Lab 4: Skriptprogrammering och testning

Denna laboration har två mål: att bekanta sig med skriptspråk i allmänhet och skriptning för testning i synnerhet, samt att utföra ett enklare blackbox-test av existerande program.

Arbeta gärna i grupper om 2 personer, men inte fler.

Deltagande är *obligatoriskt* och skall redovisas för en handledare!

Bug hunt

Skapa en arbetskatalog `lab4` i din `ioopm`-katalog. Checka ut programmen `sort1` till och med `sort6` från Mercurial-repositoryt och lägg dem i `lab4`-katalogen. Samtliga program läser in ett antal heltal och sorterar sedan dessa enligt olika sorteringsalgoritmer. Syftet med labben är att hitta buggar som har smugit sig in i programmen, samt att se hur de olika programmen beter sig med olika storlek och sortering av testdata genom att skriva enkla Pythonskript som genererar indata och kör programmen.

Vid redovisningen *skall* ni kunna besvara följande:

1. Vilken är det generellt långsammaste programmet?
2. Vilket program är snabbast vid minst 100 000 element?
3. Vilket program är snabbast vid max 10 000 element?
4. Vilket program är snabbast av 1 och 4 med nästan helt sorterat indata (t.ex. saknad front)?
5. Vilket program är snabbast av 1 och 4 med indata sorterat i fallande ordning?
6. Ange minst ett viktigt gränsvärde för något av programmen 1, 5 och 6.
7. Hitta minst ett fall då något program ger felaktigt utdata.
8. Identifiera minst en krasch.

Utöver denna lista innehåller programmen flera kända och möjligtvis flera okända fel. Se hur många du kan hitta! Endast fel som uppstår för korrekt indata betraktas som faktiska fel och programmet skall ”arbeta ostört”. Med korrekt indata menas att det indata man testar med följer specifikationerna, att alla indata är giltiga heltal (att skapa overflow genom att skicka in heltal som är för stora för att lagras i en vanlig C-`int` räknas alltså inte).

Uppgiftsbeskrivning

De program som skall testas läser indata från ”standard in” enligt följande:

1. Ett heltal på första raden som representerar hur många heltal som skall läsas in.
2. Efterföljande rader skall ha ett heltal per rad, och antalet rader måste vara minst det antal man angett på första raden (överflödiga rader ignoreras).

Programmen skriver sedan ut sina respektive resultat till ”standard out”, d.v.s. samtliga angivna heltal i stigande ordning skrivs ut på terminalen.

Exempel på körning:

```
$foo> sort
3
2
3
1
# nedanstående rader är programmets output
1
2
3
```

Testa själv för att se hur programmen beter sig!

Att testa manuellt som ovan är inte bara felbenäget utan oerhört tidsödande och chansen att man lyckas hitta alla buggar är liten – i synnerhet om man behöver testa extremt stora mängder data. (Det finns knappast någon som skulle vilja sitta och mata in en lista med flera hundra tusen slumpade tal manuellt. Och ännu färre som kan mata in flera hundra tusen tal korrekt.)

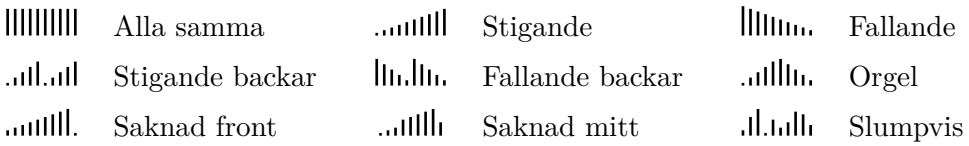
Lyckligtvis kan man i de flesta terminaler omdirigera standard in och standard out till att läsa från, respektive skriva till filer. Man kan alltså t.ex. skriva:

```
myprogram < indata.txt > utdata.txt
```

varvid `myprogram` läsa raderna i filen `indata.txt` som om de skrevs in i terminalen av användaren och skriva till `utdata.txt` i tron att det är terminalen. I Unix-liknande system är filerna alltid textfiler. Filändelsen spelar ingen roll.

Steg ett

Ett lämpligt första steg är att skriva ett Pythonskript som kan generera lämplig testdata. Detta ger inte bara konsekvent testdata för alla program, utan ger också bättre kontroll över datan, speciellt om man vill göra förändringar i datat vid ett senare tillfälle. Kom ihåg att sorteringsalgoritmer kan bete sig olika beroende på hur slumpad ordningen på talen är. Normalt är data i en applikation inte slumpvist ordnat, se t.ex. figur 1 för olika sätt att ordna datat vid testning.



Figur 1: Olika distribution vid testning. Linjernas höjd avser sorteringsordning.

Skriptet skall skapa en fil enligt specifikation för indata ovan. För en helt slumpvis ordnad lista räcker det med att använda funktionen `shuffle(list)` i modulen `random`, men detta är som sagt bara en möjlig distribution för indatat.

Vidare kan du vara hjälpt av att kunna skicka argument till Pythonskripten från kommandoprompten, vilket fungerar inte helt olikt C. I modulen `sys` finns en lista som heter `argv` (vilket borde låta bekant). Det första argumentet som skickas in ligger på plats 1 (plats 0 är reserverad för annan information), andra på plats 2 osv.

Om du i stället för att ha en ”hårdkodad” lista vill slumpa fram en ny lista vid varje körning kan du använda funktionerna `random` och `seed` i modulen random. Ett lämpligt värde på seed är t.ex. `time.time()`. *Vid testning av testprogrammet kan det dock vara bra att använda ett konstant seed-värde! Varför?*

Programmet bör även spara en lista med det förväntade resultatet (enligt specifikationen ovan). Denna kan sedan användas för jämförelse med resultatet från de program som skall testas.

Börja med att skriv ett pythonskript som skapar testdata enligt följande input till skriptet:

```
makeinput 0 100 1000
```

Detta skript skall skapa en fil med väntad utdata (en ordnad version av listan) samt en fil för vaje distribution av indatan som skall testas med 1000 element i intervallet 0 till 100. Välj lämpliga filnamn själv.

Börja med att lösa uppgiften för slumpvis ordnat data och utöka sedan programmet till att hantera även andra datadistributioner (se diagrammen). Att testa alla program för samtliga distributioner tar dock ganska lång tid. Av den anledningen är det tillåtet att begränsa mängden tester till *två* olika distributioner *utöver* slumpvis ordnad lista, saknad mitt eller front, samt fallande ordning (dessa behövs för att kunna besvara frågorna längre fram). Du måste kunna motivera din begränsning vid redovisningstillfället.

Kontrollera att du kan generera filer enligt specifikationerna, i alla fall med slumpvis ordnade listor, innan du går vidare!

Steg två

När du nu har redskapen för att genomföra tester är det dags att skriva ytterligare ett Pythonskript för att automatisera själva testningen. Visar det sig att det behövs ny eller ytterligare testdata kan detta ordnas med en enkel förändring av skriptet som genererar testdata (eller bara en enkel körning av det med ny input). Ju mer ditt testdata täcker upp desto bättre. Detta skript skall alltså göra följande:

1. Generera testdata för programmen.
2. För varje program, köra samtliga tester, logga eventuella fel vid körningen, samt jämföra faktiskt resultat med det förväntade.
3. Skriva ut testresultatet som en ”rapport” med antal körda test, antal passerade test, antal misslyckade test, vilka test som misslyckades, samt programmets output för dessa test, t.ex.:

   ```
   Tests:  54
   Passed: 54
   Failed: 0
   ```

Notera att det inte är ett krav att samtliga testfiler är genererade med ett skript. Några filer är enklare att skriva själv. Väldigt små filer som testar väldigt specifika saker kan skrivas för hand.

Vid redovisningen kommer handledaren att fråga efter körningar med testdata utefter en eller flera distributioner och med olika längd, så det är viktigt att dina program är så pass generella att det enkelt går att ordna med kommandoradsväxlar. Exempelvis kan handledaren fråga

efter rapporten för samtliga test, samt en körning av samtliga program för testdata i fallande ordning med 100 000 tal.

Allmänna tips när ni arbetar

Var inte rädd för att söka information på Internet. Att kasta sig in i ett nytt språk kan verka överväldigande, men det är oftast enklare än man tror om man bara vågar pröva i den interaktiva prompten och googla.

För att du skall komma igång följer här en kort lista över några saker du eventuellt kommer att ha användning av:

- Vilket kommando som helst som kan köras på kommandoraden kan köras med hjälp av raden `os.system('mittkommando')`, givet att man importerat modulen os.
- Konvertering mellan string och int kan lätt göras med `str(minint)` och `int(minstring)`.
- Strängar och listor kan enkelt konkateneras med +.
- Glöm inte att testa programmen med extrem indata av olika slag som t.ex höga värden, låga värden, få värden, inga värden etc. Detta är ett bra sätt att ta reda på hur programmen faktiskt beter sig och det är ofta vid sådanna fall man hittar verkliga problem.
- Skriv gärna ut och fyll i tabellen nedan. Använd flera om det underlättar.

Distribution	Storlek	Program	Tid	Notering