Inlämningsuppgift 3

Algoritmer och Datastrukturer

<https://github.com/GoblinDynamiteer/algoDataStruct/tree/master/assignments/ass03>

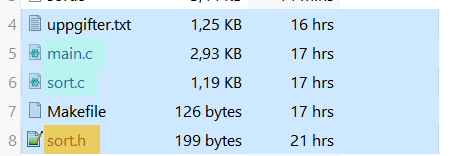
**Johan Kämpe**  
UIS16

2017-02-24

# Svar på deluppgifter:

1. **Ladda ner och packa upp filen metriker.zip från Moodle.**

Svar:



Filer har i efterhand uppdaterats med nya filer, från uppdaterad kod på Moodle.

1. **Kompilera och kör programmet med anropet main test.txt.**

**Finns det felmeddelanden vid körning? Vad skall göras för att få bort dem?**

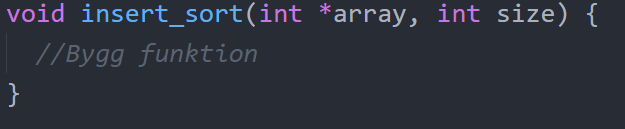
Svar:

Funktionen ***check\_if\_sorted*** i main.c ger returvärde 0 om en array inte blev korrekt sorterad efter ett anrop av en sorteringsfunktion.

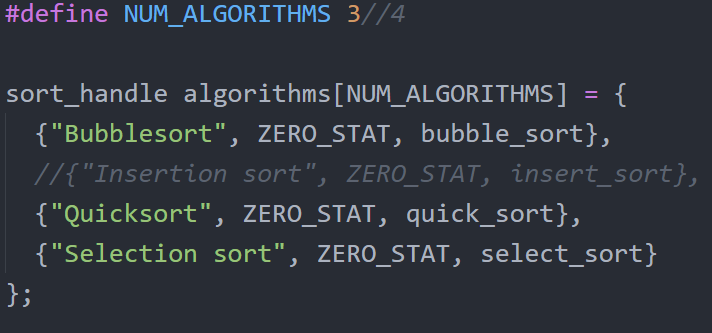
Om detta är fallet skrivs den angivna sorteringsfunktions namn ut. I vårat fall får vi meddelandet: ***ERROR: algorithm Insertion sort doesn't sort the array….***

Felet uppstår därför att funktionen **insert\_sort** anropas, men funktionen innehåller ingen kod, och sorterar således inte arrayen.

Det finns flera sätt att få bort felmeddelandena, men det bästa vore antagligen att implementera **insertion\_sort** i filen *sort.c*.

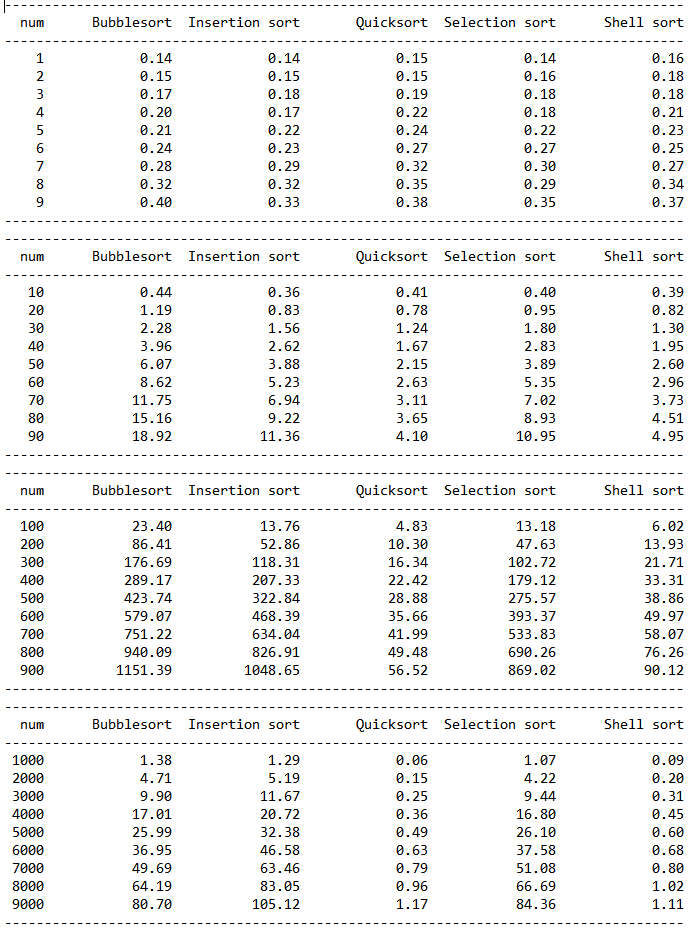


Andra alternativ är att ta bort anropet till ***check\_if\_sorted***, eller att ta bort **insert\_sort** från struktur-arrayen **algorithms** och ändra dess antal medlemmar till 3.



1. **Analysera de tre algoritmerna *bubble sort, quick sort* och *selection sort.* Undersök filen *test.txt* och koden.**
   1. **Har algoritmen enkel kod jämfört med de andra?**
   2. **Anropar algoritmen sig själv?**
   3. **Hur snabb är algoritmen jämfört med de två andra algoritmerna om arrayen har storleken 10? Ange procent!**
   4. **Ange snabbhet likt uppgift 3c, fast för arraystorlek 100.**
   5. **Ange snabbhet likt uppgift 3c, fast för arraystorlek 1000.**

Svar:

Jag kommer använda följande tider för beräkningar i analyserna:

För deluppgifter *a* och *b* i algoritmanalyserna skrivs svaren under var sitt eget kapitel. Jag valde för enkelhetens skull att bunta ihop hastighetsjämförelserna mellan de olika algoritmerna i en tabell. Tabellen finns sist i dokumentet.  
  
Jag valde också att lägga in *insertion sort*, och *shell sort* i diagrammet i uppgift 4.  
  
För uppgift 5 & 7, se bifogad kod.

Analys av bubble sort  
Bubble sort har den kod som ser mest enkel ut enligt mig, jämfört med de andra. Den kan tänkas vara en lämplig kandidat för inbyggda system. Dock, vad jag har förstått, används sällan bubble sort i praktiken, förutom som programmerings-exempel / övningar.

Bubble sort-algoritmen anropar inte sig själv.

Analys av selection sort  
Selection sort har, enligt mig, kod som är mer svårförstådd kod än bubble sort, och enklare kod än quick sort. Överlag känns den relativt enkel dock. Selection sort kan tänkas vara en lämplig sorteringsalgoritm för inbyggda system.

Selection sort-algoritmen anropar inte sig själv.

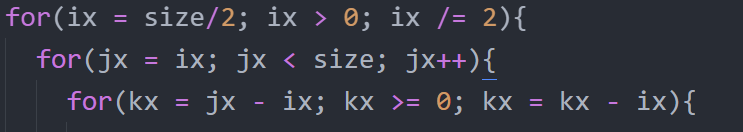
Analys av quick sort  
*Quick sort* har, enligt mig, kod som är mycket mer svårförstådd än de två andra algoritmernas kod. Den använder sig dessutom av två extra stödfunktioner, *\_quick\_sort* och *\_partition.* Quick sort kan eventuellt vara för komplicerad för ett inbyggt system.

Quick sort-algoritmen anropar sig själv.

Analys av insertion sort  
*Insertion sort* har, enligt mig, kod som är ungefär likställt avancerad med *Selection sort*, dvs. svårare än bubble sort, fast enklare än *quick sort*. Insertion sort kan tänkas vara en lämplig sorteringsalgoritm för inbyggda system.

Insertion sort-algoritmen med min implementation anropar inte sig själv.

Analys av shell sort  
*Shell sort* har, enligt mig, mycket mer avancerad kod än *Selection sort*, *bubble sort* och *Insertion sort*. Jag har svårt att avgöra om den känns enklare än koden för quick sort.

Koden som jag har implementerat shell sort från använder sig av tre nästlade for-loopar, vilket jag upplever som komplicerat.  


Shell sort kan eventuellt vara för komplicerad för ett inbyggt system.

Shell sort-algoritmen med min implementation anropar inte sig själv.

Diagram

Hastighetsjämförelser i procent mellan sorteringsalgoritmer

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Hastigheter från test.txt** | | |  |  |  |
|  | Bubble | Insertion | Quick | Selection | Shell |
| **10** | 0,44 | 0,36 | 0,41 | 0,4 | 0,39 |
| **100** | 23,4 | 13,76 | 8,83 | 13,18 | 6,02 |
| **1000** | 1,38 | 1,29 | 0,06 | 1,07 | 0,09 |
|  |  |  |  |  |  |
| **Arraystorlek 10** | |  |  |  |  |
|  | Bubble | Insertion | Quick | Selection | Shell |
| **Bubble** | 100% | 122% | 107% | 110% | 113% |
| **Insertion** | 82% | 100% | 88% | 90% | 92% |
| **Quick** | 93% | 114% | 100% | 103% | 105% |
| **Selection** | 91% | 111% | 98% | 100% | 103% |
| **Shell** | 89% | 108% | 95% | 98% | 100% |
|  |  |  |  |  |  |
| **Arraystorlek 100** | |  |  |  |  |
|  | Bubble | Insertion | Quick | Selection | Shell |
| **Bubble** | 100% | 170% | 265% | 178% | 389% |
| **Insertion** | 59% | 100% | 156% | 104% | 229% |
| **Quick** | 38% | 64% | 100% | 67% | 147% |
| **Selection** | 56% | 96% | 149% | 100% | 219% |
| **Shell** | 26% | 44% | 68% | 46% | 100% |
|  |  |  |  |  |  |
| **Arraystorlek 1000** | |  |  |  |  |
|  | Bubble | Insertion | Quick | Selection | Shell |
| **Bubble** | 100% | 107% | 2300% | 129% | 1533% |
| **Insertion** | 93% | 100% | 2150% | 121% | 1433% |
| **Quick** | 4% | 5% | 100% | 6% | 67% |
| **Selection** | 78% | 83% | 1783% | 100% | 1189% |
| **Shell** | 7% | 7% | 150% | 8% | 100% |

Angivelserna i procent anger den tid det tar för den aktuella algoritmen i vänsterspalten att sortera en array, jämfört med en annan, i den övre raden.

Om talet är 100 % så är de lika snabba (i tabellerna jämförs algoritmen också med sig själv), om talet är under 100 % så är den aktuella algoritmen snabbare. Och vice versa.

Beräkningarna utfördes i Excel, och följande formel användes:  
För *bubble sort* jämfört mot *selection sort*, på arraystorlek 1000: