# Sortera data

Hiba Qutbuddin Habib

Spring Fall 2022

## Introduktion

I denna rapporten kommer vi att analysera tre olika sorterings algoritm, urvalssortering, insätningssortering samt merge sortering. Alla dessa tre algoritmer har deras fördelar och nackdelar. Vi kommer att analysera tidskomplexitet för dessa algoritmen och utifrån det kommer vi försöka ta reda på skillanden mellan de.

## Urvalssortering

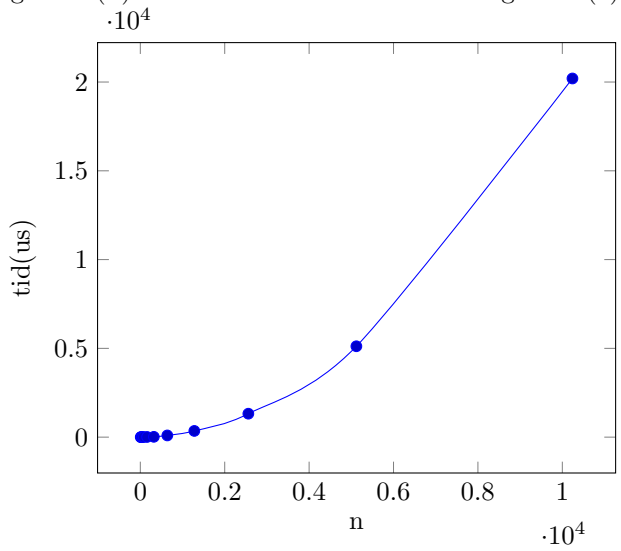
Urvalssortering algoritmen bygger på att först ska det minsta elementet i en vektor hittas sen placerars den i sitt rätt plats, samma process upprepas på alla elementer i en vektor, tills den är klar sorterad. Metoden implementeras enligt följande:

```
for (int i = 1; i < array.length - 1; i++) {
      int cand = i;// väljer en kandidat
      array[i] = array[cand];
      for (int j = 0 ; j < array.length; j++) {
      //om elemtet vid j mindre än Kandidat element
           if (array[j] < array[cand])
                  cand = j;//då har vi en ny kandidat }
      //switch platsen på den gammla och nya kandidat
      int newcand = array[cand];
      array[cand] = array[i];
      array[i] = newcand; }
```

### Resultat

Utifrån grafen nedan kan vi se att urvalssortering algoritm är kvadratisk. Detta eftersom varje gång vi jämför en element så sker det totalt$n$ jämförelse, då varje element jämförs med totalt $n$ element så har vi i slutet en algoritm som tilhör mängden $O(n^2)$. När det kommer till antal element byte så i värsta fall byter $n-1$element platsen då den sista kommer att hamna automatiskt på sin rätta plats, och i bästa fall är sekvensen redan sorterad, och då sker inga element

utbyte. Alltså när det gäller antal utbyte så i värsta fall tillhör algortimen mängden $O(n)$ och i bästa fall tillhör den mängden $O(1)$.




## Insätningssortering

Urvalssortering är en bra algoritm om vi ska sortera korta sekvenser. När sekvenser börjar blir för långt så blir den algoritmen lite oeffektiv. För denna anledning testar vi insätningssortering algoritm effektivtet när det gäller större mängd data i jämförlse med urvalssortering. Insätningssorterings kan beskrivas enligt:

∗Vektor blir från början virtualt uppdelad i en sorterad del och en osorterad del. Elementen från den osorterad del palceras på sin rätt plats i den sorterad delen.

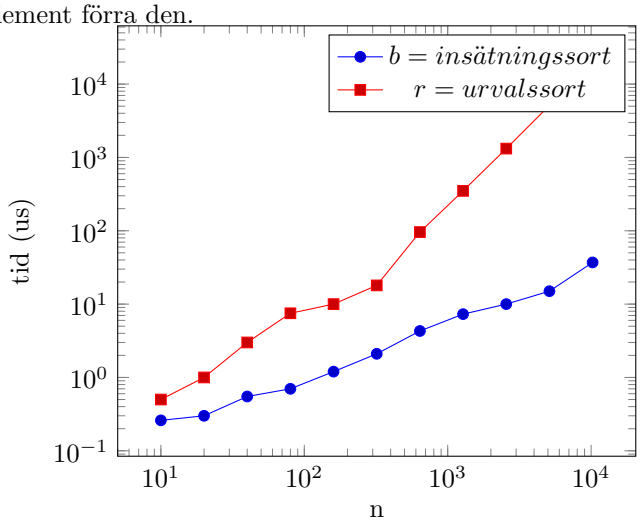
Implementering sker enligt:

```
 for (int i = 0; i < array2.length-1; i++) {
    if (array2[i] > array2[i+1]) {
        int key = array2[i+1];
        array2[i + 1] = array2[i];
        array2[i] = key;}
for (int j = i; j > 0 && array2[i + 1] < array2[i]; j--) {
    array2[j] = array2[j+1]; }
        }
```

### Resultat

Denna algoritmen är kvadratiskt alltså den tillhör mängden $O(n^2)$. Detta eftersom om sekvensen är omvänd sorterad, då behövs varje element jämförs med

de element som är förra den i sekvensen,vilket gör att den sista element jämförs med n-1 elemet, samt eftersom den sista elementet ska egentligen vara den första elementet i sekvensen så sker det totalt, n element utbyte för att den ska hamna på rätt plats. I bästa fall så är sekvensen redan sorterat och då tillhör algoritmen mängden $O(n)$, detta eftersom ingen element utbyte behöver ske, samt när det gäller antal jämförelse så kommer den sista elementet att jämföras med n-element förra den.




Grafen ovan visar resultat av tid mätningar för både urvalssortering samt insätningssortering. Utifrån grafen kan vi se att när mängd data växer, dvs när vektorsstorlek växer så är insätningssortering mycket effektivare, då kan vi tydligt se att mot slutet så tar det mycket kortare tid för insätningssortering jämfort med urvalssortering. Utifrån detta kan vi konstatera att även om både algoritmer tillhör mängden $O(n^2)$ så kan det ena vara effektivare än det andra när det kommer till större mängd data.

# Merge sortering

Denna tredje algoritmen bygger på principen ”söndra och härska”, då merge sortering kan beskrivas enligt följande steg:

1) En sekvens delas i n-lika stora delsekvenser.
2) Både delskvenser slås samman i en sorterad ordning.
3) Andra stegen upprepas tills vi har en enda sekvens kvar.

## Resultat

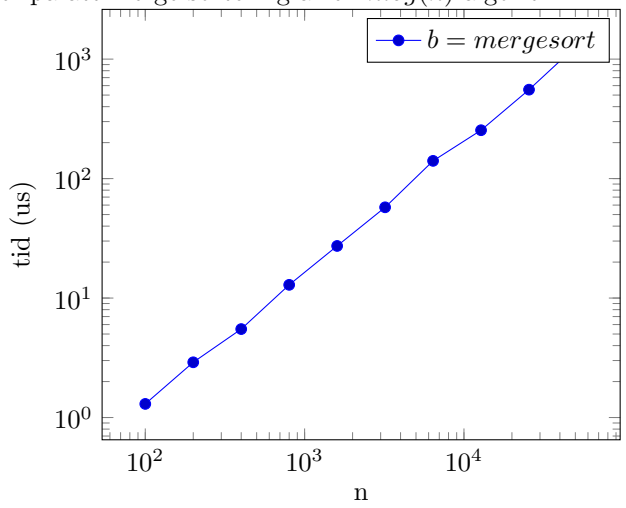
I tabelen samt grafen nedan kan vi se resultat över tid mätning för merge sortering. Om vi bara kollar på grafen så kan vi kanske säga att merge sortering är en linjär algoritm, däremot om vi kollar på tabelen så ser vi tydligare att merge sortering tillhör mängden $n(log(n)$.

| **n** | **100** | **200** | **400** | **800** | **1600** | **3200** | **6400** | **12800** | **25600** | **51200** |
|---|---|---|---|---|---|---|---|---|---|---|
| Tid | 1.3 | 2.9 | 5.5 | 12.9 | 27.3 | 57.5 | 140.7 | 254 | 555 | 1306 |
| t/n | 0.013 | 0.015 | 0.014 | 0.016 | 0.017 | 0.018 | 0.022 | 0.020 | 0.022 | 0.026 |
| t/n(logn) | 0.05 | 0.014 | 0.008 | 0.006 | 0.005 | 0.005 | 0.004 | 0.004 | 0.004 | 0.004 |

Table 1: Tidmätning

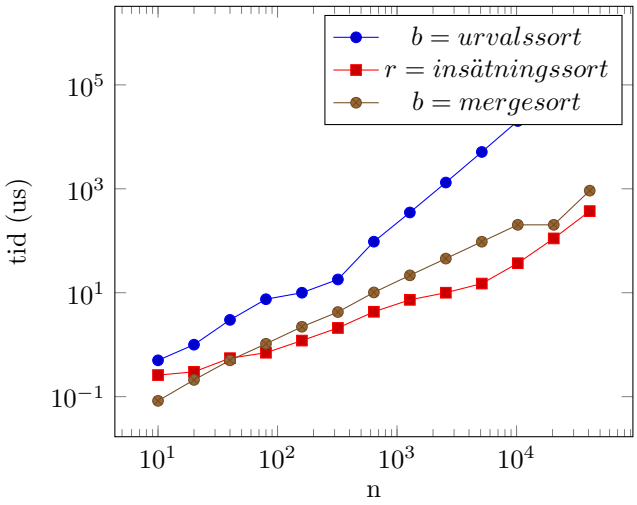
Då värdet av $t/n$ är ingen konstant värde, vilket innebär att linjen har ingen konstant lutning och då denna algoritmen kan ej vara linjär.

Däremot $t/(nlog(n))$ varierar i början för att sedan hålla sig stabil, vilket tyder på att merge sortering är en $nlog(n)$ algoritm.




## Sammanfattning

I en tidigare uppgift har vi diskutera att det finns fördelar med att arbeta med sorterad data. I denna uppgifter har vi analyserat tre olika sorterings metoder, alla har sina fördelar och nackdelar. Vi kom fram till att vissa sorterings algoritmer är effektivare att använda när det är en liten mängd data såsom ”urvalssortering” medans andra är bättre att använda när mängd data blir större såsom ”insätningssortering”. Vi har även analyserat merge sort, som är en effektiv sätt att sortera stora mängd data men nackdelen med den är att den kräver större utrymme jämfört med urvals och insätningssortering , då man utnyttjar flera vektorer, för att bland annat dela sekvensen samt kopiera.




∗I grafen ser vi förhålandet mellan vektor storlek(n) samt tiden för de tre olika sorteringsalgoritmer som representerades ovan.