Linköpings Tekniska Högskola
Institutionen för Datavetenskap
Tommy Färnqvist

8 januari 2014

TDDI16 Datastrukturer och algoritmer
Tentamen (TEN1)
2014-01-08, 8–12 (T1, T2)

Examinator:	Tommy Färnqvist
Jour:	Tommy Färnqvist (telefon 070 4547668).
Max poäng:	22 poäng (betyg 5 = 19p, 4 = 15p, 3 = 11p)
Hjälpmedel:	INGA HJÄLPMEDEL TILLÅTNA!!!

VÄNLIGEN IAKTTAG FÖLJANDE

- Lösningar till olika problem skall placeras enkelsidigt på separata blad. Skriv inte två lösningar på samma papper.
- Sortera lösningarna innan de lämnas in.
- MOTIVERA DINA SVAR ORDENTLIGT: avsaknad av, eller otillräckliga, förklaringar resulterar i poängavdrag. Även felaktiga svar kan ge poäng om de är korrekt motiverade.
- Om ett problem medger flera olika lösningar, t.ex. algoritmer med olika tidskomplexitet, ger endast optimala lösningar maximalt antal poäng.
- SE TILL ATT DINA LÖSNINGAR/SVAR ÄR LÄSBARA.
- Lämna plats för kommentarer.

Lycka till!

1. För var och en av funktionerna nedan, ange den komplexitet (A–J) som bäst matchar dess exekveringstid. (3 p)

```
1. public static int f1 (int n) {
      int x = 0;
      for (int i = 0; i < n; i++)
         x++;
      return x;
   }

2. public static int f2 (int n) {
      int x = 0;
      for (int i = 0; i < n; i++)
         for (int j = 0; j < i*i; j++)
            x++;
      return x;
   }

3. public static int f3 (int n) {
      if (n <= 1) return 1;
      return f3(n-1) + f3(n-1);
   }

4. public static int f4 (int n) {
      if (n <= 1) return 1;
      return f4(n/2) + f4(n/2);
   }

5. public static int f5 (int n) {
      if (n <= 1) return 1;
      return f1(n) + f5(n/2) + f5(n/2);
   }

6. public static int f6 (int n) {
      // 1<<i is the same as 2 to the power of i.
      // Ignore integer overflow.
      // 1<<i takes constant time.
      for (int i = 0; i < n; i=1<<i);
   }
```

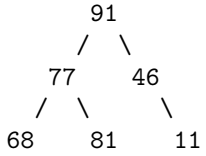
(A) Konstant (D) n (G) n^3 (J) $n!$

(B) $\log^* n$ (E) $n \log n$ (H) 2^n

(C) $\log n$ (F) n^2 (I) 3^n

2. Heapar (3 p)

(a) Ge två olika skäl till varför följande binära träd inte är en heap: (1)



(b) Antag att vi vill skapa en heap som innehåller nycklarna `D A T A S T R U C T U R E`. (Alla jämförelser använder alfabetisk ordning.) Visa den resulterande min-heapen om vi bygger upp den med successiva anrop till `insert()` (med start i `D`). (1)

(c) Antag att vi vill skapa en heap som innehåller nycklarna `D A T A S T R U C T U R E`. (Alla jämförelser använder alfabetisk ordning.) Visa den resulterande min-heapen om vi bygger den från botten och upp (med metoden som fungerar i linjär tid). (1)

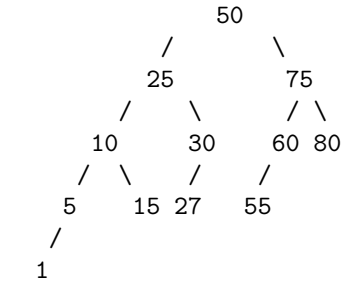
3. Sökträd (5 p)

(a) Betrakta följande sorteringsalgoritm. Sätt in varje element i indatasekvensen, ett i taget, i ett vanligt binärt sökträd. Utför sedan en traversering i inorder av trädet och skriv ut elementen i den ordning de träffas på. Vad är tidskomplexiteten i värsta fallet för algoritmen? Vad är tidskomplexiteten i bästa fallet? (2)

(b) Beskriv hur trädet i (a) kan modifieras så att algoritmen uppfyller följande egenskap: (1)

- den exekverar i $O(n \log n)$ tid i värsta fallet

(c) Visa hur följande AVL-träd ser ut efter operationen `remove(80)`: (2)



4. Algoritmkonstruktion (5 p)

(a) Ge en algoritm för att hitta den största *anagrammängden* i ett givet språk. En anagrammängd är en mängd av ord sådana att alla ord är varandras anagram. Till exempel är $\{\text{tars}, \text{arts}, \text{rats}, \text{star}\}$ en engelsk anagrammängd. (2)

Din algoritm ska ha exekverings tid $O(NL^2)$, där L är längden av det längsta ordet i språket och N är antalet ord i språket. Du får anta att du har en textfil med alla ord i språket. Du behöver inte beskriva hur man läser in textfilen och de behöver heller inte räkna in tiden för att läsa filen i din exekveringstid. Beskriv vilka datastrukturer du använder och hur du konstruerar anagrammängden. Förklara varför din algoritm har tidskomplexitet $O(NL^2)$.

(b) Ge en algoritm för att hitta den längsta *anagramstegen* i ett givet språk. En anagramstege är en sekvens av ord sådana att ord $k+1$ är ett anagram av ord k plus någon bokstav i språkets alfabet. Till exempel är "to, lot, lost, toils, tonsil, lotions, colonist, locations, coalitions, dislocation, conditionals, consolidation, consolidations" en engelsk anagramstege. (3)

Din algoritm ska ha exekverings tid $O(NL^2)$, där L är längden av det längsta ordet i språket och N är antalet ord i språket. Du får anta att du har en textfil med alla ord i språket. Du behöver inte beskriva hur man läser in textfilen och de behöver heller inte räkna in tiden för att läsa filen i din exekveringstid. Beskriv vilka datastrukturer du använder och hur du konstruerar anagramstegen. Förklara varför din algoritm har tidskomplexitet $O(NL^2)$.

5. Sortering (4 p)

(a) Om vi startar med sekvensen `6 1 14 10 5 12 11 9`, vilken/vilka av nedanstående sorteringsalgoritmer träffar på sekvensen `1 5 6 10 14 12 11 9` under sorteringsprocessen? (2)

- insertionsort
- selectionsort
- quicksort (med elmentet längst till vänster som pivot)
- mergesort
- heapsort

(b) När du plötsligt vaknar en natt ser du en tydlig väg till rikedom och berömmelse. Du ska bygga en robotnoshörning och åka runt i landet och sjunga sånger om naturen för barn som kommer att tillåtas leka och interagera med noshörningen. Eftersom en riktig noshörning skulle vara för farlig tänker du dig att en robotnoshörning är lättare att hålla koll på. I var och en av situationerna nedan, vilken sorteringsalgoritm (A-D) skulle du använda? I varje fall kan du anta att minneskapacitet inte är något problem och att målet är att minimera exekveringstiden så att noshörningen kan reagera så snabbt som möjligt vid eventuella problem. (2)

- Noshörningen är utrustad med ett stort antal sensorer som var och en genererar objekt av typen Observation. Typen Observation har flera instansvariabler, som importance, timestamp, pressure, temperature, light intensity, etc. Objekten är placerade i en osorterad array och varje gång 1000000 objekt (av typen Observation) har genererats skickas de till en central enhet som sorterar objekten på fältet `importance`, vilket har typ double. Vilken sorteringsalgoritm skulle du använda för att minimera körtiden för att sortera alla observationer på `importance`?
- Då det varit när ögat ett par gånger har du bestämt att refaktorera sorteringsprocessen för att hantera den ovanliga men farliga situationen då några observationer genereras med felaktiga värden på `importance`. Det visar sig att du kan upptäcka detta genom att sortera på `timestamp` och `importance` för varje observation.

  I stället för att sortera på `importance` vill du alltså först sortera observationerna på `timestamp`. Variabeln `timestamp` har en jämförbar typ som kallas DateTime. Vilken sorteringsalgoritm skulle du använda för att minimera körtiden för att sortera alla 1000000 observationer på `timestamp`?
- Efter att ha sorterat på `timestamp` vill du sortera på `importance` så att alla objekt med samma `timestamp` fortfarande är samlade i sekvensen. Vilken sorteringsalgoritm skulle du använda för att minimera körtiden och samtidigt behålla den här "klustringen"?
- Du itererar över arrayen, uppdaterar värdet på `importance` för de väldigt få dåliga observationerna med nya värden och sorterar igen. Vilken sorteringsalgoritm använder du för att sortera på `importance` samtidigt som du minimerar körtiden?

(A) Quicksort (B) Mergesort (C) Insertionsort (D) Selectionsort

6. Vi använder en array med indexen 0 till 8 för att implementera en hashtabell där kollisioner hanteras med hjälp av separat länkning (*separate chaining*). Hashfunktionen som används är $h(k) = k \bmod 9$. Visa en representation av tabellen efter att nycklarna 5, 28, 19, 15, 20, 33, 12, 17, 10 satts in i den initialt tomma tabellen. (2 p)