Försättsblad

Sammanfattning

I denna rapport undersöks exekveringstiden för tre olika sorteringsalgoritmer med samma värsta falls komplexitet vilket i detta fall är ordo(n2). Det används tre olika sorters indata med två olika storlekar, 500 tal och med 2000 tal där alla är heltal men det är olika sorterade från början. Den första är helst slumpgeneraread, den andra är nästan helt sorterad och den sista är nästan omvänt sorterad. Alla algoritmerna körs 600 och ger ett medelvärde av de 600 gångerna ges och den processen görs 100 gånger för att sedan ta ut medelvärdet av de 100 medelvärdena. Sedan jämförs alla resultaten med varandra för att därefter vissa att Bubblesort är den som presterar bäst på alla testerna.

Inledning

Allt jämnt med att tekniken går framåt blir det viktigare för program att blir snabbare och då kan minsta nanosekund vara avgörande för om slutresultatet är bra eller dåligt. Ett sätt att få sitt program att jobba optimalt är att välja rätt sorts algoritm till rätt sorts arbete. Ett exempel kan vara att en sorteringsalgoritm kan vara bra på att sortera indata som är nästan helt sorterad men helt värdelös när det kommer till indata som är helt slumpgenererad, så det gäller att veta algoritmernas styrkor och svagheter när man väljer vilken algoritm man ska använda i programmet. Många gånger ska algoritmen köras flera tusen gånger och då kan valet av algoritm vara avgörande för slutresultatet, om en suboptimal algoritm körs en gång och gör en liten negativ differens i resultatet gör inte så stor skillnad men om den upprepas flera gånger blir den negativa differensen snabbt ett problem. Därför gäller det att man är noggrann när man väljer algoritm för att kunna spara viktiga nanosekunder i slutresultatet.

Bakgrund

Syftet med dessa tester som vi har gjort är för att kunna jämföra hur Bubblesort, Insertionsort och Selectionsort presterar i ett exekveringstidssammanhang för olika former av indata. Anledningen att valet föll på just dessa sorteringsalgoritmer då de har samma värsta falls komplexitet. Den hypotes som vi satte upp i början av laborationen var att Insertionsort och Bubblesort skulle vara likvärdiga i många tester men att Insertionsort skulle vara oftare än inte vara snabbare än Bubblesort[[1]](#footnote-1), till sist skulle Selectionsort vara den som tar längst tid varje gång då den har den värsta normal falls komplexiteten.

Frågeställning

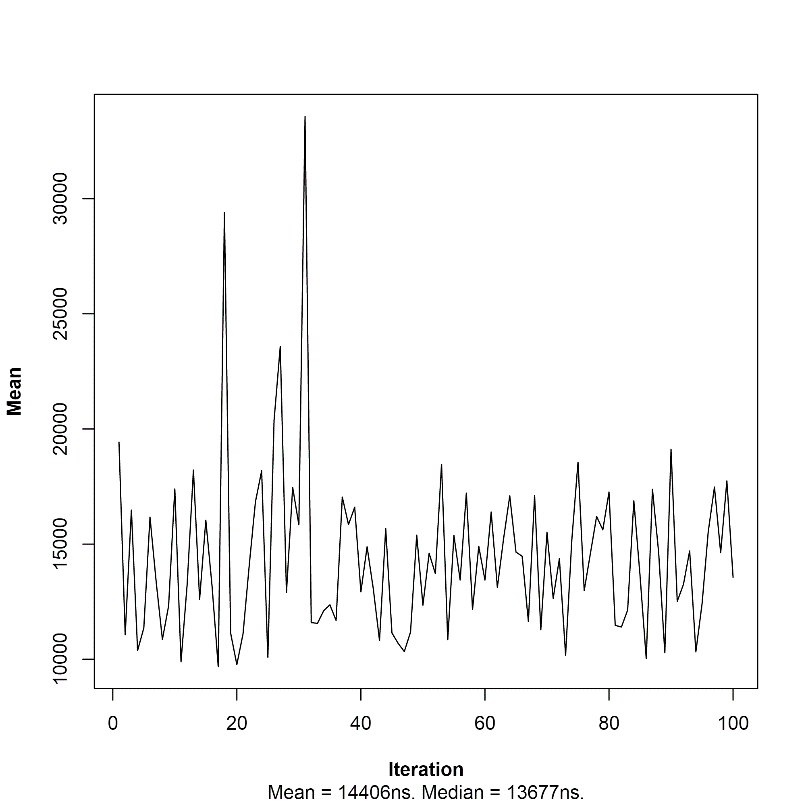
* I vilka fall av indata presterar respektive algoritm bäst?
* I vilka fall av indata presterar respektive algoritm sämst?
* Hur presterar algoritmerna i jämförelse med varandra för motsvarande indata?

Metodbeskrivning

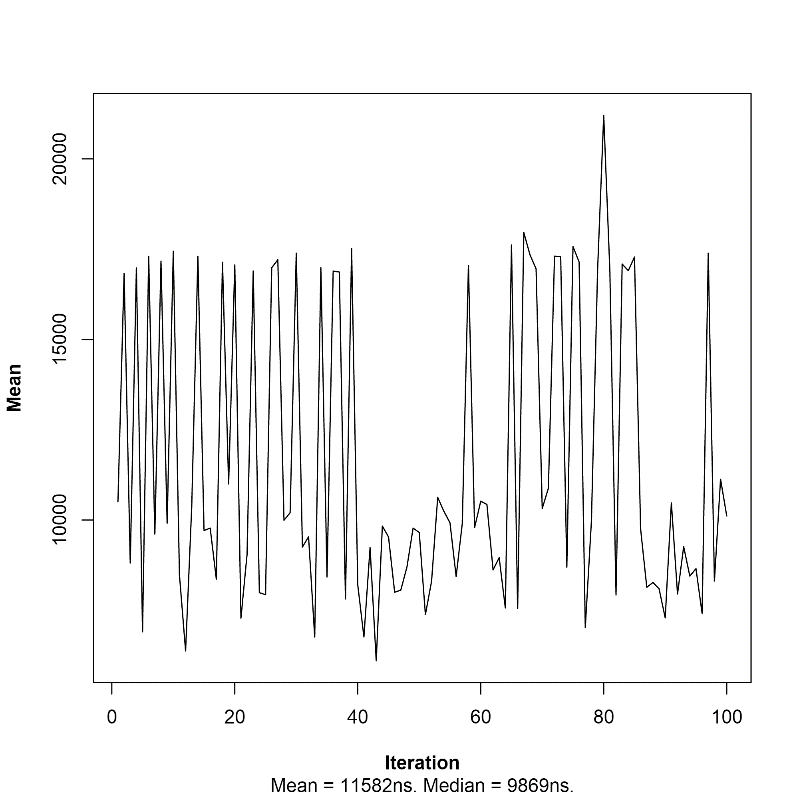
De program som användes för att utföra denna undersökning var Eclipse där alla sorterings algoritmer skrevs i java. Sedan för att kunna köra algoritmerna flera gånger och få ut ett medelvärde så skrevs två program i R med hjälp av Rstudio. **(förklaring av sorterings algoritmerna och hur det skrevs Emil).** För att få fram indatan som skulle sorteras användes först för den slumpgenererade listan ett eget skrivet program i java som hette RandomSorted.java som använde javas Random.nextInt() för att kunna få trovärdigt slumpgenerade tal. Sedan för att få fram de andra två formerna av indata användes liknande program som hette almostNotSortedIndata.java och almostSortedIndata.java och de skapar indata-filer med ett visst antal element (i detta fall 2000 och 500). Den itererar då 500 eller 2000 gånger i en for-loop. I almostSortedIndata.java skapas för varje iteration "i" ett slumptal som kan vara ett värde från "i" till "i+3". Alltså om vi är på iterationen 345 så kommer det slumpas ett tal mellan 345-348 och lägga in det i en fil. I almostNotSorted.java är det istället "i" till "i-3".

När alla förberedelseuppgifter var klara blev undersökningens upplägg sådant att de tre olika algoritmerna vilket var Bubblesort, Selectionsort och Insertionsort kördes 600 gånger och de gjordes på tre olika sorterade listor vilket var en slumpgenerade lista, en nästan helt sorterad lista och en nästan omvänt sorterad lista med två olika storlekar 500 och 2000 vilket i slutändan gjorde att det blev 18 olika mätningar. För varje gång en algoritm har körts 600 gånger så togs ett medelvärde av de körningarna och det gjordes 100 gånger för att kunna få ett medelvärde av medelvärdena. Att köra den 100 gånger gör att risken för outliers minskar vilket ger ett mer säkert resultat.

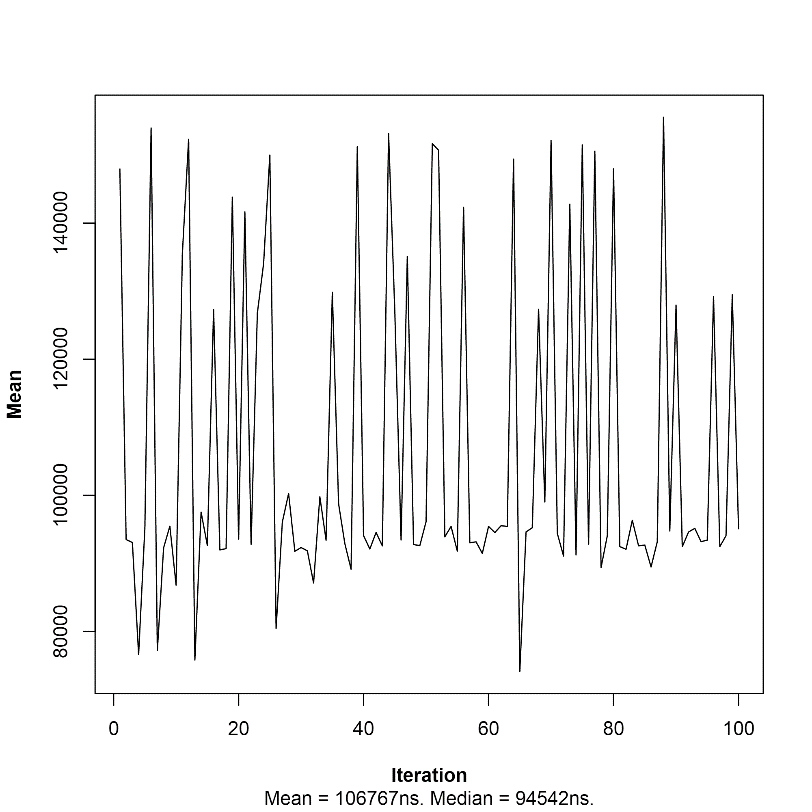
Efter varje körning gjorde en plot av medelvärdena av medelvärdena och medelvärdet av dem räknades ut. I figur 1-3 ned kan man se exempel på plottar från mättningarna som gjordes med 500 element som var nästan sorterad från början.



Figur 1: Plot för medelvärdena av medelvärdena som körts 600 gånger med sorteringsalgoritmen Insertionsort på en lista med 500 element som var nästan helt sorterade.



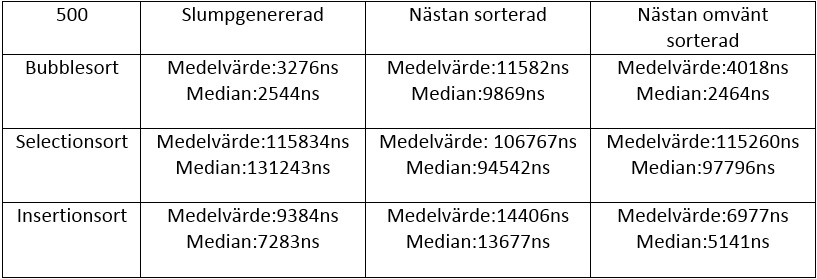
Figur 2: Plot för medelvärdena av medelvärdena som körts 600 gånger med sorteringsalgoritmen Bubblesort på en lista med 500 element som var nästan helt sorterade.



Figur 3: Plot för medelvärdena av medelvärdena som körts 600 gånger med sorteringsalgoritmen Selectionsort på en lista med 500 element som var nästan helt sorterade.

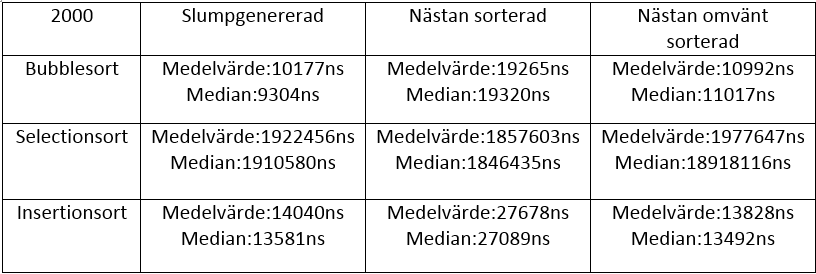
Resultat

Här är en tabell över alla medelvärden och medianer som är framtagna från undersökningen med indata som hade 500 element i sig.



Figur 4: Tabell med medelvärde och medianer för alla tester som gjordes med 500 element.

Detta är en tabell med alla medelvärden och medianer som är framtagna från undersökningen med indata som hade 2000 element i sig.



Figur 5: Tabell med medelvärde och medianer för alla tester som gjordes med 2000 element.

Diskussion

När mätningarna genomfördes var webbläsaren uppe. Detta skulle kunna vara ett validitetshot då det tar extra resurser och kan därför medföra att exekveringstiden ökar. Dock gäller detta för samtliga mätningar vilket därför inte bör spela någon större roll. Majoriteten av mätningarna hade tydliga outliers, vilket syns tydligast i figur 1 där exekveringstiden ökade markant under två mätningar. Webbläsaren skulle kunna spela en roll i detta men då den var konstant uppe och inte användes under själva körningen, är det mer troligt att det är garbage collector som frigör utrymme i minnet vilket får exekveringstiden att öka. Värt att nämna är att outliers vi talar om är ett medelvärde av 600 körningar, alltså måste datorn just under dessa körningar vara extra belastad av externa faktorer.

Det man kan avgöra utifrån resultaten av mätningarna i vårt test är att Bubblesort presterar bäst på alla olika former av indata vilket kan tyckas vara konstigt då de flesta andra källor som jämför sorteringsalgoritmer visar på att Insertionsort skulle vara likvärdig eller bättre än Bubblesort på de flesta formerna av indata. Att Bubblesort blev något snabbare beror högst troligt på att JIT-kompilatorn har lyckats optimera Bubblesort mer än Selectionsort.

Den sorteringsalgoritm som presterar svagast på alla testerna är Selectionsort vilket inte är speciellt konstigt då Selectionsorts värsta falls komplexitet är den samma som Selectionsorts normal falls komplexitet.

Det man kan se att utifrån resultaten är att de olika formerna av indata inte har haft någon större inverkan på slutresultatet då de i förhållande till varandra ger ungefär samma resultat för alla de olika formerna av indata. Att det blir så kan bero på att Insertionsort samt Bubblesort har snarlika tillvägagångssätt att sortera en vektor. Jämförs Selectionsorts körningar på de olika indata (med samma mängd element) så syns det tydligt att sorteringsalgoritmen presterar likadant oberoende av vilken indata. Det är inte förvånansvärt då Selectionsort har bästa-fall-, medel-fall och värsta-fall-komplexitet ordo(n^2).

Slutsats

För att återkoppla till själva forskningsfrågorna kan man utifrån figur 4 och 5 se väldigt tydliga svar på frågorna. Den första frågan var *”I vilka fall av indata presterar respektive algoritm bäst?”* man kan tydligt se att Bubblesort presterade bäst i alla tester som gjordes.

Den andra frågan var *”I vilka fall av indata presterar respektive algoritm sämst?”* det går även i denna fråga se att Selectionsort presterade sämst i alla tester som utfördes.

Den sista frågan var *”Hur presterar algoritmerna i jämförelse med varandra för motsvarande indata?”* här kan man se att i alla olika former av indata är den förhållandevisa skillnaden mellan algoritmerna omkring den samma.

Litteraturförteckning

[1] Martin Höst, 2016, (EDAA35 Utvärdering av programvarusystem), 2016-04-16: 15:10, <http://fileadmin.cs.lth.se/cs/Education/EDAA35/komp/bok.pdf>.

[2] ”Sorting Algorithm Animations”, 2016-04-27, 11:30, <http://www.sorting-algorithms.com/>.

[3] ”Wikipedia”, 2016-04-27: 11:30, <https://en.wikipedia.org/wiki/Bubble_sort>.

[4] ”Wikipedia”, 2016-04-27: 11:30, <https://en.wikipedia.org/wiki/Selection_sort>.

[5] ”Wikipedia”, 2016-04-27: 11:30 [https://en.wikipedia.org/wiki/Insertion\_sort](https://en.wikipedia.org/wiki/Selection_sort).

1. ”Sorting Algorithm Animations”, <http://www.sorting-algorithms.com/>, (2016 April 27, 11:30) [↑](#footnote-ref-1)