Inlämningsuppgift A – Python edition:

Analys av tre jämförelsebaserade sorteringsalgoritmers egenskaper och körtider

Uppgiften ska utföras individuellt och består av två delar som examineras stegvis:

1) En förberedande teorigenomgång som examineras med ett individuellt kunskapsquiz (se mer information på Canvas om quizet).

2) Implementation och analys av tre olika sorteringsalgoritmer som examineras baserat på inlämnad programkod och rapport (se vidare instruktioner nedan).

Förberedande teorigenomgång

I den förberedande teorigenomgången ska kapitel 2, 3, 6, 7 samt kapitel 8.1 i kursboken[1] läsas igenom noggrant. Fokus ska vara på att förstå hur de jämförelsebaserade sorteringsalgoritmerna som kursen tar upp i ovan nämnda kapitel fungerar, centrala designval i implementationen av dessa samt viktiga, relaterade aspekter såsom algoritmernas tids- och minneskomplexitet. För att förstå hur effektiv en sorteringsalgoritm är och hur dess körtid T(n) växer som en funktion av indatans storlek (dvs. antal element, n, som ska sorteras) är det även viktigt att förstå grundläggande komplexitetsteori och den asymptotiska notation som behandlas i framför allt kapitel 3 i kursboken. Läs även noga igenom de delar som behandlar ”worst-case analysis”, ”best-case analysis” samt ”average-case analysis” av vissa algoritmer i kapitel 2 och kapitel 6, 7 och 8.1 för att förstå skillnaden mellan dessa och på vilka olika sätt indata kan påverka körtiden.

Implementation och analys av tre olika sorteringsalgoritmer

1) Implementation

Du ska självständigt implementera ett program som läser in slumpvis genererade heltal. Programmet ska sedan kunna sortera dessa givna datamängder med hjälp av tre olika sorteringsalgoritmer som du har implementerat själv från scratch. Dessa tre algoritmer är 1) Quicksort, 2) Heapsort och 3) Insertionsort. Pseudokod som beskriver dessa sorteringsalgoritmer finns i kursboken och naturligtvis kan du utgå från den.

[1] Cormen T.H., Leiserson C.E., Rivest L.E., Stein C. (2009), *Introduction to Algorithms (3e upplagan)*, MIT Press.

För ytterligare instruktioner gällande implementation och testning av algoritmerna, se avsnittet ”Instruktioner för inlämning av programkod och rapport” nedan samt ta del av de olika frågor och svar finns kursens forum i Discord.

2) Mätning och analys av algoritmernas faktiska och förväntade körtider

- Kör ditt program och mät körtiden T(n) för n = 1000 samt n=10 000 för samtliga tre algoritmer som du har implementerat. Säkerställ att du inte har en mängd andra onödiga processer som körs på din dator, som kan påverka körtiderna och dina mätningar negativt. Dokumentera dina resultat.
- Använd teorin från framför allt kapitel 3.1 i kursboken och antagandet om att det finns teoretiska undre och övre gränser för en algoritms körtid, T(n). Reflektera över om dina uppmätta körtider speglar ”worst-case”, ”average case” eller ”best case”.
- Anta sedan att T(n) = O(f(n)), vilket medför T(n) ≤ c*f(n) för värsta fallet. Beräkna värden på faktor ’c’ för resp. algoritm, baserat på dina mätvärden av T(1000) och T(10 000). Du ska själv anta lämplig funktion f(n) för dina beräkningar av värden på ’c’ samt kunna förklara och motivera detta.
- Beräkna baserat på samma formel som ovan förväntade körtider T(n) för n = 50 000 och n = 100 000. Dokumentera dina beräkningar, antaganden och resultat.
- Testa därefter dina prediktioner av förväntade körtider genom att låta ditt program sortera de givna datamängderna bestående av 100 000 element minst fem gånger per algoritm och dokumentera uppmätta körtider T(n), för n = 100 000. Ange alltid enhet. Återigen, säkerställ att du inte har en mängd andra onödiga processer som körs på din dator, som kan påverka körtiderna och dina mätningar negativt.
- Analysera dina resultat och observationer med koppling till teorin.

3) Dokumentation och redovisning av genomfört arbete

Du ska dels lämna in din programkod (se instruktioner i nästa kapitel), dels självständigt dokumentera ditt arbete i en skriftlig rapport som ska vara 3–4 A4-sidor exklusive bilagor. Brödtexten ska ha 12pt Times New Roman. Ange namn, studentakronym, kurskod samt sidnummer på varje sida av rapporten. Rapporten ska innehålla följande delar med rekommenderad typ av innehåll:

a) Beskrivning av algoritmerna och implementationen av dessa

Baserat på den teorigenomgång som gjorts i deluppgift (1), ska en tydlig och korrekt beskrivning av de tre implementerade algoritmerna göras. För varje algoritm ska dess huvudsakliga egenskaper, styrkor och svagheter beskrivas. Varje påstående ska styrkas genom referens till kurslitteraturen (inkl. sidhänvisning). De val som gjorts inför implementationen av respektive algoritm ska också tydligt beskrivas motiveras. Vill du bifoga en kort och tydlig beskrivning av respektive algoritm ska det göras i form av pseudokod[2] och läggas som bilaga.

b) Tillvägagångssätt

[2] Vänligen följ rekommenderade principer i Kap. 3 https://www.kth.se/social/upload/524b352ff2765470ed21f75a/pseudokod.pdf

Redogör tydligt och kortfattat för hur du har gått tillväga för att lösa uppgiften självständigt i olika steg (t.ex. vilket material och vilka källor har du utgått från och varför, hur skedde implementationen och testningen, hur planerades och genomfördes experimenten inkl. vilka antaganden gjordes, i vilken datormiljö gjordes experimenten, etc).

c) Resultat, analys och slutsatser

I rapporteringen av dina mätningar, beräkningar och analyser ska du både redogöra för dina resultat och dina numeriska beräkningar. Det vill säga, du ska redovisa dina uppmätta körtider T(n) liksom beräknade värden på 'c', T(50 000) samt T(100 000) för respektive algoritm och underliggande numeriska beräkningar och antaganden.

Mätvärden och andra resultat ska sammanställas i tabeller med tydliga rubriker. Detaljerade numeriska beräkningar ska redovisas separat i en bilaga.

Redogör avslutningsvis för dina reflektioner och slutsatser från uppgiften som helhet genom att tydligt svara på frågeställningarna nedan med återkoppling till kurslitteraturen:

- Hur presterade dina implementerade algoritmer i förhållande till varandra och överensstämmer dina resultat med vad du förväntade dig, baserat på kurslitteraturens beskrivning av algoritmerna och dina val av implementationer?

- Hur förhåller sig dina prediktioner av förväntad körtid T(100 000) med resultaten från dina mätningar? Hur skulle prediktionerna kunna förbättras?

- Vilken algoritm av de tre du implementerat är lämpligast att använda för att sortera 1000, 10 000, 50 000 resp. 100 000 element?

- Finns det skillnader i algoritmernas beteende vid tester på nästan sorterad data?

d) Källförteckning

e) Bilagor

Instruktioner för inlämning av programkod och rapport

1. Skriftlig rapport

Rapporten ska lämnas in separat från programkoden och i form av ett pdf-dokument. Det vill säga, inlämningen av rapporten sker på ett särskilt ställe i Canvas och endast pdf-dokument accepteras där. Både inlämning av rapport och kod måste göras för att inlämningen ska betraktas som fullständig och kunna bedömas.

2. Programkod

Din implementation **ska** bestå av tre filer (**quicksort.py**, **heapsort.py** och **insertionsort.py**). Dessa filer ska ha ett lint-score av minst 8.0 (av10) vid körning med **pylint**.

Till din hjälp har du fått en fil **test_sort.py** som ligger i en zippad mapp. Filen test_sort.py innehåller funktioner för att testa dina sorteringsmoduler. Där testas både att de sorterar rätt, lint-score och tidskrav.

Lägg filen i samma mapp som dina Python-filer med dina sorteringsfunktioner. Testet kommer att köra din implementation med filer som innehåller 1000, 10 000, 50 000, samt 100 000 element. Testerna gör även en tidsmätning. Tänk på att det finns många faktorer som kan påverka körtiden. Mätningar bör därför utföras flertalet gånger och ett medelvärde tas fram för att få mer rättvisande data. Din implementation **måste** korrekt kunna sortera alla fallen innehållande 1000 samt 10 000 element inom rimlig tid (mindre än 5 sekunder per fil och sorteringsalgoritm) för att bedömas.

I mappen finner du också en undermapp ”TestData” som innehåller ett antal olika filer med osorterade heltal som används av testet och som även kan användas vid egna tester och/eller mätningar. Dessa filer är namngivna på formen

”AntalHeltal(MinimumTal,MaximumTal).txt”

Exempelvis filen ”10000(0,100000).txt” innehåller 10000 heltal som är minst 0, och maximalt 100000, som inte ligger i någon specifik ordning.

I filen **test_sort.py** finns testdatafilerna arrangerade i tre olika listor. En lista med ett grundset av randomiserade heltalsfiler med 1000 eller 10 000 element, ett set med större filer som omfattar 50 000 eller 100 000 (tar längre tid att testa) och ett set med nästan sorterad data (1 000 – 50 000 heltal). Du kan aktivera de olika testfilerna genom att toggla kommentarer kring de olika versionerna av konstantvariabeln **TEST_DATA_FILES**.

Mätningarna sparas i en fil som du själv väljer namnet på via inmatning körningen.

I din inlämning av programkod **ska** du ha en .zip fil med följande filer i

- **quicksort.py**
- **heapsort.py**
- **insertionsort.py**

Notera att inlämningen av programkoden sker på ett särskilt ställe i Canvas, och rapporten på ett annat ställe. Båda inlämningarna måste göras för att inlämningen ska betraktas som fullständig och kunna bedömas.

Testerna är utprovade på skolans datorer i salarna H320-H322. Om du har svårt att passera alla tester på din egen dator kan du koppla upp dig mot en av skolans datorer och köra dem där.

Länk till access av skolans datorer i datorsalarna: https://vacantcomp.student.bth.se/