Skriftlig Eksamen
Datastrukturer og Algoritmer (DM02)

Institut for Matematik og Datalogi
Odense Universitet

Torsdag den 6. juni 1996, kl. 9–13

Alle sædvanlige hjælpemidler (lærebøger, notater, etc.) samt brug af lommeregner er tilladt.

Eksamenssættet består af 4 opgaver på 6 nummererede sider (1–6). Fuld besvarelse er besvarelse af alle 4 opgaver.

De enkelte opgavers vægt ved bedømmelsen er angivet i procent. Der må gerne refereres til algoritmer og resultater fra lærebogen inklusive øvelsesopgaverne. Specielt må man gerne begrunde en påstand med at henvise til, at det umiddelbart følger fra et resultat i lærebogen (hvis dette altså er sandt!). Henvisninger til andre bøger (udover lærebogen) accepteres ikke som besvarelse af et spørgsmål.

Bemærk, at hvis der er et spørgsmål i en opgave, man ikke kan besvare, må man gerne besvare de efterfølgende spørgsmål og blot antage, at man har en løsning til de foregående spørgsmål.

Opgave 1 (20%)

Medianen af en liste af tal er pr. definition det tal, der ville stå i midten, hvis listen blev sorteret (det venstre af de to midterste, hvis der er et lige antal). Mere formelt er medianen af den *sorterede* følge $x_1, x_2, \ldots, x_n$ elementet med indeks $\lfloor \frac{n+1}{2} \rfloor$; altså $\frac{n+1}{2}$ rundet ned. For eksempel er medianen af tallene $4, 3, 1, 2, 4, 2$ tallet 2, da det er det tredie ($\lfloor \frac{6+1}{2} \rfloor$) tal i $1, 2, 2, 3, 4, 4$.

I denne opgave antager vi, at vi givet et indeks i kan aflæse værdien x_i i konstant tid. Specielt kan vi få fat i $x_{\lfloor \frac{n+1}{2} \rfloor}$ i konstant tid.

Nu har vi givet to *sorterede* lister A og B (begge af længde n), og vi vil gerne finde medianen af deres forening. For eksempel er medianen af foreningen af $1, 2, 2, 3, 4, 4$ og $3, 4, 5, 6, 6, 7$ tallet 4 (midterst i $1, 2, 2, 3, 3, 4, 4, 4, 5, 6, 6, 7$).

Spørgsmål a: Forklar, hvordan man kan lave en algoritme, der løser problemet i lineær tid; dvs. $\Theta(n)$. □

Spørgsmål b: Lav en hurtigere rekursiv algoritme, der løser problemet.

Hjælp: Sammenlign de to listers medianer og lav et tilsvarende problem af halv størrelse. □

Spørgsmål c: Opstil en rekursionsligning (recurrence equation), der udtrykker tiden for at løse et sådan problem som funktion af tiden, det tager at løse problemet af halv størrelse.

Udled kompleksiteten af algoritmen (O-notation). □

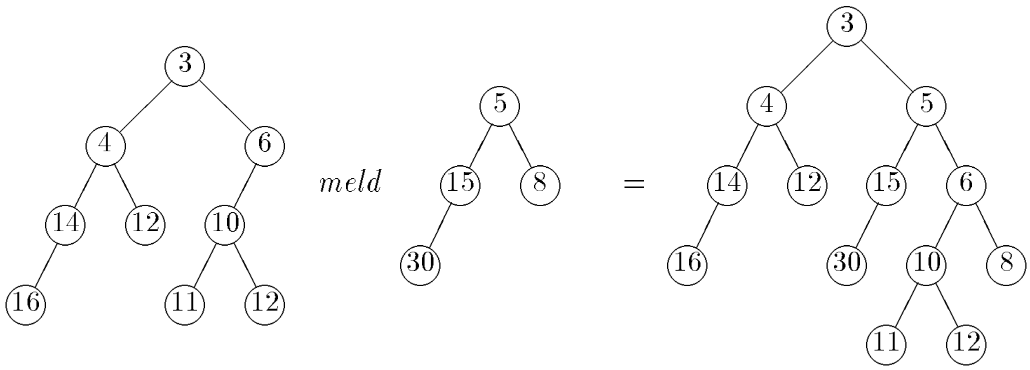
Opgave 2 (35%)

Fra Kingston kendes binomialkøer (siderne 153–156), der har en operation *meld*, som kan sætte to sådanne strukturer sammen til én. I denne opgave skal vi se på en anden implementation af en prioritetskø, der også skal have en *meld* operation.

To heap-ordnede træer kan *meld*'es til ét heap-ordnet træ ved at flette deres højrestier. Højrestien er stien fra roden til det blad, der er længst til højre (fra nu af kaldet højrebladet).

Knuderne i disse træer har indtil videre tre felter: *left*, *right* og *pri* til henholdsvis venstre- og højrebarn samt prioritet.

Nedenfor ses et eksempel på to heap-ordnede træer, der *meld*'es sammen.



Operationen foretages af følgende algoritme. Det antages, at argumenterne til funktionen er rødderne i de to træer. Operationen må gerne ødelægge de oprindelige træer (og gør det da også)!

```
function meld(s,t)
  if s = nil then return t endif
  if t = nil then return s endif

  if s.pri < t.pri then
    s.right = meld(s.right, t)
    r = s
  else
    t.right = meld(s, t.right)
    r = t
  endif

  return r
end
```

Vi udstyrer nu knuderne med et ekstra felt, *rank*. En knude x's rank defineres til at være længden af den korteste sti fra x til en knude, der mangler et venstre- eller højrebarn (eller begge).

I resultattræet ovenfor (træet helt til højre) har bl.a. knuderne med prioritet 8 og 15 rank 0, mens roden har rank 2 (da vi kan komme til knuden med prioritet 12 i to skridt).

Spørgsmål a: Hvilke knuder kan risikere at få ændret deres rank, når der foretages en *meld* vha. funktionen meld? □

Spørgsmål b: Antag, at de heap-ordnede træer s og t har knudernes rank registreret i det nye felt. Angiv, hvordan algoritmen ovenfor skal modificeres for at r = meld(s,t) har de korrekte værdier i rank-felterne efter operationen. □

Vi kalder nu et heap-ordnet træ med rank-felter *venstretungt*, hvis der gælder for alle knuder x, at hvis det har et højrebarn, så har det også et venstrebarn. Desuden skal der gælde, at hvis x har både et venstrebarn y og et højrebarn z, så er $y.rank \geq z.rank$. I illustrationen ovenfor er de to argumenttræer venstretunge, men resultattræet er ikke (knuden med prioritet 5 er et problem).

Spørgsmål c: Antag, at s og t er venstretunge træer. Angiv, hvordan algoritmen ovenfor skal modificeres, for at r = meld(s,t) bliver et venstretungt træ efter operationen. □

Spørgsmål d: Vis, at højrestien i et venstretungt træ med n elementer har længde $O(\log n)$.

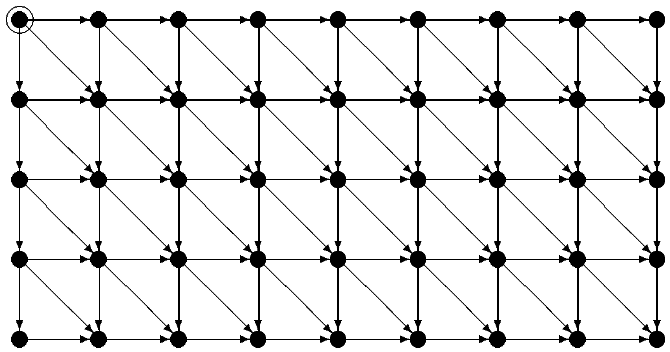
Hjælp: Vis ved induktion i knudernes rank, at der for alle knuder x gælder, at $2^{x.rank} \leq size(x)$, hvor $size(x)$ er antallet af knuder i undertræet, der har x som rod. Vis dernæst, at $x.rank$ er lig med afstanden til højrebladet for alle knuder x på højrestien. □

Ovenstående viser, at *meld* er $O(\log n)$, hvor n_1 og n_2 er størrelsen af de to argumenter, og $n = \max(n_1, n_2)$.

Spørgsmål e: Forklar, hvordan operationerne *insert* og *deletemin* kan implementeres med tidskompleksitet $O(\log n)$. □

Opgave 3 (25%)

Opgaven drejer sig om en særlig type orienterede vægtede grafer med ikke-negative vægte. Knuderne sidder i et rektangulært gitter, og hver knude har en kant til dens sydlige, østlige og sydøstlige naboer (hvis disse knuder findes). Den nordvestligste knude kaldes *startknuden*. Grafer, der ser ud som lige beskrevet, kaldes i det følgende *sydøstgrafer*. Et eksempel med 5 rækker og 9 søjler ses nedenfor. Startknuden er markeret med en ring. Kanternes vægte er *ikke* angivet på tegningen.



Spørgsmål a: Hvis en sydøstgraf har r rækker og s søjler, så har den naturligvis $n = r \cdot s$ knuder. Hvor mange kanter har den? Svaret skal udtrykkes ved r og s. □

Vi er nu interesserede i at beregne længden af den korteste vej fra startknuden til alle andre knuder i grafen.

Spørgsmål b: Hvilken kompleksitet garanterer Dijkstra's algoritme os, hvis vi bruger den til at løse ovenstående problem på sydøstgrafer? Svaret skal være på formen $O(f)$, hvor f er en funktion af n. □

Spørgsmål c: Beskriv selv en algoritme, der for sydøstgrafer løser problemet i tid $O(n)$. □

Selv om den generelle analyse af tidskompleksiteten af Dijkstra's algoritme ikke *lover* os, at algoritmen afvikles i tid $O(n)$ på sydøstgrafer, kunne det jo godt være, at det alligevel skete. Det er dog ikke tilfældet:

Spørgsmål d: Argumentér for, at det kan tage *mere* end lineær tid at afvikle Dijkstra's algoritme på sydøstgrafer.

Hjælp: se på sydøstgrafer med $r = 2$ og meget små vægte på de øverste $s - 1$ kanter og meget store vægte på resten. □

Opgave 4 (20%)

Opgaven drejer sig om hashing under anvendelse af teknikken *chaining*. Som det forklares i Kingston side 124, har man en tabel, hvis indgange er hægtede lister af nøgler. Derved kan man lave *insert* i konstant tid (man sætter bare ind først i listen), mens *delete* kan tage tid proportionalt med listens længde.

Risikoen for at få lange lister (dvs. risikoen for kollisioner) afhænger naturligvis af tabellens størrelse. For at holde *delete* rimeligt effektiv bestemmer vi os nu til, at når antallet af nøgler i strukturen n udgør halvdelen af tabelstørrelsen $size$, så laver vi en ny, dobbelt så stor tabel (dvs. af størrelse $2 \cdot size$) og flytter alle nøglerne over i den nye tabel. Dette kaldes en *genbygning*. Dette sker selvfølgelig i forbindelse med en *insert*, så *insert* er nu ikke længere $O(1)$, men betydeligt dyrere. Tabelstørrelsen $size$ starter med at være 2 og er dermed altid en 2-potens.

Spørgsmål a: Hvor lang tid tager en *insert*, der udløser en genbygning, henholdsvis én, der ikke gør? □

Spørgsmål b: Betragt funktionen $4(n - \frac{size}{4})$. Hvordan ændres denne ved en *insert*, der udløser en genbygning, henholdsvis én, der ikke gør? □

Spørgsmål c: Vis, at *insert* er amortiseret $O(1)$. □