Uppgift 2: Sorteringsalgoritmer

Klottpapper

Du kan använda textrutan nedan som klottpapper. Du kan ta loss klottpapperet så att det blir en “flytande” ruta med pilknappen i textrutans övre högra kant. Du kan flytta det flytande fönstret från övre balken och ändra dess storlek i nedre högra hörnet. Du kan sedan bläddra fram till uppgiftens frågor utan att klottpapperet försvinner utom synhåll. Genom att stänga klottpapperet returneras det tillbaka rutan under denna instruktion.

Öppna klottpapperet i en annan flik

Tätä voit käyttää suttupaperina

Material

Introduktion till sorteringsalgoritmer

Sorteringsalgoritmer är datorprogram som ordnar en mängd data, t.ex. siffror eller bokstäver, på ett visst sätt i stigande eller fallande ordning. Sorteringsalgoritmer används ofta inom databehandling, statistik och datorprogrammering. Allmänt kända sorteringsalgoritmer är t.ex. samsortering (merge sort), kvicksortering (quick sort), bubbelsortering (bubble sort) och utbytessortering (exchange sort). Varje algoritm har sina egna styrkor och svagheter, så valet beror på mängden data som ska användas och datans karaktär samt de tillgängliga resurserna.

Det kan finnas betydande skillnader i hur snabba sorteringsalgoritmerna är, särskilt för stora datamängder. **Bubbelsortering** och **utbytessortering** är inte lika effektiva som många andra sorteringsalgoritmer, som t.ex. kvicksortering och samsortering, men de är lättare att förstå och kan vara användbara om antalet element som ska sorteras är tillräckligt litet.

När man använder sorteringsalgoritmer lagras de data som ska sorteras ofta i en **tabell**. Tabeller är datastrukturer som används vid programmering och som gör det möjligt att lagra flera värden av samma typ. Tabeller används ofta när det finns ett behov av att behandla en stor mängd data som är ordnade i en viss logisk grupp.

Tabeller består av element, där de alla har sin egen plats i tabellen. Elementens placering definieras med ett tal som kallas index. Det första elementet i en tabell har index 0, det andra har index 1 och så vidare.

Index	0	1	2	3	osv.
Element	12	4	-34	34	...

I programmeringsspråk anges indexen i en tabell vanligen som ett tal som placeras inom hakparenteser. I tabellen ovan har elementet med det första indexet `0` värdet 12. Elementet med det andra indexet `1` har värdet 4 och så vidare.

I följande exempel, som är skrivet i et programmeringsspråk, placeras det första elementet i tabellen i variabeln `tal_ett`. Variabelns värde skrivs sedan ut på skärmen med kommandot `print()`. Du kan prova att skriva ut värdena för de olika elementen i tabellen genom att ändra indexet inom hakparenteserna på rad 2.

Observera att om du refererar till ett index i programmet som inte finns i tabellen (t.ex. `4` eller `-1` i tabellen nedan), kommer programmet att generera ett felmeddelande `undefined`. Detta beror på att referensen ligger utanför intervallet för tabellens index, med andra ord finns inte indexet i tabellen.

Exempel på hur man refererar till ett enskilt element i en tabell

Du kan testa exemplet genom att på rad 2 ändra indexet för det sökta elementet.

```
1 tabell = [12, 4, -34, 34]
2 tal_ett = tabell[0]
3 print(tal_ett) // skriv ut talet 12 om indexet på rad 2 är 0
```

Bubbel- och utbytessortering använder den ovan beskrivna tabellformade datastrukturen som indata. Algoritmen definierar hur tabellens element gås igenom, och slutresultatet är en tabell som är sorterad i stigande ordning.

Bubbelsortering

1. Algoritmen börjar (iteration 1) med att jämföra det första (med index `0`) och det andra (med index `1`) elementet i tabellen. Om de är i fel ordning, dvs. om det första elementet i tabellen är större än det element som det jämförs med, byter de plats med varandra.
2. Algoritmen går vidare till att läsa nästa par i tabellen (det andra och tredje elementet, dvs. index `1` och index `2`, i tabellen) och jämför dem. Om de är i fel ordning byter de plats med varandra. Om man fortsätter på detta sätt (tredje och fjärde elementet, fjärde och femte elementet osv.), går man igenom hela tabellen.
3. Efter det här börjar algoritmen jämföra på nytt från början av tabellen (iteration 2, … osv.) (dvs. första och andra elementet jämförs) och går igenom tabellen igen. Genomgången av tabellen (dvs. iterationerna) upprepas ända tills det inte längre sker något byte under en iteration.

Ett exempel på hur bubbelsorteringen framskrider. På varje rad presenteras tabellens innehåll efter att två på varandra följande element bytt plats. Platsbytet görs endast om elementen är i fel ordning.

Iteration	0	1	2	Förklaring
	18	13	11	Begynnelsetillstånd
1	**18**	**13**	11	Elementen med index 0 och 1 byter plats

Iteration	0	1	2	Förklaring
1	13	**18**	**11**	Elementen med index 1 och 2 byter plats
2	**13**	**11**	18	Elementen med index 0 och 1 byter plats
3	11	13	18	Sorterad tabell, inga byten behöver mera göras

Det gjordes 3 byten.

I fråga 2.2 där det efterfrågas hur motsvarande sortering fungerar, ska svaret i det fallet ges i nedanstående form. Märk att begynnelsetillståndet inte upprepas två gånger som i tabellen ovan.

```
18 13 11
13 18 11
13 11 18
11 13 18
```

Utbytessortering

1. Algoritmen börjar med att i tur och ordning jämföra det första elementet (med index `0`) i tabellen med alla efterföljande element (med index `1..n-1`, där n = antalet element i tabellen). Om de element som ska jämföras är i fel ordning (det för stunden första elementet i tabellen är större än det element som det ska jämföras med), byter de plats med varandra. Efter att ha gått igenom tabellen en gång är det första elementet i tabellen nu det minsta elementet.
2. Algoritmen övergår till att jämföra det andra elementet (med index `1`) i tabellen med de efterföljande elementen (med index `2..n-1`), och elementen byter plats med varandra om de är i fel ordning.
3. Detta fortsätter tills det näst sista och sista elementet i tabellen jämförs. När dessa element har jämförts och vid behov bytt plats med varandra är tabellen i ordning (sorterad).

Ett exempel på hur utbytessorteringen framskrider. På varje rad presenteras tabellens innehåll efter att två på varandra följande element bytt plats.

Iteration	0	1	2	Förrklaring
	18	13	11	Begynnelsetillstånd
1	**18**	**13**	11	Elementen med index 0 och 1 byter plats
1	**13**	18	**11**	Elementen med index 0 och 2 byter plats
2	11	**18**	**13**	Elementen med index 1 och 2 byter plats
3	11	13	18	Sorterad tabell, för att det nästsista och det sista elementet har jämförts

Det gjordes 3 byten.

Bytesoperationen

En central del av sorteringsalgoritmernas funktion är att byta plats på två element i en lista. I exemplet ovan på rad 2, byter de två första elementen (18 och 13) plats i tabellen. I praktiken bör detta implementeras i ett programmeringsspråk så att ett av de två element som ska bytas

lagras i en tillfällig variabel. Nedan visas en bytesoperationen i ett programmeringsspråks kod. I exemplet har tabellen namnet `lista`

```
temp = lista[0]
lista[0] = lista[1]
lista[1] = temp
```

Efterom samma uppgift upprepas flera gånger under sorteringen, gör man ofta så när man programmerar, att man skriver ett eget underprogram (en funktion) för den upprepade uppgiften. I detta fall gör vi det under namnet `swap`. Underprogrammet har givits tre parametrar inom parentes:

- namnet på tabellen som ska behandlas
- indexet för första elementet som ska bytas
- indexet för andra elementet som ska bytas

Då skulle bytet i det föregående exemplet kunna göras med anropet:

```
swap(lista, 0, 1)
```

Observera att `swap` byter sinsemellan plats på elementen oberoende av on de är i ordninig eller ej.

Exempel på underprogrammet swap

Elementen i tabellen `list` med indexen 0 och 1 byter sinsemellan plats. Listan skrivs ut före och efter bytet.

```
1 list = [13, 47, 1]
2 print(list)
3 swap(list, 0, 1)
4 print(list)
```

Frågor

Fråga 2.1 (0-8 poäng)

För varje tabell som anges, hur många gånger utför sorteringsalgoritmerna ovan en operation där elementen byter plats med varandra?

a) 12, 6, 27, -4

Bubbelsortering

Utbytessortering

b) -23, 143, 13, 3, 32

Bubbelsortering

Utbytessortering

c) 143, 45, 32, 11, -10

Bubbelsortering

Utbytessortering

d) 1, 3, 5, 7, 9, 11, 13, 15, 19, 21

Bubbelsortering

Utbytessortering

Fråga 2.2 (0-10 poäng)

Fråga 2.2.1 – Bubbelsortering (0-4 poäng)

Sortera nedanstående heltalstabell i stigande ordning med hjälp av **bubbelsortering**. Skriv dina svar steg för steg i textfältet enligt exemplet i materialet, så att en rad innehåller tabellens tillstånd efter varje **byte**. På den andra raden skriver du tabellens tillstånd efter det första bytet, på den tredje raden tillståndet efter det andra bytet osv.
Den första raden innehåller färdigt tabellens begynnelsetillstånd: `42 52 87 23`
Kom ihåg att även lämna detta begynnelsetillstånd i ditt svar.

```
42 52 87 23
```

Fråga 2.2.2 - Utbytessortering (0-6 poäng)

Sortera den nedanstående heltalstabellen i stigande ordning med hjälp av **utbytessortering**. Skriv ditt svar steg för steg i textfältet, så att en rad innehåller tabellens tillstånd efter varje **byte**. Skriv alltså tabellens tillstånd efter första bytet på den andra raden, tillståndet efter andra bytet på den tredje raden osv.
Den första raden innehåller färdigt tabellens begynnelsetillstånd: `154 143 85 45 32 13`

8
Kom ihåg att även lämna detta begynnelsetillstånd i ditt svar.

```
154 143 85 45 32 13 8
```

Fråga 2.3 (0-12 poäng)

Skriv med hjälp av kodredigeraren nedan ett program som sorterar listan på första raden i stigande ordning med den angivna sorteringsalgoritmen. Du kan lägga till kommandon till koden med hjälp av knapparna under redigeraren. Kommandot `swap()` byter plats på listans element för de index som parametrarna anger. När du löser uppgiften får du endast använda kommandona som anges i knapparna. Om du vill stryka ett kommando som du lagt till, stryk radens innehåll på normalt sätt som i en textbehandlare. **Det får finnas endast ett kommando på varje rad**.

Exempel:

```
lista = [1, 9, 7]
swap(lista, 0, 2)     //  nu är listan [7, 9, 1]
```

Om du vill göra "anteckningar" åt dig själv, kan du göra det med `//`-strängen som i exemplet ovan.

Med kommandot `print(lista)` kan du skriva ut listan. Användningen av `print`-raderna påverkar inte poängsättningen av uppgiften.

Fråga 2.3.1 Bubbelsortering (0-4 poäng)

Skriv ett program som ordnar listan i stigande ordning med **bubbelsortering**.
Med kommandot `print(lista)` kan du skriva ut listan som den ser ut då.
Listans begynnelsetillstånd: 13, 47, 1

```
lista = [13, 47, 1]
```

Fråga 2.3.2 Utbytessortering (0-8 poäng)

Skriv ett program så att listan ordnas i stigande ordning med hjälp av **utbytessortering**.
I denna fråga **får du inte använda** `print()`-kommandot.

```
lista = [12, -2, 1, 10, 8]
```