Studentnummer _____

Løsningsforslag, eksamen i
IT1105/TDT4120 Algoritmer og datastrukturer
13. desember 2004 0900-1300

Tillatte hjelpemidler: Bestemt enkel kalkulator. Ingen trykte eller håndskrevne hjelpemidler.
Merk: Svarene skal skrives på angitt sted på oppgavearket.

Oppgave 1 (20%)

For hver algoritme nedenfor skal du *kort* beskrive (*i*) *hvilket problem* algoritmen løser, (*ii*) *hvilke krav* som eventuelt stilles til probleminstansene og (*iii*) grovt *hvordan algoritmen virker*. Hvert delspørsmål besvares med *maksimalt 50 ord*, gjerne i stikkordsform.

Merknad til sensur: Ved svar på mer enn 50 ord skal det legges vekt på de 50 første ordene; den resterende teksten skal i liten eller ingen grad telle for vurderingen. Figurer kan tas med i vurderingen. Det skal ikke trekkes poeng for svar på over 50 ord.

4% *a*) QuickSort

Svar: Sortering. Splitt-og-hersk-algoritme der verdiene partisjoneres i store og små verdier rundt en splittverdi og hver halvdel sorteres rekursivt.

4% *b*) Dijkstras algoritme

Svar: Korteste vei i en graf der kantvektene ikke er negative. Oppdaterer korteste vei funnet så langt hver gang en node besøkes; besøker alltid den nærmeste av de ubesøkte nodene (vanligvis ved hjelp av en heap).

4% *c*) Binærsøk

Svar: Søk i en sortert tabell. Sammenligner søkenøkkel med midterste verdi og søker rekursivt i riktig halvdel.

4% *d*) Prims algoritme

Svar: Bygger et minimalt spenntre ved hele tiden å legge til den letteste gjenværende kanten ut fra treet (til en node som ikke ligger i treet). Bruker gjerne en heap for å holde styr på de gjenværende nodene.

4% *e*) Warshalls algoritme

Svar: Finner den transitive tillukningen til en graf ved hjelp av dynamisk programmering. Et delproblem består i å finne ut hvorvidt man kan komme seg fra node u til node v ved bare å gå via de k første nodene.

Oppgave 2 (20%)

I de følgende deloppgavene, kryss av for "ja" eller "nei" og gi en kort begrunnelse for svaret.

Merk: Svar med gal eller manglende begrunnelse gir ingen poeng.

4% *a*) En dårlig algoritme A som sjekker om et heltall n er et primtall har kjøretid $\Theta(n)$. Har denne algoritmen eksponentiell kjøretid?
Hint: Hvordan defineres problemstørrelse?

Svar: Ja.

Begrunnelse: Problemstørrelsen er $m \in \Theta(\log n)$, så kjøretiden blir $\Theta(2^m)$.

4% *b*) Har balanserte, rettede trær færre mulige topologiske ordninger enn andre sammenhengende, rettede asykliske grafer med like mange noder?

Svar: Nei.

Begrunnelse: Det vil alltid finnes en annen DAG med like mange eller færre ordninger. (Man kan for eksempel legge til en kant for å finne en slik graf.)

Merknad til sensur: Et unntak her er grafer med én eller to noder. Her finnes det ingen sykliske rettede grafer om man ikke tillater selv-løkker. Svaret "ja" kan også forsvares om man forstår det slik at det spørres om hvorvidt det *finnes* et balansert, rettet tre (for eksempel en sti) som har færre ordninger enn en en annen DAG med like mange noder. Dette må da fremgå av begrunnelsen.

4% *c*) Steinar Hopp vil komme seg over en bred elv. Det er naturligvis begrenset hvor langt han kan hoppe, så han planlegger å hoppe fra stein til stein. Han vil finne den sekvensen av steiner som gir færrest hopp, og mener at dybde-først-søk egner seg bedre enn bredde-først-søk til dette. Er du enig? Angi i begrunnelsen hvordan algoritmene kan brukes, og hvorfor du foretrekker den ene fremfor den andre.

Svar: Nei.

Begrunnelse: Han ønsker i praksis å finne korteste vei i en graf (der alle kant-vekter er 1). Hvis han skal bruke DFS må han prøve alle stier (eksponentiell kjøretid), mens med BFS får han svaret direkte i lineær tid.

4% *d*) I asymptotisk notasjon beskriver O og Ω ulike ting. Er det riktig å si at O er et mål på *worst-case*-kjøretiden til en algoritme og at Ω er et mål på *best-case*-kjøretiden?

Svar: Nei.

Begrunnelse: O(*f*) er mengden av funksjoner som har *f* som øvre grense – dette har ingenting med *worst-case* å gjøre i seg selv. (Tilsvarende for Ω, nedre grense og *best-case*).

4% *e*) Vi kan si at et problem *A* er vanskeligere enn et problem *B* hvis alle instanser av problem *B* også kan ses som instanser av problem *A*, men ikke omvendt. Er topologisk sortering et vanskeligere problem enn vanlig sortering?

Merknad til sensur: I den engelske oppgave teksten var den siste forekomsten av "*A*" i teksten over ved en feiltakelse byttet ut med "*B*". Det ble opplyst om dette på eksamen.

Svar: Ja.

Begrunnelse: Vanlig sortering kan ses som topologisk sortering der alle elementer er avhengig av de mindre elementene.

Oppgave 3 (20%)

4% *a*) Her følger en liste med betegnelser på kjøretidsklasser – ordne dem i stigende rekkefølge: *konstant*, *eksponentiell*, *lineær*, *kubisk*, *kvadratisk*, *faktoriell*, *logaritmisk*, *n-log-n*. Ingen begrunnelse er nødvendig.

Merknad til sensur: Med *faktoriell* er ment som en oversettelse (i adjektivsform) av det engelske begrepet *factorial* (det vil si, kjøretiden er fakultetet av *n*, eller Θ(*n*!)). Dette ble opplyst på eksamen.

Svar: *konstant*, *logaritmisk*, *lineær*, *n-log-n*, *kvadratisk*, *kubisk*, *eksponentiell*, *faktoriell*.

4% *b*) Hva gjør følgende programsnutt og hva blir kjøretiden? Bruk Θ-notasjon og begrunn svaret. Skriv kort.

Algorithm *Azathoth*(*A*[1…*n*])

```
1. i ← 1
2. j ← n
3. while i < j
4.     if A[i] > A[j]
5.         i ← i + 1
6.     else
7.         j ← j – 1
8. return A[i]
```

Svar: Finner det minste elementet i A i $\Theta(n)$ tid.

Begrunnelse: Den beveger seg innover fra sidene, og dropper hele tiden den som er størst. Hver posisjon besøkes én gang.

6% *c*) Hva gjør følgende programsnutt og hva blir kjøretiden? Funksjonen *floor*() runder ned til nærmeste heltall. Bruk Θ-notasjon og begrunn svaret. Skriv kort.

Algorithm *Astaroth*(*i*, *j*, *A*[1…*n*], *B*[1…*n*])

```
1. if i = j
2.     if A[j] > B[j]
3.         return A[j] – B[j]
4.     else
5.         return B[j] – A[j]
6. k ← floor((i + j) / 2)
7. return Astaroth(i, k, A, B) + Astaroth(k + 1, j, A, B)
```

Svar: Finner summen av absoluttverdi-forskjeller mellom A og B med kjøretid $\Theta(n)$.
(Evt.: Finner Manhattan-avstanden/taxi-avstanden mellom A og B.)

Begrunnelse: Vi antar at $j \geq i$. Disse angir et delproblem. Hvis $i = j$ returneres $|A[i] - B[i]|$. Ellers beregnes verdien rekursivt for hver halvdel, og vi returnerer summen. Dette gir rekurrensen $T(n) = 2T(n/2) + \Theta(1)$ som løses til $\Theta(n)$.

6% *d*) En algoritme har et fast sett med instruksjoner som alltid utføres, et sett med instruksjoner der antallet steg er en fast prosent av problemstørrelsen og k rekursive kall på en k-del av problemet. Hva blir kjøretiden til

algoritmen? Bruk Θ-notasjon og begrunn svaret med en kort skisse av utregningen din.

Svar: $\Theta(n \log n)$.

Begrunnelse: Vi kan sette opp rekurrensen $T(n) = kT(n/k) + \Theta(n)$. Dette kan løses med Master-teoremet ved å sette $a = b = k$ og $d = 1$. Eventuelt kan man løse det iterativt (*backward substitution*), men det er litt mer tungvindt:

$$T(n) = kT(n/k) + \Theta(n)$$

$$= k(kT(n/k^2) + \Theta(n/k)) + \Theta(n) = k^2T(n/k^2) + 2\cdot\Theta(n)$$

$$= k^iT(n/k^i) + i\cdot\Theta(n)$$

[Sett $k^i = n$, dvs. $i = \log_k n$. Anta $T(1) \in \Theta(1)$.]

$$= \Theta(n) + (\log_k n)\cdot\Theta(n) = \Theta(n \log n)$$

Oppgave 4 (10%)

Hvis vi fargelegger nodene i en graf og to nabonoder har samme farge kalles dette en konflikt. En graf er k-fargbar hvis den kan fargelegges med k farger uten at det oppstår konflikter.

4% *a*) Finnes det en kjent polynomisk algoritme for å avgjøre om en graf er k-fargbar, for en vilkårlig k? Hvis ja, hvordan virker denne, og hva er kjøretiden (i Θ-notasjon)? Hvis nei, hvorfor?

Svar: Nei – fordi dette er et NP-komplett problem. Det finnes ingen kjente polynomiske løsninger på noen NP-komplette problemer.

2% *b*) Hvordan vil du løse problemet hvis $k = 2$? Angi øvre og nedre asymptotiske grenser for kjøretiden.

Svar: En graf kan bare tofarges på en måte (hvis vi ser bort fra fargevalg). For å sjekke om det er mulig trenger vi bare traversere grafen. Kjøretiden for denne algoritmen er $\Omega(1)$ og $O(n)$.

4% *c*) Anta at du har en graf som *ikke* er tofargbar. Du ønsker å fargelegge den med to farger slik at antall konflikter blir minst mulig. Gi en kort beskrivelse av en enkel *branch and bound*-løsning på dette problemet.

Svar: Vi prøver alle muligheter rekursivt. Heuristikken vår er antall konflikter i det vi har fargelagt så langt. Hvis denne overstiger beste løsning så langt kan vi tilbakespore (*backtrack*).

Oppgave 5 (15%)

For hver av algoritmene under, hvilke av de følgende asymptotiske kjøretidsklassene kan brukes til å beskrive algoritmens kjøretid? Du kan ikke anta noe om problem-instansene.

$\Omega(n)$, $\Omega(n \log n)$, $\Omega(n^2)$, $O(n)$, $O(n \log n)$, $O(n^2)$, $\Theta(n)$, $\Theta(n \log n)$, $\Theta(n^2)$

Oppgi de relevante kjøretidsklassene (for eksempel "$\Omega(n^2)$, $O(n)$") eller skriv "Ingen" hvis ingen av klassene kan brukes. Hvis flere klasser kan brukes skal alle oppgis i svaret. Gi en kort begrunnelse til hver deloppgave.

5% *a*) Innsetting av n verdier i et tomt binært søketre.

Svar: $\Omega(n)$, $\Omega(n \log n)$, $O(n^2)$.

Begrunnelse: Hvis treet er balansert i hvert trinn (*best-case*) får vi $\Theta(n \log n)$; for et degenerert tre (*worst-case*) får vi $\Theta(n^2)$. Det er bare grensene over (av de tilgjengelige alternativene) som dekker begge disse tilfellene (og alt imellom).

5% *b*) Sortering ved innsetting.

Svar: $\Omega(n)$, $O(n^2)$.

Begrunnelse: Hvis tabellen er sortert (*best-case*) får vi $\Theta(n \log n)$; hvis den er ordnet synkende (*worst-case*) får vi $\Theta(n^2)$. Det er bare grensene over (av de tilgjengelige alternativene) som dekker begge disse tilfellene (og alt imellom).

5% *c*) MergeSort.

Svar: $\Omega(n)$, $\Omega(n \log n)$, $O(n \log n)$, $O(n^2)$, $\Theta(n \log n)$.

Begrunnelse: MergeSort har en deterministisk kjøretid på $\Theta(n \log n)$. Det er bare grensene over (av de tilgjengelige alternativene) som beskriver denne klassen.

Oppgave 6 (15%)

Student Lurvik skal summere n positive flyttall, $x[1]…x[n]$, slik at avrundingsfeilen blir minst mulig. Avhengig av hvordan dette gjøres, kan de ulike tallene delta i et ulikt antall summasjoner. For eksempel vil $x[1]$, $x[2]$, $x[3]$ og $x[5]$ delta i flere summasjoner (tre) enn $x[6]$ (én) i følgende summeringsrekkefølge:

$$(((x[1] + x[3]) + (x[2] + x[5])) + x[6])$$

Merk her at både tall-rekkefølgen og parentes-settingen velges fritt.

Hvor stor avrundingsfeilen for en sum av to flyttall blir er avhengig av hvor stor summen er (større sum gir større feil). Lurvik innser at det kan være lurt å summere slik at de små flyttallene deltar i flest mulig summasjoner (det vil si, de kommer "dypt" i parentes-settingen) mens de store flyttallene deltar i færrest mulig. Hvis $p(i)$ er "parentes-dybden" til flyttall $x[i]$ (antall parenteser utenfor elementet) så definerer Lurvik *feilen* til en mulig løsning som

$$p(1)\cdot x[1] + p(2)\cdot x[2] + \ldots + p(n)\cdot x[\mathrm{n}].$$

Lurvik ønsker nå å finne en parentes-setting som minimaliserer dette feil-uttrykket. Skisser en løsning på problemet. Hvilken designmetode bruker du? Vis, med støtte i pensum, at løsningen er korrekt.

Svar: Dette kan lett løses ved hjelp av en grådighetsalgoritme. Hold styr på alle summer så langt (i en heap) og summer hele tiden de to minste. Dette er det samme som gjøres i beregning av Huffman-koder, og algoritmen blir derfor korrekt.

(Man kan se $p(i)$ som lengden på koden til symbol i, mens $x[i]$ er frekvensen. Huffman-koder minimaliserer forventet tegn-lengde, som er summen i oppgaven over.)