Høgskolen i Østfold

EKSAMEN

Emnekode: ITF 20006	Emne: Algoritmer og Datastrukturer
Dato: 22.05.2015	Eksamenstid: kl 09.00 til kl 13.00
Hjelpemidler: 4 dobbeltsidige ark med notater	Faglærer: Lars Magnusson
Eksamensoppgaven: Oppgavesettet består av 9 sider inklusiv denne forsiden. Kontroller at oppgaven er komplett før du begynner å besvare spørsmålene. Dersom det er noen spørsmål ang. oppgavesettet, kan faglærer nås på mob: 91117116	
Sensurdato: 15.06.2015 Karakterene er tilgjengelige for studenter på studentweb senest 2 virkedager etter oppgitt sensurfrist. Følg instruksjoner gitt på: www.hiof.no/studentweb	

Oppgavesettet har fire deler. Hver del er merket med en prosent som angir hvor stor andel hver del utgjør av hele oppgavesettet.

Oppgavesettet inneholder en del oppgaver som krever tekstlige besvarelser. Prøv å svar så kort og konsist som mulig på disse da dere ikke blir vurdert på annet enn innhold.

1 Analyse (40%)

I denne delen skal du ta i bruk de teknikkene vi har gjennomgått i kurset for å analysere algoritmer.

Oppgave 1 - Enkel Analyse (10%)

I denne oppgaven skal du utføre en såkalt detaljert analyse av de enkle algoritmene gitt i pseudokode under. Du skal så bruke resultatet fra den detaljerte analysen til å komme frem til en asymptotisk grense (asymptotisk notasjon) for algoritmene. Du kan anta at hver programlinje tar en tidsenhet å utføre hver gang den kjøres.

```
OPPGAVE1A(n)
1  sum = 0
2  for i = 1 to n
3       sum = sum + i
4  return sum
```

```
OPPGAVE1B(n)
1  sum = 0
2  for i = 1 to n
3       sum = sum + i
4  for i = 1 to n
5       for j = 1 to 2n
6            sum = sum + i
7  return sum
```

```
OPPGAVE1C(n)
1  sum = 0
2  i = 1
3  while i ≤ n
4       sum = sum + i
5       i = i * 3
6  return sum
```

OPPGAVE1D(n)

```
1  sum = 0
2  for i = 1 to n^2
3        sum = sum + i
4  return sum
```

Oppgave 2 - Masterteoremet (10%)

I denne oppgaven skal du i første omgang bedømme hvorvidt det er mulig å benytte masterteoremet for å finne en asymptotisk grense for et sett med recurrence-likninger. I de oppgavene det er mulig skal du vise alle stegene involvert i å bruke masterteoremet for å finne den asymptotiske grensen, og i de oppgavene det ikke kan benyttes skal du forklare hvorfor.

A

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 8T(n/2) + \Theta(n^2) & \text{if } n > 1 \end{cases}$$

B

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 4T(n/4) + \Theta(n) & \text{if } n > 1 \end{cases}$$

C

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 9T(n/3) + \Theta(n \log n) & \text{if } n > 1 \end{cases}$$

D

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 2T(n/2) + \Theta(n^2) & \text{if } n > 1 \end{cases}$$

Oppgave 3 - Avansert Analyse (20%)

I denne oppgaven skal du analysere algoritmen gitt under. A er en 1-indeksert tabell med heltall, og j er et heltall. Det første kallet til algoritmen vil være OPPGAVE3$(A, 1)$. $A.length$ angir størrelsen til tabellen.

```
OPPGAVE3(A, j)
1  if j > A.length
2      return
3  i = j − 1
4  key = A[j]
5  while i > 0 and A[i] > key
6      A[i + 1] = A[i]
7      i = i − 1
8  A[i + 1] = key
9  OPPGAVE3(A, j + 1)
```

A

Du skal beskrive hva algoritmen gjør med egne ord, og du skal bevise at algoritmen er korrekt.

B

Du skal resonnere deg frem til en recurrence-likning som beskriver kjøretiden til algoritmen. Du kan begrense deg til finne en øvre grense for average-case kjøretid.

C

Sett opp et rekursjonstre eller utfør et matematisk resonnement som leder frem til en gjetning for en øvre grense for kjøretiden til algoritmen over. Denne gjetningen skal så bevises med substitusjonsmetoden. Prøv å gjør beviset så fullstendig som mulig.

D

Hva er kjøretiden til algoritmen ved bestefall input? Sett opp en recurrence-likning og utfør et matematisk resonnement for å komme frem til svaret. Du trenger ikke å bevise at gjetningen er korrekt.

2 Sortering (20%)

I denne delen skal du vise at du har oversikt over de ulike sorteringsalgoritmene vi har gjennomgått kurset.

Oppgave 1 - Velge Sorteringsalgoritme (10%)

I oppgavene under er det gitt enten et sett med eksempler på, eller en beskrivelse av, lister som skal sorteres. Du skal i hvert tilfelle forklare hvilke sorteringsalgoritmer som er passende, og i de tilfellene hvor flere er aktuelle skal du argumentere for hvilken som ville vært aller best.

A

Lister med rundt en million heltall i intervallet $[0, 10^{15}]$.

B

- $[1, 3, 2, 4, 6, 5, 7, 9, 8]$
- $[1, 3, 4, 6]$
- $[2, 4, 8, 5, 9, 7, 1, 6, 5, 4, 8, 9, 7, 6, 6, 1, 2, 1, 4]$
- $[3, 4, 6, 5, 1, 2, 9, 1, 1, 4, 6, 5]$
- $[4, 8, 5, 1, 6, 8, 9, 7, 6, 6, 1, 2, 1, 4]$
- $[9, 3, 2, 5, 6, 1, 7, 4, 8]$
- $[2, 5, 6, 4, 8, 5, 9, 7, 1, 4, 8, 5, 1, 6, 8, 9, 7, 6, 6, 1, 2, 1, 4, 6, 5, 4, 8, 9, 7, 6, 6, 1, 2, 1, 4]$
- $[2, 5, 5, 9, 7, 1, 8, 5, 1, 6, 8, 9, 7, 8, 5, 1, 6, 5, 4, 8, 8, 5, 1, 9, 7, 6, 6, 1, 2, 1, 4]$
- $[9, 7, 1, 8, 7, 1, 7, 7, 1, 4, 1, 4, 4, 5, 7, 1, 4, 8, 5, 1, 1, 8, 5, 1, 2]$
- $[3, 4, 6, 5, 1, 9, 8]$
- $[2, 5, 4, 6, 5, 1, 6, 7, 4, 8, 4, 8, 5, 9, 7, 1, 4, 8, 5, 1, 6, 8, 9, 7, 6, 4, 6, 5, 1, 6, 1, 2, 1, 4, 6, 4, 6, 5, 1, 8, 5, 9, 7, 1, 6, 5, 5, 4, 8, 9, 4, 8, 5, 1, 6, 9, 7, 6, 7, 6, 6, 4, 6, 5, 1, 1, 2, 1, 4]$
- $[6, 5, 1, 9, 8, 6, 6, 1, 2, 1, 4, 8, 5, 1, 6, 8]$

C

Lister med mellom en million og ti millioner flyttal med normalfordelt distribusjon.

D

- $[111, 35, 2]$
- $[451, 259, 261, 71]$
- $[2056101, 4052303]$
- $[34165, 1289, 1146501]$
- $[84851, 168917, 68612, 14]$
- $[93136, 171, 8]$
- $[2564, 859, 714, 851]$
- $[661, 65, 97, 12]$
- $[34651, 98]$
- $[65198, 6612, 148516, 8]$
- $[71564, 18519, 410]$
- $[851, 917, 68, 14]$

Oppgave 2 - MERGE-SORT (10%)

Denne oppgaven tar for seg MERGE-SORT algoritmen, som er listet under sammen med hjelpefunksjonen MERGE.

MERGE(A, p, q, r)

```
1  n_1 = q - p + 1
2  n_2 = r - q
3  let L[1..n_1 + 1] and R[1..N_2 + 1] be new arrays
4  for i = 1 to n_1
5       L[i] = A[p + i - 1]
6  for j = 1 to n_2
7       R[j] = A[q + j]
8  L[n_1 + 1] = ∞
9  R[n_2 + 1] = ∞
10 i = 1
11 j = 1
12 for k = p to r
13      if L[i] ≤ R[j]
14           A[k] = L[i]
15           i = i + 1
16      else A[k] = R[j]
17           j = j + 1
```

MERGE-SORT(A, p, r)

```
1  if p < r
2       q = ⌊(p + r)/2⌋
3       MERGE-SORT(A,p,q)
4       MERGE-SORT(A,q+1,r)
5       MERGE(A,p,q,r)
```

A

Vis så detaljert som mulig hvordan algoritmen ville kjørt på følgende input. Her kan det være greit å basere seg på hva som kommer inn og hva som kommer ut av hvert funksjonskall.

$$[41, 3, 16, 9, 15, 1, 21, 82, 7, 12]$$

B

Hvilke fordeler og ulemper har MERGE-SORT i forhold til QUICKSORT? Trekk inn relevant teori fra kurset.

3 Datastrukturer (20%)

I denne delen skal du vise at du har oversikt over ulike datastrukturene vi har gjennomgått i kurset.

Oppgave 1 - Kollisjonshåndtering i Hashtabeller (10%)

Denne oppgaven tar for seg ulike typer kollisjonshåndtering hashtabeller. Vi har en hashtabell med 10 plasser, og vi benytter en hashfunksjon som er definert som følger.

$$h(k) = k \mod 10$$

Vi har følgende liste med heltall som skal settes inn i den tenkte hashtabellen.

$$[1, 3, 12, 53, 91, 103]$$

A

Tegn slik hashtabellen vil se ut etter innsetting av tallene ved bruk av *lenking (chaining)* for kollisjonshåndtering.

B

Tegn slik hashtabellen vil se ut etter innsetting av tallene ved bruk av *åpen adressering* og *lineær probing* for kollisjonshåndtering.

C

Tegn slik hashtabellen vil se ut etter innsetting av tallene ved bruk av *åpen adressering* og *kvadratisk probing* for kollisjonshåndtering.

Oppgave 2 - Søketrær (10%)

I denne oppgaven dreier seg om innsetting i søketrær. Vi har følgende elementer som skal settes inn i trærne.

$$[10, 6, 4, 1, 5, 11, 2, 19, 12, 7, 3]$$

A

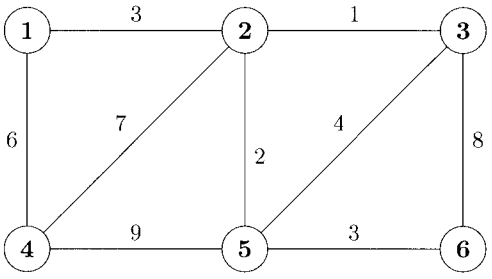
Tegn det binære søketreet som ville vært resultatet av å sette inn elementene i den angitte rekkefølgen.

B

Tegn det rødsvarte søketreet som ville vært resultatet av å sette inn elementene i den angitte rekkefølgen.

4 Grafer (20%)

I denne delen skal du vise at du har oversikt over de ulike grafalgoritmene vi har gjennomgått i kurset. Under er en urettet vektet graf som skal benyttes i alle oppgavene under.



Oppgave 1 - Kruskal (10%)

Denne oppgaven dreier seg om KRUSKAL algoritmen for å bygge minimum spenntrær som er listet under. Hjelpefunksjonene MAKE-SET, FIND-SET og UNION er operasjoner for å henholdsvis lage et disjunkt (disjoint) sett, for å finne hvilket disjunkt sett et element tilhører, og for å slå sammen to disjunkte sett.

```
KRUSKAL(G, w)
1  A = ∅
2  for each vertex v ∈ G.V
3       MAKE-SET(v)
4  sort the edges of G.E into nondecreasing order by weight w
5  for each (u, v) taken from the sorted list
6       if FIND-SET(u) ≠ FIND-SET(v)
7            A = A ∪ {(u, v)}
8            UNION(u, v)
9  return A
```

A

Du skal vise hvordan KRUSKAL algoritmen kjører på den gitte grafen over ved å inkrementelt bygge opp det som til slutt vil bli det resulterende spenntreet. Det kan være fordelaktig å også tegne opp relevante hjelpestrukturer.

B

Du skal modifisere KRUSKAL algoritmen slik at den ikke slår sammen kanter som har en vekt som er større enn gjennomsnittet av vekten på alle kantene. Dette bør kunne gjøres uten å øke den asymptotiske kjøretiden. Du skal vise den modifiserte pseudokoden og hvordan den modifiserte algoritmen ville kjørt på den gitte grafen over.

Oppgave 2 - Korteste Vei (10%)

I denne oppgaven skal dere vise at dere vet hvordan DIJKSTRA og BELLMAN-FORD algoritmene for korteste vei fungerer. Begge algoritmene er listet under. Hjelpefunksjonen INIT-SINGLE-SOURCE er ansvarlig for å initialiserer estimatet for avstand og forelderpekeren i alle nodene, og RELAX oppdaterer estimatet og

forelderpekeren hvis en kortere vei har blitt funnet. Hjelpefunksjonen EXTRACT-MIN henter ut den noden med korteste vei fra prioritetskøen.

```
BELLMAN-FORD(G, w, s)
1  INIT-SINGLE-SOURCE(G, s)
2  for i = 1 to |G. V| − 1
3       for each edge (u, v) ∈ G. E
4            RELAX(u, v, w)
5  for each edge (u, v) ∈ G. E
6       if v. d > u. d + w(u, v)
7            return FALSE
8  return TRUE
```

```
DIJKSTRA(G, w, s)
1  INIT-SINGLE-SOURCE(G, s)
2  S = ∅
3  Q = G. V
4  while Q ≠ ∅
5       u = EXTRACT-MIN(Q)
6       S = S ∪ {u}
7       for each vertex v ∈ G. Adj[u]
8            RELAX(u, v, w)
```

A

Vis hvordan BELLMAN-FORD ville kjørt på grafen gitt over ved å vise hvordan korteste-vei-treet og estimatene oppdateres underveis. Start søket i node 1.

B

Vis hvordan DIJSTRA ville kjørt på grafen gitt over ved å vise hvordan korteste-vei-treet og estimatene oppdateres underveis. Start søket i node 1. Det kan være hjelpsomt å også tegne relevante hjelpestrukturer.

C

Forklar hva som skiller BELLMAN-FORD og DIJKSTRA algoritmene for korteste vei fra hverandre.