

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:

 uppgifter.txt	1,25 KB	16 hrs
 main.c	2,93 KB	17 hrs
 sort.c	1,19 KB	17 hrs
 Makefile	126 bytes	17 hrs
 sort.h	199 bytes	21 hrs

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

2. 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 vårt 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**.

```
void insert_sort(int *array, int size) {  
    //Bygg funktion  
}
```

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.

```
#define NUM_ALGORITHMS 3//4  
  
sort_handle algorithms[NUM_ALGORITHMS] = {  
    {"Bubblesort", ZERO_STAT, bubble_sort},  
    //{"Insertion sort", ZERO_STAT, insert_sort},  
    {"Quicksort", ZERO_STAT, quick_sort},  
    {"Selection sort", ZERO_STAT, select_sort}  
};
```

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

Svar:

Jag kommer använda följande tider för beräkningar i analyserna:

num	Bubblesort	Insertion sort	Quicksort	Selection sort	Shell sort
1	0.14	0.14	0.15	0.14	0.16
2	0.15	0.15	0.15	0.16	0.18
3	0.17	0.18	0.19	0.18	0.18
4	0.20	0.17	0.22	0.18	0.21
5	0.21	0.22	0.24	0.22	0.23
6	0.24	0.23	0.27	0.27	0.25
7	0.28	0.29	0.32	0.30	0.27
8	0.32	0.32	0.35	0.29	0.34
9	0.40	0.33	0.38	0.35	0.37

num	Bubblesort	Insertion sort	Quicksort	Selection sort	Shell sort
10	0.44	0.36	0.41	0.40	0.39
20	1.19	0.83	0.78	0.95	0.82
30	2.28	1.56	1.24	1.80	1.30
40	3.96	2.62	1.67	2.83	1.95
50	6.07	3.88	2.15	3.89	2.60
60	8.62	5.23	2.63	5.35	2.96
70	11.75	6.94	3.11	7.02	3.73
80	15.16	9.22	3.65	8.93	4.51
90	18.92	11.36	4.10	10.95	4.95

num	Bubblesort	Insertion sort	Quicksort	Selection sort	Shell sort
100	23.40	13.76	4.83	13.18	6.02
200	86.41	52.86	10.30	47.63	13.93
300	176.69	118.31	16.34	102.72	21.71
400	289.17	207.33	22.42	179.12	33.31
500	423.74	322.84	28.88	275.57	38.86
600	579.07	468.39	35.66	393.37	49.97
700	751.22	634.04	41.99	533.83	58.07
800	940.09	826.91	49.48	690.26	76.26
900	1151.39	1048.65	56.52	869.02	90.12

num	Bubblesort	Insertion sort	Quicksort	Selection sort	Shell sort
1000	1.38	1.29	0.06	1.07	0.09
2000	4.71	5.19	0.15	4.22	0.20
3000	9.90	11.67	0.25	9.44	0.31
4000	17.01	20.72	0.36	16.80	0.45
5000	25.99	32.38	0.49	26.10	0.60
6000	36.95	46.58	0.63	37.58	0.68
7000	49.69	63.46	0.79	51.08	0.80
8000	64.19	83.05	0.96	66.69	1.02
9000	80.70	105.12	1.17	84.36	1.11

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.

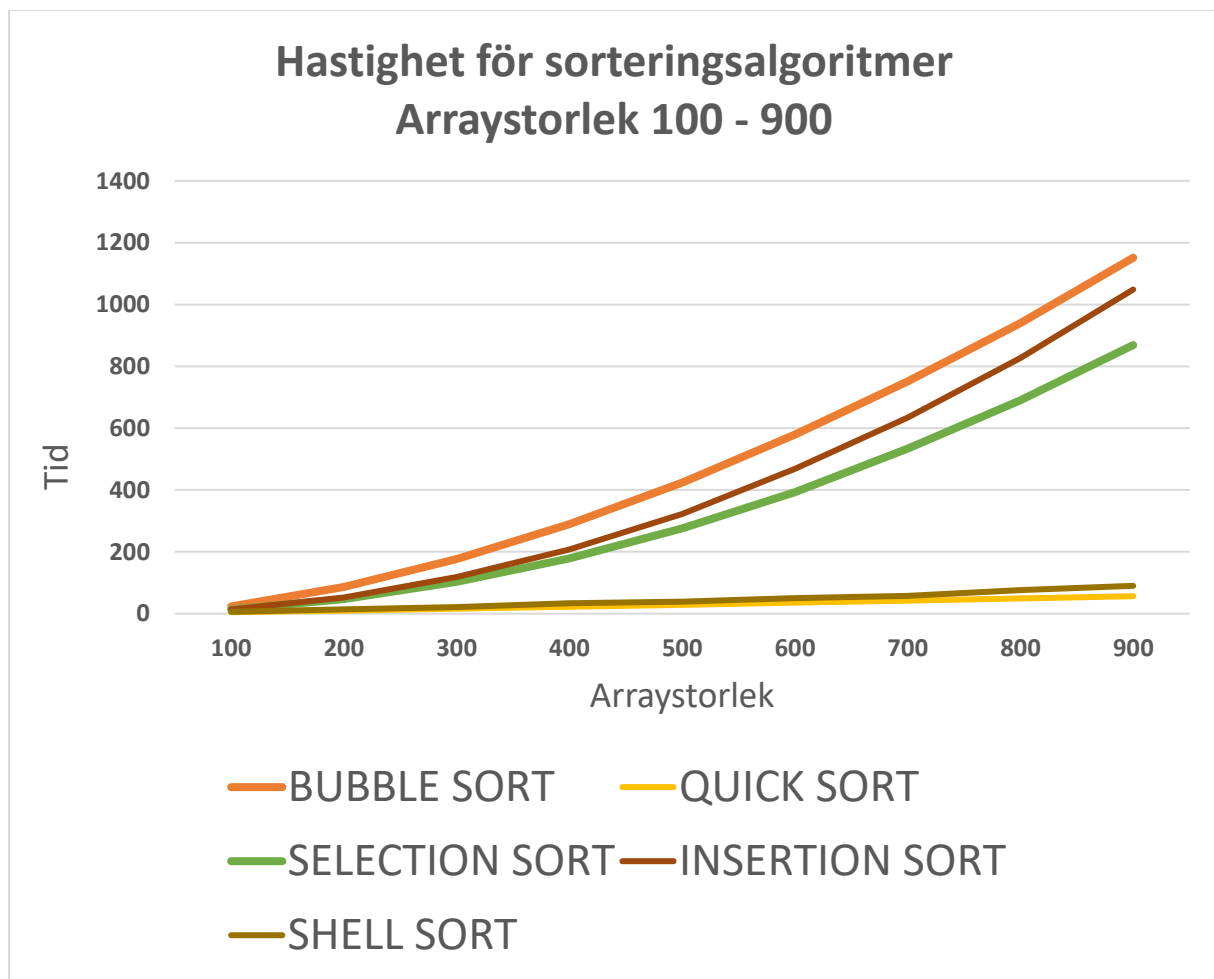
Koden som jag har implementerat shell sort från använder sig av tre nästlade for-loopar, vilket jag upplever som komplicerat.

```
for(ix = size/2; ix > 0; ix /= 2){  
    for(jx = ix; jx < size; jx++){  
        for(kx = jx - ix; kx >= 0; kx = kx - ix){
```

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

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

DIAGRAM



HASTIGHETSJÄMFÖRELSE 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:

$$\left(\frac{1,38}{1,07}\right) * 100 \approx 129\%$$