# Obligatorisk Oppgave 2

# Algoritmer og Datastrukturer ITF20006

Lars Vidar Magnusson

Frist 27.02.15

Den andre obligatoriske oppgaven tar for seg forelesning 5, 6, og 7 som dreier seg om sortering.

## Praktisk Informasjon

Det er med disse oppgavene dere skal få den nødvendige erfaringen dere trenger, så det kreves at alle innleveringer skal være individuelt arbeid. Det er selvsagt til lov å hjelpe hverandre, men den endelige besvarelsen skal være et resultat av eget arbeid. Besvarelser som viser tegn til plagiering vil bli underkjente.

Dere kan selv velge i hvilket språk dere implementerer algoritmene, men hjelpefunksjoner vil bli gitt i Java. Hvis dere skulle trenge direkte hjelp med koden, kan det være greit å holde seg til enten Java eller C#.

Dere skal lage en enkel rapport som beskriver besvarelsen og samler alle tekstlige svar. Rapporten skal struktureres i henhold til oppgaven. Koden som løser programmeringsoppgavene skal være kjørbar og skal ta en inputfil som argument fra kommandolinjen, for så å skrive ut løsningen. Koden skal leveres i ren kildekode (ikke et prosjekt fra en eller annen IDE med en masse unødvendige filer), og det skal legges ved et enkelt skript som kompilerer koden.

Både rapport, kode og skript skal pakkes i en tarball og skal sendes til meg på lars.v.magnusson@hiof.no med emne "ITF20006 $OPPGAVE $STUDENT".

## Oppgave 1 - QUICKSORT

I denne oppgaven skal dere implementere QUICKSORT og RANDOMIZED-QUICKSORT. Pseudokoden for de to algoritmene finner du under.

```
QUICKSORT(A, p, r)
 if p < r
     q = PARTITION(A, p, r)
     QUICKSORT(A, p, q − 1)
     QUICKSORT(A, q + 1, r)
```

```
PARTITION(A, p, r)
  x = A[r]
  i = p − 1
  for j = p to r − 1
      if A[j] ≤ x
          i = i + 1
          exchange A[i] with A[j]
  exchange A[i + 1] with A[r]
  return i + 1

RANDOMIZED-QUICKSORT(A, p, r)
  if p < r
      q = RANDOMIZED-PARTITION(A, p, r)
      RANDOMIZED-QUICKSORT(A, p, q − 1)
      RANDOMIZED-QUICKSORT(A, q + 1, r)

RANDOMIZED-PARTITION(A, p, r)
  i = RANDOM(p, r)
  exchange A[r] with A[i]
  return PARTITION(A, p, r)
```

Begge algoritmene skal sammenlignes med en sortingsalgoritm med verstefall kjøretid på $O(n^2)$ e.g. INSERTION-SORT og en sorteringsalgoritme med optimal verstefall kjøretid ($\Theta(n \log n)$) e.g. MERGE-SORT.

Sammenligningen skal gjøres ved å teste alle algoritmene med et sett med ulike permutasjoner og inputstørrelser. Det er skal sørges for at både bestefall og verstefall permutasjoner er med i settet som testes. Resultatene skal samles i en tabell og skal plottes i en graf. Diskuter resultatene.

## Oppgave 2 - RADIX-SORT

I denne oppgaven skal dere implementere RADIX-SORT algoritmen. Dette krever også at dere implementerer en stabil sorteringsalgoritme som f.eks. COUNTING-SORT for å sortere tallene på et siffer.

```
RADIX-SORT(A, d)
  for i = 1 to d
      use a stable sort to sort array A on digit i
```

Implementasjonen skal kunne dele tallene som skal sorteres opp i siffer på ulike måter i.e. den skal ikke låses til et tallsystem. I forelesningene diskuterte vi følgende grense $\Theta(\frac{b}{r}(n + 2^r))$ for kjøretiden til RADIX-SORT, hvor $b$ er antall bits per tall i input og $r$ er det valgte antallet bits per siffer. I denne oppgaven kan dere begrense dere til at $b = 32$ siden vi skal sortere integers som normalt sett er 4 bytes store. Dere må tillate at implementasjonen deres kan kjøres med forskjellig valgte verdier for $r$ i.e. utvid algoritmen slik at $r$ kan sendes som et

parameter.

Dere skal teste algoritmen på forskjellige inputstørrelser og med forskjellig verdier for $r$. Resultatene skal samles i en tabell og skal plottes i en graf. Diskuter resultatene.

Lykke til!