|  |
| --- |
| Högskolan Dalarna |
| Laboration 1 |
| GMI24H |

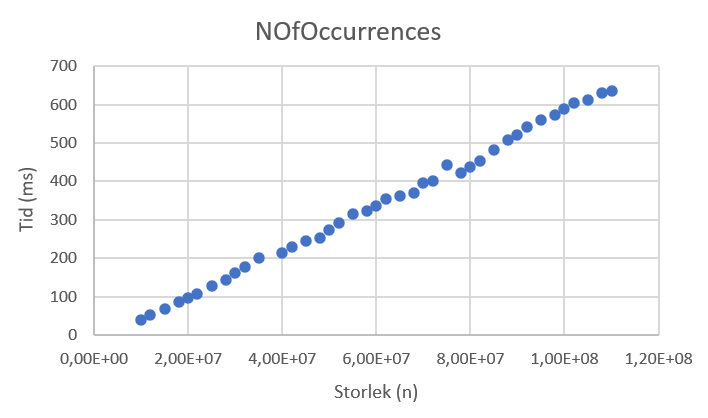
|  |
| --- |
| Laila Assadi  2022-06-09 |

# Uppgift 1

En bild som visar text

Automatiskt genererad beskrivning

Algoritmen beräknar antal gånger ett specifikt tal förekommer i en array. Om arrayen är tom så är x lika med 0. Arrayen ska kunna ta in negativa värden och metoden ska kunna hitta negativa värden. Om det bestämda värdet saknas i arrayen så ska 0 returneras. Algoritmen har tidskomplexitet O(n). Diagrammet nedan visar att om indata fördubblas så kommer tiden också att fördubblas.

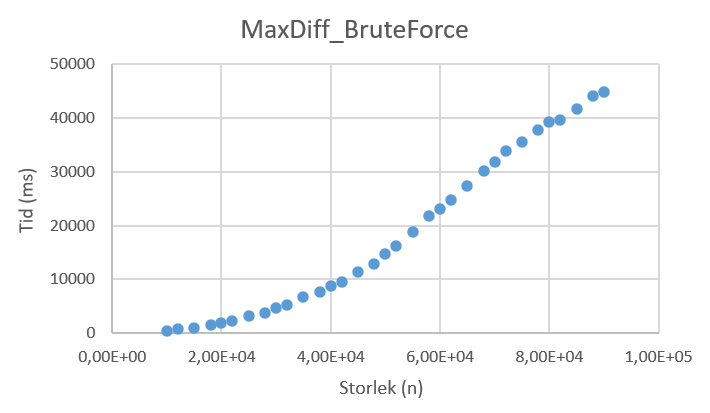


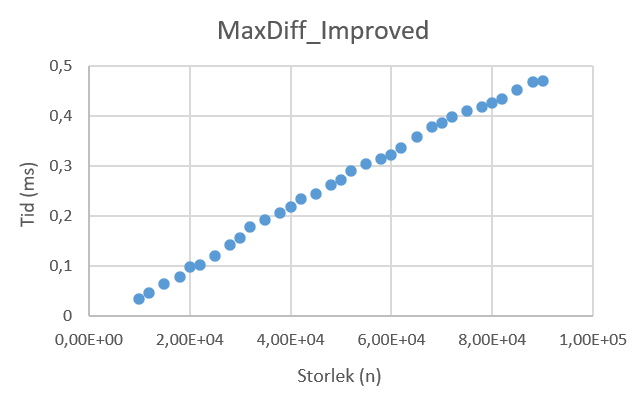
# Uppgift 2

Algoritmen bruteForce har två nästlade for loopar och därför blir tidskomplexiteten T(n) = O(n\*n) = O(n2). Den förbättrade algoritmen har två if-satser inuti en for loop, därför kommer if-satserna att loopa igenom lika många gånger som for-loopen som är n antal gånger. Dess tidskomplexitet är T(x) = O(n).

Om storleken är exempelvis n = 10 så borde bruteforce algoritmen ta 102 = 100 tidsenheter och den förbättrade versionen borde ta 10 tidsenheter.

I bruteforce algoritmen går den yttersta for-loopen igenom arrayen n antal gånger. Den innersta for-loopen exekverar n gånger för varje iteration av den yttersta for-loopen. Detta ger n\*n exekveringar, och därför är dess tidskomplexitet O(n2). Den förbättrade algoritmen har en for-loop och kommer därför bara gå igenom arrayen n antal gånger, och därför är tidskomplexitet O(n). Detta kan man se i graferna nedan.





# Uppgift 3

Algoritmen reversearray har två nästlade while-loopar och dess tidskomplexitet är T(n) = O(n\*n) = O(n2). Den förbättrade versionen har en while-loop och dess tidskomplexitet är T(n) = O(n).

While-loopen i den förbättrade algoritmen kommer gå igenom arrayen så länge i är mindre än j. Det som händer är att metoden byter plats på första och sista elementen i arrayen, sedan byter den plats på andra och näst sista elementen, osv. Detta ger minneskomplexiteten O(1) eftersom algoritmen arbetar ”in-place”.

