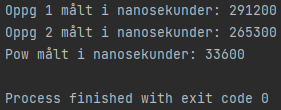
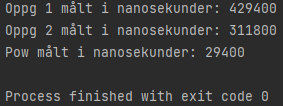
**Rapport øving 2**

Jonathan Løseth & Anders Tellefsen

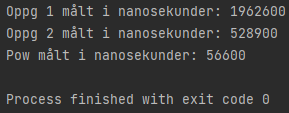
For oppgave 1 fant vi ut at tidskompleksiteten er O(n), grunnet at a = 1 og b = n – 1 = -1 som gir T(n) =theta(n^^log (1)) = O(n). Oppgave 2 har a = 2 og b = 2 som gir T(n) = theta (n^^k \* log(n)) = theta(log(n)). Vi ser at første oppgave har lineær tidsbruk, mens oppgave 2 har logaritmisk. For oppgave 3 tester vi med Math.Pow (Java), som ut fra info vi kan finne bruker O (1) i tidskompleksitet.

Nedenfor er det resultater for diverse tester (Målt i nanosekund).

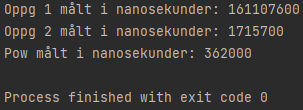
n = 10

n = 100

n = 1 000



n = 10 000



På n = 100 000 fikk vi stack overflow, så der kan vi ikke måle tidsbruk. Vi kan se på tidsmålingene at Math.pow definitivt er den raskeste metoden, som også gir mening med tanke på at tidsbruken skal være O (1). Algoritmen for oppgave 1 er den tregeste, og økte ikke lineært- Tidsbruken for n = 100 og n = 1 000 ble 4,58x høyere for sistnevnte. Hvis vi tar mellom siste testen, ble n = 10 000 82x høyere en n = 1 000. Diagrammet for den ga ikke noe nyttig informasjon.

Under ser du en tabell på tidsbruken.

