O-notasjon, binærsøk, sortering(bubble, selection, insertion, merge)

O-notasjon

Problemstilling oppsummert i en setning: Hvor mange steg trenger man for å løse en algoritme?

Når man snakker om O-notasjon er det ikke at vi snakker om hastigheten men effektiviteten fordi kjøretid kan variere veldig på maskinen man bruker.

Agenda:

-Hva er Big O

-Hvilken kjøretider ser vi mest av?

-Hvordan regne Big O

Hvilken kjøretider ser vi mest av?

O(1) – konstant tid

O(log(n)) – logaritmisk tid

O(n) – lineær tid

O(n log(n)) – lineæritmisk tid / loglineær tid

O(n^2) – kvadratisk tid

O(2^n) – eksponensiell tid

O(n!) – kombinatorisk tid

O(root(n))

Tier list:

S: O(1)

A: O(log n)

B: O(n)

C: O(n log(n))

D:

E: O(n^2)

F: O(n!) , O(2^n)

S er best, goat kjøretid

A er veldig bra

B er bra

C er akseptabelt

E er dårlig

F du fortjene å bli kasta ut av IFI

Eksempel:

Bubble sort = O(n^2)

Binærsøk = O(log(n))

Merge sort, quick sort = O(n log(n))

Lineær søk, traversering av lenkeliste = O(n)

Aksessering index i array = O(1)

Hvordan regne Big O?

-Matematisk

-Big O på kode

Telle steg i kode

Følgende er primitive steg:

-tilordninger, så x = 5

-aritmetiske operasjoner, så x + y

-sammenligninger, så i <= len(liste)

-aksessering på index i arrayer, så Array[i]

-returnering, så return x

Disse koster 1 i tidsbruk, så ganske irrelevant når vi skal ha O-notasjon

En løkke, både While og For-løkke arver kostand fra test og body, altså While test do body, så ganges kostnaden med antall iterasjoner

Hvis vi kaller på metoder arver vi også dennes kjøretid

Ting å se etter:

Iterering av n elementer gir kjøretid n

Løkker ganger vi kjøretiden n med løkken, så nøstet løkker blir n\*n så n^2

Se etter den raskeste voksende delen, T = dn^2 + an + b, her er raskeste dn^2 hvis du ser for deg grafen i hodet

Simplifiser Big O ligningene:

O(n^3 + 50n^2 + 10000)

O((n + 30) \* (n + 5))

O( n\*log(n) + log(n)\*log(n))

O(n + n + n + n + n + n)

Shors algorithm

Euclids algorithm

Binærsøk