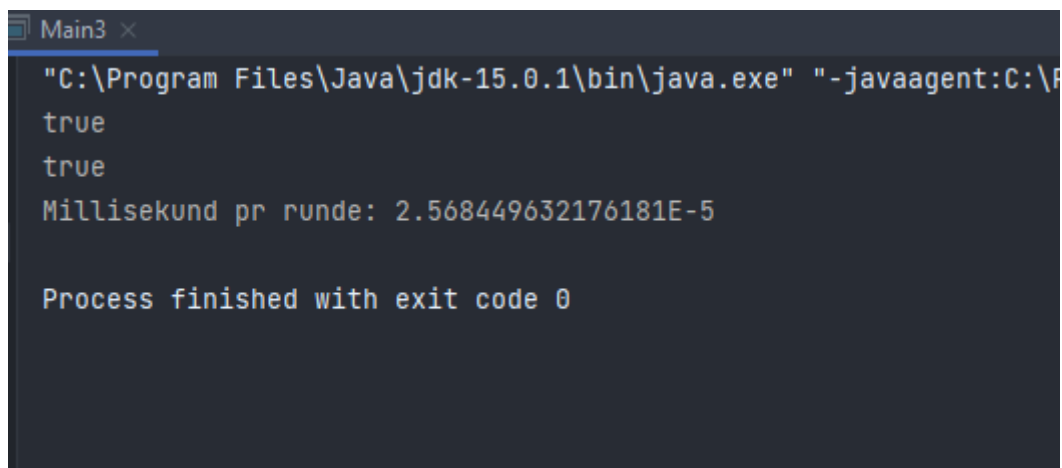
3.

På bilde 7 og 8 kan du se at Math.pow regner riktig og tiden den bruker på å regne ut  $1.001^{5000}$ .



```
1 import java.util.Date;
2
3 public class Main3 {
4     public static void main(String[] args) {
5         //Klasse for tidtaking og verifisering av Javas egen eksponentialfunksjon
6         Date start = new Date();
7         int runder = 0;
8         double tid;
9         Date slutt;
10
11         System.out.println(Math.pow(2,10)==1024.0);
12         System.out.println(Math.pow(3,14)==4782969);
13
14         do{
15             Math.pow(1.001,5000);
16             slutt = new Date();
17             ++runder;
18         }
19         while(slutt.getTime()-start.getTime()<1000);
20         tid = (double) (slutt.getTime()-start.getTime())/runder;
21         System.out.println("Millisekund pr runde: " + tid);
22     }
23 }
```

Bilde 7



```
Main3 x
"C:\Program Files\Java\jdk-15.0.1\bin\java.exe" "-javaagent:C:\P
true
true
Millisekund pr runde: 2.568449632176181E-5

Process finished with exit code 0
```

Bilde 8

Tidtaking.

På bilde 9 er algoritme 1 kjørt med 5000 som  $n$  og på bilde 10 er den kjørt med 10000 som  $n$ . Det kommer tydelig frem av tidene per runde at algoritmens kompleksitet kan oppgis som  $O(n)$ , da en dobling av  $n$  dobler tiden. Det kan man også se av algoritmen, da den vil gå gjennom alle tall fra  $n$  til 0. En annen ting som og er verdt å nevne er at denne algoritmen ikke alltid klarer å kjøre med 10000 som  $n$ , noe som er helt uproblematisk for de andre algoritmene.

```
"C:\Program Files\Java\jdk-15.0.1\bin\java.exe"  
true  
true  
Millisekund pr runde: 0.018360414945377764  
  
Process finished with exit code 0
```

Bilde 9

```
"C:\Program Files\Java\jdk-15.0.1\bin\java.exe"  
true  
true  
Millisekund pr runde: 0.036112816438554046  
  
Process finished with exit code 0
```

Bilde 10

På bilde 11 er algoritme 2 kjørt med 5000 som n og på bilde 12 er den kjørt med 10000 som n. Vi kan se at en dobling av n kun fører til en liten endring i tid. Denne algoritmen er mange titalls ganger mer effektiv enn algoritme 1, og grunnen til dette er at n hele tiden halveres under utregningen.

```
"C:\Program Files\Java\jdk-15.0.1\bin\java.exe" "-java  
true  
true  
Millisekund pr runde: 1.6406635978929088E-5  
  
Process finished with exit code 0
```

Bilde 11

```
"C:\Program Files\Java\jdk-15.0.1\bin\java.exe  
true  
true  
Millisekund pr runde: 1.77742804313633E-5  
  
Process finished with exit code 0
```

Bilde 12

Til slutt har jeg kjørt Math.pow med 5000 som n på bilde 13 og 10000 som n på bilde 14, og man kan tydelig se at algoritme 2 er raskere.

```
"C:\Program Files\Java\jdk-15.0.1\bin\java.exe" "-jav  
true  
true  
Millisekund pr runde: 2.514653894102493E-5  
  
Process finished with exit code 0
```

Bilde 13

```
"C:\Program Files\Java\jdk-15.0.1\bin\java.exe" "-  
true  
true  
Millisekund pr runde: 2.535763971292413E-5  
  
Process finished with exit code 0
```

Bilde 14