Linköpings Tekniska Högskola
Institutionen för Datavetenskap
Tommy Färnqvist

xx yy 20zz

TDDC91
Datastrukturer och algoritmer
Datortentamen (DAT1)
20zz-yy-xx

Examinator: Tommy Färnqvist
Max poäng: 110 poäng (betyg 5 = 108p, 4 = 106p, 3 = 100p)
Hjälpmedel: Inga hjälpmedel tillåtna, förutom OpenDSA!

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. Efter inloggning i datortentasystemet finns i startmenyn för Linux Mint: (100 p)

   - "OpenDSA" (öppnar URL för tentamensversion av OpenDSA i Chrome) och
   - "Inloggningsuppgifter OpenDSA" (öppnar sida med inloggningsuppgifter till tentamensversion av OpenDSA i Chrome).

   Starta tentamensversionen av OpenDSA, logga in med inloggningsuppgifterna enligt ovan och lös de anvisade uppgifterna. Genom att klicka på ditt inloggningsnamn kan du se i betygsboken hur många av de poänggivande uppgifterna du löst hittills. När du har full poäng i betygsboken är du färdig med den här uppgiften och kan logga ut. Du behöver inte skicka in något via tentamenssystemet. Om du inte löser samtliga poänggivande uppgifter får du noll poäng på den här tentamensuppgiften.

2. 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

3. En *läckande stack* är en generalisering av en stack som har metoder för att lägga till en sträng, ta bort den senast inlagda strängen och att ta bort en slumpvis utvald sträng, som i följande gränssnitt: (4 p)

	`LeakyStack()`	skapa en tom läckande stack
`void`	`push(String item)`	pusha strängen på den läckande stacken
`String`	`pop()`	ta bort och returnera den senast pushade strängen
`void`	`leak()`	ta bort en slumpvis utvald sträng från den läckande stacken

Beskriv, gärna med pseudokod, hur man kan implementera metoderna `push(String)`, `pop()` och `leak()` i tid proportionell mot $\log(N)$ i värsta fallet, där N är antalet strängar insatta i datastrukturen. Antag att du har tillgång till en slumptalsgenerator `Random.uniform(N)` som returnerar slumptal mellan 0 och $N - 1$ med likformig sannolikhet.

Exempel:

```
LeakyStack stack = new LeakyStack();
stack.push("A");   // A           [ add A ]
stack.push("B");   // A B         [ add B ]
stack.push("C");   // A B C       [ add C ]
stack.push("D");   // A B C D     [ add D ]
stack.push("E");   // A B C D E   [ add E ]
stack.pop();       // A B C D     [ remove and return E ]
stack.push("F");   // A B C D F   [ add F ]
stack.leak();      // A B C F     [ choose D at random; delete D ]
stack.leak();      // A C F       [ choose B at random; delete B ]
stack.pop();       // A C         [ remove and return F ]
stack.pop();       // A           [ remove and return C ]
stack.pop();       //             [ remove and return A ]
```

4. Givet en $N \times N$-matris av reella tal, ge en algoritm för att hitta det största talet som uppträder (minst) en gång på varje rad (eller avgör att inget sådant tal existerar). (3 p)

9	6	3	8	5
3	5	1	6	8
0	7	5	3	5
3	5	7	8	6
4	3	5	7	9

För full poäng krävs att exekveringtiden för din algoritm är proportionell mot $N^2 \log(N)$ i värsta fallet. Din algoritm får använda sig av $\mathcal{O}(N^2)$ extra minne.