HØGSKOLEN I SØR-TRØNDELAG

Avdeling for informatikk og e-læring

Målform:	LØSNINGSFORSLAG
Eksamensdato:	1. desember 2015
Varighet/eksamens tid:	5 timer
Emnekode:	TDAT2005
Emnenavn:	Algoritmer og datastrukturer
Klasse(r):	2ING
Studiepoeng:	10
Faglærer(e): (navn og telefonnr på eksamensdagen)	Helge Hafting, tlf 73559544 Mildrid Ljosland, tlf 73559556/93080942
Kontaktperson(adm.) (fylles ut ved behov – kun ved kursemner)	
Hjelpemidler:	Ett stemplet A4-ark med valgfritt innhold
Oppgavesettet består av: (antall oppgaver og antall sider inkl. forside)	7 oppgaver og 6 sider inkludert forside og 2 vedlegg.
Vedlegg består av: (antall sider)	2 sider

Merknad:

Oppgaveteksten kan beholdes av studenter som sitter eksamenstiden ut.

NB! Les gjennom hele oppgavesettet før du begynner arbeidet, og disponer tiden.

Dersom noe virker uklart i oppgavsettet, skal du gjøre dine egne antagelser og forklare dette i besvarelsen.

Lykke til!

Oppgave 1 (30%)

Søkemotorselskapet «BedreEnnGoogle» har K (for tiden 100, men det øker jevnt og trutt) maskiner som samler inn websider fra hver sine områder/domener, og produserer søketreff fra de websidene de har samlet inn. Hver av de K maskinene sender sine beste treff videre til en hovedmaskin som plukker ut de N høyest rangerte treffene og sender dem videre til kunden. For tiden er N=1000, men man vurderer å øke det.

Per i dag sender hver av de K maskinene N treff til hovedmaskinen, slik at hovedmaskinen plukker ut de N beste blant K*N forslag. Men det betyr jo at (N-1)*K treff egentlig er bortkastet, så det er kommet et forslag om at hver maskin bare skal produsere R treff, der R≥ N/K og R < N. (R må selvfølgelig være et heltall, så N/K avrundes til minste heltall som er større eller lik N/K.) Man er imidlertid usikker på hvor stor R må være for at man skal være rimelig sikker på å få de N beste totalt.

Eksempel (K=2 og N=10): Gitt følgende rangeringer i sortert rekkefølge:
Maskin 1: 53, 39, 25, 13, 11, 9, 5, 3, 2, 1
Maskin 2: 36, 31, 28, 24, 22, 19, 17, 14, 10, 8
Da bør treffene 53, 39, 36, 31, 28, 25, 24, 22, 19, 17 leveres. Men hvis hver maskin bare leverer R=10/2=5 treff, vil hovedmaskinen levere treffene 53, 39, 36, 31, 28, 25, 24, 22, 13, 11. Altså blir de to siste feil. Hvis R=6 vil resultatet bli 53, 39, 36, 31, 28, 25, 24, 22, 19, 13 som også er feil, mens hvis R=7 (eller større) får vi riktig resultat.

Du jobber i «BedreEnnGoogle» og har fått i oppdrag å finne ut hvor stor R bør være. Du bestemmer deg for å lage et program som simulerer dette og på bakgrunn av simuleringsresultatene finner sannsynligheten for at de N beste treffene er blitt med for ulike verdier av R.

Du finner ut at programmet ditt må gjøre følgende:

A. Gjenta 1000 ganger:
    1. Produser K tallrekker, hver med N tall, ved hjelp av en random-generator.
    2. Fra de K*N tallene, finn de N største. Dette er fasiten.
    3. Gjenta for 10 forskjellige R-verdier mellom N/K og N:
        Finn ut om fasiten oppnås hvis vi tar med R treff fra hver maskin.
        Hvis ja: Øk antallOK for denne R-verdien med 1.

A. For hver av de 10 antallOK-ene, del på 1000, så har du sannsynligheten for å presentere alle de N beste for denne R-verdien.

I alle spørsmålene om kompleksitet i oppgavene under, skal du gi svaret i O-notasjon uttrykt ved N og K. Du behøver ikke å tenke på Ω og Θ. Husk begrunnelse! Du kan anta at N≥K.

a) Hva er kompleksiteten av punkt 1, altså å produsere K tallrekker med N tall i hver? Hva blir kompleksiteten når du skal gjenta dette 1000 ganger?

Svar: Å produsere et tall med en Random-generator, er O(1). Siden vi totalt produserer KN tall, blir kompleksiteten O(KN). (Vi har en ytre løkke som går K ganger, og en indre som går N ganger.) Å gjenta dette 1000 ganger gir fortsatt O(KN). (Konstanter forsvinner i O-natasjon.)

Heretter kan du se bort fra at det skal gjentas 1000 ganger, og bare se på ett gjennomløp av løkka A.

b) Hvis du starter punkt 2 med å sortere de K tallrekkene, hva blir kompleksiteten av dette?

Svar: Å sortere N tall er O(NlogN) hvis man bruker en rask sorteringsalgoritme. (Eventuelt O(N) hvis man forutsetter at alle tallene er heltall og man har en øvre

grense på dem som ikke er større enn N, da kan tellesortering brukes. Men dette er ikke helt realistisk her.) K slike sortereringer blir dermed O(KNlogN) (eventuelt O(KN).

c) Når du har de K sorterte tallrekkene, hvordan vil du finne de N største tallene? Hvis du kommer på mer enn en måte å gjøre det på, velg den mest effektive.

Svar: Det smarteste her vil være å legge det største tallet i hver tallrekke i en maks-heap. Deretter henter du ut største element, finner ut hvilken tallrekke det kom fra, og legger inn neste tall fra denne tallrekka. Dette gjentas N ganger. Andre mulige måter å gjøre det på (eksempler):

- *Legg alle KN tallene inn i en tabell og sorter den*
  - *Ved en slags flettesortering der to og to tallrekker flettes sammen inntil vi har bare en igjen.*
  - *Ved å bruke velgesortering, men stoppe etter N runder.*
  - *Ved å bruke heapsort, men slutte etter N runder.*
- *Legg alle KN tallene inn i en tabell, men ta først alle på posisjon 0, deretter alle på posisjon 1 osv. Den samlede tabellen vil være nesten sortert, og (synkende) innsettingssortering vil bli rask.*
- *Start med å si at pos[i]=0, for alle i mellom 0 og K. Gjenta N ganger: Finn den største av de K tallene på posisjon pos[i], si at dette er tallrekke nr m. Øk pos[m] med 1.*

a) Hva blir kompleksiteten av den metoden du har beskrevet i c? Av punkt 2 totalt?

Svar: Legge K tall inn i en heap: O(K) (ved bruk av lag_heap). Hent ut største (uten å omorganisere heapen,): O(1). Finne ut hvilken tallrekke dette hører til: O(1) forutsatt at man sørger for å lagre det sammen med tallet. Sette inn nytt tall: Gjøres ved å putte det inn i rota og bruke prio_ned som er O(logK). Siden dette må gjentas N ganger, får vi totalt O(NlogK). Totalt for punkt 2 får vi O(sortering)+O(uthenting). Siden logK < K vil O(uthenting) < O(sortering), slik at den totale kompleksiteten for punkt 2 blir O(sortering)=O(KNlogN) eventuelt O(KN).

Punkt 3 kan gjøres på følgende måte hvis man i punkt 2 holder rede på hvilken tallrekke de ulike tallene stammer fra:
Tell opp antall tall i fasiten som kommer fra de ulike tallrekkene. Finn maksimum av disse antallene, kall det M. For hver av de 10 R-verdiene: Hvis M≤R, øk antallOK med 1.

b) Hva blir kompleksiteten av punkt 3? Og hva blir kompleksiteten av 1, 2 og 3 totalt? Av hele algoritmen, inkludert 1000 gjennomløp av løkka i A samt punkt B?

Svar: Kompleksiteten av punkt 3: Først må vi sjekke N tall, som er O(N). Deretter må vi finne maks av K tall, som er O(K), men siden N≥K, vil vi totalt få O(N).
Kompleksiteten av 1, 2 og 3 totalt: maks(O(pkt 1),O(pkt 2),O(pkt 3))=maks(O(KN), O(KNlogN),O(N)) = O(KNlogN) (eventuelt O(KN) hvis vi bruker tellesortering).
Siden punkt B bare er en enkel løkke som går 10 ganger, vil ikke dette bidra til kompleksiteten. Løkka som går 1000 ganger bidrar heller ikke, så kompleksiteten av hele algoritmen blir lik kompleksiteten av punkt 2, altså O(KNlogN) (eventuelt O(KN)).

Nå skal vi gå over til å se på hvordan du kan gjøre punkt 2 hvis du ikke starter med å sortere de K tallrekkene.

c) Punkt 2 kan utføres ved å legge alle de K*N tallene i en heap og hente ut de N største. Hva blir kompleksiteten av dette?

*Svar: Legge alle tallene i en heap: O(KN). Hente ut de N største: N*O(log(KN)). Siden log(KN)=log(K)+log(N) og N≥K, får vi N*O(log(KN))=O(N(logN+logK))=O(NlogN). Totalt: O(N(K+logN)).*

Algoritmen quickselect er en algoritme som har samme tankegang som quicksort, men i stedet for å sortere tallene, løser den problemet: Blant n tall,

finn de k minste (og dermed også de n-k største). (NB: n og k her betyr ikke det samme som N og K ellers i oppgaven.)
Den deler inn i "små" og "store" tall akkurat som quicksort, men kaller seg selv rekursivt bare på den ene delen som inneholder delingspunktet mellom de k minste og de n-k største. Dermed finner den etterhvert skillepunktet, og ender opp med de k minste på den ene siden og de n-k største på den andre siden (men vanligvis ikke i sortert orden).

Algoritmen er slik:

```
public static void quickselect(int[] t, int v, int h, int k) {
    if (h-v>2) {
        int delepos = splitt(t, v, h);
        if (delepos  > k) quickselect(t, v, delepos-1, k);
        else if (delepos < k) quickselect(t, delepos+1, h, k);
        else return;
    }
    else median3sort(t,v,h);
}
```

Metodene splitt og median3sort er som i læreboka:

```
public static int median3sort(int[] t, int v, int h) {
    int m = (v+h)/2;
    if (t[v]>t[m]) bytt(t,v,m);
    if (t[m] > t[h]) {
        bytt(t, m, h);
        if (t[v] > t[m]) bytt(t, v, m);
    }
    return m;
}
public static int splitt(int[] t, int v, int h) {
    int iv, ih;
    int m = median3sort(t, v, h);
    int dv = t[m];
    bytt(t,m, h-1);
    for (iv=v,ih=h-1;;) {
        while(t[++iv]<dv);
        while(t[--ih]>dv);
        if (iv>=ih) break;
        bytt(t,iv,ih);
    }
    bytt(t,iv,h-1);
    return iv;
}
```

d) Demonstrer hvordan dette vil fungere hvis du skal finne de 5 minste (og de 5 største) i tabellen 25, 13, 5, 31, 19, 8, 12, 10, 21, 4.

Svar:

Start, v=0, h=9, k=5	*25*	*13*	*5*	*31*	*19*	*8*	*12*	*10*	*21*	*4*
Median3sort, v=0, h=9, m=4	*4*				*19*					*25*
Splitt, v=0, h=9, m=4, dv=19					*21*				*19*	
				10	*12*		*21*	*31*		
iv= 6							*19*		*21*	
Etter første splitt, delepos = 6	*4*	*13*	*5*	*10*	*12*	*8*	*19*	*31*	*21*	*25*
6>5, v=0, h=5, k=5	*4*	*1*	*5*	*1*	*1*	*8*				

		3		0	2					
Median3sort, v=0, h=5, m=2										
Splitt, v=0, h=5, m=2, dv=5			12		5					
iv=1		5			13					
Etter andre splitt, delepos=1	4	5	12	10	13	8				
1<5, v=2, h=5, k=5			12	10	13	8				
Median3sort, v=2, h=5, m=3			8	10		12				
Splitt, v=2, h=5, m=3, dv=10				13	10					
iv=3				10	13					
Etter tredje splitt, delepos=3			8	10	13	12				
3<5, v=4, h=5					13	12				
5-4 ikke >2, median3sort, v=4, h=5, m=4					12	13				
Ferdig resultat:	4	5	8	10	12	13	19	31	21	25

e) Vis at quickselect har kompleksitet O(n)

Svar:
Vi har en algoritme der vi først har metodekallet splitt, som er O(n). Deretter har vi en if-setning som vi i normalt går inn i (hvis ikke, avsluttes metoden). Inni den ytre if-en har vi en ny if der vi normalt får et rekursivt kall med cirka halv størrelse (hvis ikke metoden avsluttes).
Vi kan derfor sette opp følgende uttrykk:

$$T(n) = T(n/2) + cn$$

$$a = 1 \quad b = 2 \quad k = 1$$

$$b^k = 2^1 = 2 > a$$

$$T(n) \in O(n)$$

(Vi får O(n), ikke $\Theta(n)$ siden rekursjonen noen ganger kan avsluttes før vi har nådd helt ned til basis.)

f) Vi kan utføre punkt 2 ved å lage en lang tabell av alle de K tabellene og utføre quickselect på den. Hva blir nå kompleksiteten av punkt 2?

Svar: Vi får nå n=KN. Dermed får vi O(KN).

Oppgave 2 (15%)

a) Forklar hva leksikografisk ordning er.
Svar: Leksikografisk sortering brukes når det som skal ordnes består av 0 eller flere symboler hentet fra et alfabet, dvs. er strenger over dette alfabetet. Symbolene i alfabetet må ha en partiell ordning som vi kaller R. Da gjelder:
For to vilkårlige strenger x og y sammenligner vi symbolene fra venstre mot høyre. Så lenge de er like, fortsetter vi mot høyre.
Hvis vi kommer til en posisjon i der symbolet i x er forskjellig fra symbolet i y, er x «mindre enn» y hvis x[i]Ry[i] og y er «mindre enn» x hvis y[i]Rx[i].
Hvis vi kommer til en posisjon i der det er slutt på x, men ikke på y, er x «mindre enn» y. Hvis det er slutt på y, men ikke på x, er y «mindre enn» x. Hvis det er slutt på begge, er x og y «like store».

Fra https://no.wikipedia.org/wiki/Eliteserien_i_fotball_for_menn henter vi fl lgende beskrivelse:

Lagene rangeres etter antall poeng (tre for seier, ett for uavgjort, null for tap) ved sesongslutt. Hvis to eller flere lag ender med samme poengsum, blir m lforskjellen (antall scorede m l minus antall sluppet inn) tellende, deretter antall scorede m l, og til slutt lagenes innbyrdes resultater.

b) I vedlegg 1 er gitt klassen «Fotball». Lag en klasse «Lag» slik at setningen Collections.sort(lagene); i «Fotball» sortere lagene i henhold til reglene gitt over, bortsett fra at du ikke skal ta hensyn til innbyrdes resultater (siste regel). Du skal bruke leksikografisk ordning for å få til dette.

Svar:

```
public class Lag implements Comparable<Lag> {
   private String navn;
   private int[] poeng;
   // poeng[0]: Antall poeng
   // poeng[1]: Målforskjell
   // poeng[2]: Antall scorede mål
   public Lag(String navn) {
      this.navn = navn;
      poeng = new int[3];
   }
   public int compareTo(Lag detAndre) {
      for (int i=0; i < 3; i++) {
         if (poeng[i] != detAndre.poeng[i]) {
            return detAndre.poeng[i]-poeng[i];
         }
      }
      return 0;
   }
   public void register (int antScore, int antSluppet) {
      if (antScore > antSluppet) poeng[0] += 3;
      else if (antScore == antSluppet) poeng[0] += 1;
      poeng[1] += (antScore - antSluppet);
      poeng[2] += antScore;
   }
   public String toString() { // Ikke spurt om i oppgaven
      return navn+":\t" + poeng[0]+"\t"+poeng[2]+"-"+(poeng[2]-poeng[1]);
   }
}
```

Oppgave 3 (15%)

Se grafen i vedlegg 2.

a) Finn maksimum flyt gjennom grafen, ved hjelp av flytøkende veier. Oppgi hvilke flytøkende veier du brukte, og hvor mye flyt du fikk lagt til med hver av dem.
Her fins mange korrekte løsninger. Dette er én av dem:
KABS:20
KEFS:20
KCDS:20
KABDFS:5
KABDCEFS:20
Maksimal flyt blir 85, uansett hvilke veier man bruker for å finne ut dette.

b) Sorter grafen topologisk, eller forklar hvorfor dette ikke er mulig.
F.eks: KCAEBDFS eller KCABDEFS. Det er noen flere rette løsninger også.

c) Se bort fra retningen på kantene i denne deloppgaven. Finn et minimalt spenntre for grafen.

Kan f.eks. bruke Kruskals metode, og får disse kantene med i det minimale spenntreet:
DF:5, CA:5, CD:20, BS:20, DS:20, KE:20 og CE:20. Denne grafen har mange andre minimale spenntrær; men summen av vektene må være 110.

d) Tegn inn en ny kant fra S til C. Finn de sterkt sammenhengende komponentene i denne nye grafen.
Med kanten SC får vi to sterkt sammenhengende komponenter. Noden K alene blir en komponent, alle de andre nodene blir den andre komponenten.

Oppgave 4 (10%)

a) Fortell kort om hvordan Lempel-Ziv komprimering fungerer.
Kompresjon oppnås ved å erstatte gjentatte strenger med referanse til den forrige strengen. Når man komprimerer, leser man gjennom teksten. Strenger man ikke har sett før, skriver man opp som vanlig. Strenger man har sett, erstattes med en referanse som forteller hvor langt bakover man må gå for å finne kopien, og hvor lang kopien skal være. Dermed er det mulig å pakke ut igjen senere. Hvis referansene er kortere enn strengene de erstatter, oppnår vi kompresjon.

b) Demonstrer Lempel-Ziv kompresjon på «fort, fortere, fortest»
Her bruker jeg klammer [] for å vise referanser. Første tall er angir «tegn bakover», andre tall angir hvor mange som skal kopieres:
«fort, [6,4]ere[9,7]st» I alt 11 tegn erstattes av to referanser.

Oppgave 5 (10%)

a) Er det mulig å utvikle en algoritme som sorterer heltall asymptotisk raskere enn O(n)? Begrunn svaret.
Nei. Hvis algoritmen skal være asymptotisk raskere enn O(n), kan den ikke se på alle de n heltallene. Men et tall som «hoppes over» kan jo stå feil i forhold til de andre, og blir dermed ikke sortert på plass. En slik algoritme gir altså ikke korrekt sortering i alle tilfeller.

b) I øvingsopplegget så vi på en blandet sorteringsalgoritme, quicksort med innsettingssortering som «hjelpealgoritme». Hva oppnår vi med en slik hjelpealgoritme?
Vi oppnådde raskere sortering. Quicksort er en algoritme som er god på store datasett. Innsettingssortering er raskere på svært små datasett, og kan dermed brukes når quicksort produserer små datasett som mellomresultater.

Oppgave 6 (10%)

Gitt D = {2,3,4,5,6,7,8,9,10,11,12}.

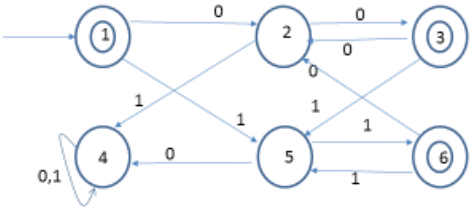
a) La relasjonen R være definert på D ved at (a,b)ϵR hvis og bare hvis (a mod 5)=(b mod 5). Vis at R er en ekvivalensrelasjon.
Svar: En ekvivalensrelasjon er refleksiv, symmetrisk og transitiv.
Refleksiv: Ja siden (a mod 5) = (a mod 5) uansett a.
Symmetrisk: Ja siden (a mod 5) = (b mod 5) medfører at (b mod 5) = (a mod 5).
Transitiv: Ja. Transitiv sier at hvis aRb og bRc, så aRc. Anta at (a mod 5)=x. Siden aRb, må da også (b mod 5)=x. Og siden bRc må (c mod 5) = x. Dermed får vi at (a mod 5)=(c mod 5), altså aRc.
Dermed er R en ekvivalensrelasjon.

b) La relasjonen S være definert på D ved at (a,b) ϵS hvis og bare hvis a mod b=0. Er S en ekvivalensrelasjon? En partiell ordning? Husk begrunnelse.
Svar:
Refleksiv: Ja siden a mod a = 0 uansett A.
*Transitiv: Anta at a mod b = 0 og b mod c=0. Er a mod c=0? Siden a mod b = 0, har vi at a = k*b for et eller annet heltall k. Siden b mod c = 0, er b*

*= m*c for et eller annet heltall m. Da har vi at a=k*b=k*m*c=n*c der n er heltallet k*m. Dermed er a mod c = 0. Altså er S transitiv.*
Symmetrisk: Anta at a mod b = 0. Det medfører ikke at b mod a = 0, så S er ikke symmetrisk. Eksempel: 10 mod 2=0 men 2 mod 10 = 2.
*Antisymmetrisk: S er antisymmetrisk hvis det er slik at aSb og bSa bare inntreffer når a=b. a mod b = 0 gir at a=k*b for et eller annet heltall k. b mod a = 0 gir at b = m*a for et eller annet heltall m. Hvis både aSb og bSa skal gjelde, må vi ha at a = k*b = k*m*a. Siden 0 ikke er med i D, kan ikke a være 0. Dermed kan vi dele på a, og får k*m=1, altså m = 1/k. Siden både k og m skal være heltall, er den eneste måten å få til dette på, at k=m=1, altså a=b. Dermed har vi bevist at S er antisymmetrisk.*
Oppsummering: S er ikke en ekvivalensrelasjon siden den ikke er symmetrisk. Men den er en partiell ordning siden den er refleksiv, antisymmetrisk og transitiv.

Oppgave 7 (10%)

Gitt automaten A vist nedenfor.



a) Skriv opp alfabet, starttilstand, aksepterende tilstander og neste-tilstand-funksjonen for A.
Svar:
Alfabet: 0, 1
Starttilstand: 1
Aksepterende tilstander: {1,3,6}
Neste-tilstand-funksjon:

Tilstand\Input	*0*	*1*
1	*2*	*5*
2	*3*	*4*
3	*2*	*5*
4	*4*	*4*
5	*4*	*6*
6	*2*	*5*

b) La L være språket definert av A og la L_i være mengden av de strengene i L som har lengde i. Skriv opp L_0, L_1, L_2, L_3 og L_4. Lag et regulært uttrykk for L.
Svar:
L_0={ε}
L_1=Ø
L_2={00, 11}
L_3=Ø
L_4={0000, 0011, 1100, 1111}
Regulært uttrykk for L: $(00|11)^*$

Vedlegg 1

```
public class Fotball {
   ArrayList<Lag> lagene = new ArrayList<Lag>();

   public static void main(String[] args) {
      Fotball f = new Fotball();
      f.test();
   }

   public void test() {
      lagene.add(new Lag("LagA"));
      lagene.add(new Lag("LagB"));
      lagene.add(new Lag("LagC"));
      lagene.add(new Lag("LagD"));
      lagene.add(new Lag("LagE"));
      lagene.add(new Lag("LagF"));
      for (int i = 0; i < lagene.size(); i++) {
       for (int j = 0; j < lagene.size(); j++) {
          if (i != j) {
             int a = (int)(Math.random()*5);
             int b = (int)(Math.random()*5);
             registerKamp(lagene.get(i), lagene.get(j), a, b);
          }
       }
      }
      Collections.sort(lagene);
      skrivResultat();
   }
   public void registerKamp(Lag a, Lag b, int scoreA, int scoreB) {
      a.register(scoreA, scoreB);
      b.register(scoreB, scoreA);
   }
   public void skrivResultat() {
     System.out.println("Navn Poeng Scoret-Imot");
     for (int i = 0; i < lagene.size(); i++) {
         System.out.println(lagene.get(i));
      }
   }
}
```

Eksempler på utskrift:

```
Navn Poeng Scoret-Imot
LagB:     20          26-19
LagC:     14          21-21
LagD:     14          20-23
LagE:     12          19-17
LagF:     11          23-26
LagA:     11          15-18

Navn Poeng Scoret-Imot
LagE:     16          19-15
LagA:     16          15-12
LagF:     13          21-21
LagC:     13          22-23
LagD:     12          23-25
LagB:     12          20-24
```

Vedlegg 2

Graf for oppgave 3