Oppgave 1

Tidsmålinger:

Under testing var x = 1.001.

Tid oppgitt i antall nanosekunder

n	Innebygd pow funksjon (ns)	Funksjon oppg 2.1-1 (ns)	Funksjon oppg 2.2-3 (ns)
10	51500	1200	1100
50	51800	1400	1100
250	53100	15200	1100
500	48700	27100	1200
5000	49100	275900	1100

Lav n:

Begge programmene bruker lav tid. Omtrent likt.

Innebygd metode bruker lengre tid.

Høy n:

Program fra oppgave 2.2-3 bruker omtrent like lang tid som med lav n. Konstant.

Program fra oppgave 2.1-1 bruker langt mer tid enn program fra 2.1-1. Øker i forhold n.

Innbygd metode bruker litt mindre tid. Omtrent konstant.

Asymptotisk analyse/årsak:

T(n) = aT(n/b) + cn^k

Oppgave 2.1-1:

```
//Task 2.1-1
double pow1(double x,int n){
    if(n == 0){
        return 1;
    }
    else{
        return pow1(x,n-1) * x;
    }

}
```

Tidsforbruket på programmet vil være lik T(n) = T(n-1) + 1:

Programmet vil alltid utføres rekursivt n-1 ganger, derfor T(n-1). Ingen løkker eller gjennomganger av store datamengder, derfor kan vi si at leddet c*n^k er lik c*n^0. c blir 1, siden den bare gjør en sammenligning; if(n == 0). På grunn av antall rekursive kall i programmet er 1, blir a = 1.

T(n) = T(n-1) + 1, ser da at T(n) ∈ Θ (n), fordi den alltid må utføre funksjonen n-1 rekursive ganger. Beste og verste gjennomkjøring blir alltid lineær (n).

Oppgave 2.2-3:

```
//Task 2.2-3
double pow2(double x, int n){
    if(n == 0) {
        return 1;
    }
    //Even
    else if (n%2 == 0){
        return pow2(x*x,n/2);
    }
    //Odd
    else{
        return x*pow2(x*x,(n-1)/2);
    }
}
```

Programmet vill utføres rekursivt n/2 ganger eller hvis n begynner som et oddetall (n-1)/2. Ingen løkker, derfor kan vi si at leddet cn^k er lik c*n^0. Antall rekursive kall i funksjonen er 1, a = 1.

T(n) = T((n-1)/2) + c*1

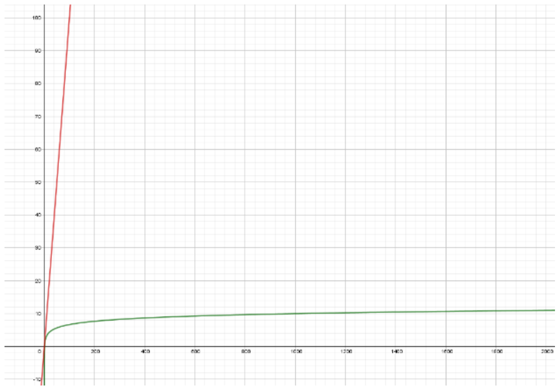
T(n) = T(n/2) + c*1

k = 0, a = 1, b = 2

Mastermetoden:

Hvis b^k = a, har vi T (n) ∈ Θ (n^k · log n) → b^k = 2^0 = 1 = a

Da får vi T(n) ∈ Θ (n^0 * log(n)), som da blir T(n) ∈ Θ(log(n))



Bildet til høyre viser funksjonen f(n) = n (rød) og f(n) = log(n) (grønn). Ved høyere n, vil log(n) utføre langt mindre rekursjoner enn f(n) = n vil gjøre. Log(n) holder seg konstant lav. På grunn av,

$T(n) \in \Theta(\log(n))$,

vil derfor utførelses tiden av programmet fra oppgave 2.2-3 også være konstant lav ved forskjellige verdier av n. Dette er grunnen til at opphøyings programmet i oppgave 2.2-3 vil bruke mindre tid enn opphøyings programmet i oppgave 2.1-1 ved stor n. I tillegg til at tidsforbruket i programmene er omtrent lik.

Kan også tenkes at, om man skulle telle seg ned til null, hadde det gått raskere å alltid halvere tallet man teller ned fra, enn å måtte telle alle tallene i tallrekken nedover. Derfor vil T(n/2) gå raskere enn T(n-1).