Oppgavesettet har fire deler. Hver del er merket med en prosent som angir hvor stor andel hver del utgjør av hele oppgavesettet. Den første delen har dobbelt så mange oppgaver som de siste tre, så sørg for å sette av nok tid til denne.

Oppgavesettet inneholder en del oppgaver som krever tekstlige besvarelser. Prøv å svar så kort og konsist som mulig på disse slik at dere ikke kommer i tidsnød.

1 Analyse (40%)

I denne delen skal du ta i bruk de teknikkene vi har gjennomgått i kurset for å analysere algoritmer. Delen består av fire oppgaver hvor alle bortsett fra en har mindre deloppgaver.

Oppgave 1 - Enkel Analyse

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  i = 1
3  while i ≤ n
4      sum = sum + i
5      i = i + 2
6  return sum
```

OPPGAVE1C(n)

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

```
OPPGAVE1D(n)
1  sum = 0
2  i = 1
3  while i ≤ n
4       for j = 1 to n
5            sum = sum + i
6       i = i + 2
7  return sum
```

Oppgave 2 - Faktisk Kjøretid

I den forrige opgaven forenklet vi den detaljerte analysen ved at vi sa at hver programlinje tar en tidsenhet å utføre. Forklar hvorfor dette er en forenkling og hvordan en detaljert analyse uten forenklinger ville skilt seg fra den som ble gjort over. Du trenger ikke å utføre noen ny analyse, men gi gjerne noen eksempler.

Oppgave 3 - Avansert Analyse

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

```
OPPGAVE3(A, x, p, q)
1  if p ≥ q    // Sjekker om vi har nådd basistilfellet
2       return A[p] == x
3  m = ⌊(p + q)/2⌋
4  if A[m] == x
5       return TRUE
6  if x < A[m]
7       return OPPGAVE3(A, x, p, m − 1)
8  else
9       return OPPGAVE3(A, x, m + 1, q)
```

A

Beskriv med egne ord hva algoritmen gjør, og hvilke endringer, eller eventuelt hvilke restriksjoner, som skal til for å gjøre algoritmen til en fungerende søkealgoritme som returnerer hvorvidt et element blir funnet eller ikke.

B

Sett opp en recurrence-likning som beskriver kjøretiden til algoritmen over. Du trenger ikke ta hensyn til at algoritmen kan avbrytes hvis korrekt element blir funnet underveis i.e. anta at algoritmen må rekursere helt til basistilfellet inntreffer.

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 matematisk induksjon. Prøv å gjør beviset så fullstendig som mulig.

D

Hvorfor er det passende å bare se på den øvre grensen for algoritmen over? Prøv og knytt argumentasjonen opp i mot teorien i kurset.

Oppgave 4 - Masterteoremet

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 benytte masterteoremet for å finne den asymptotiske grensen, og i de oppgavene det ikke kan benyttes skal du forklare hvorfor.

A

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 4T(n/2) + \Theta(n) & \text{if } n > 1 \end{cases}$$

B

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

C

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

2 Sortering (20%)

I denne delen skal du vise at du har oversikt over de ulike sorteringsalgoritmene vi har gjennomgått kurset. Delen består av to oppgaver med mindre deloppgaver.

Oppgave 1 - Velge Sorteringsalgoritme

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

- $[1, 3, 2, 4, 6, 5, 7, 9, 8]$
- $[14, 16, 15, 17, 19, 22, 20, 24, 25, 28, 29, 11]$
- $[2, 4, 7, 5, 11, 17, 19, 20, 28, 29, 1, 31, 32, 37, 40, 41, 39, 44, 55, 58, 59, 49, 60, 61]$
- $[3, 4, 6, 5, 1, 2, 9, 10, 11, 14, 16, 15]$

C

Lister med mellom en million og ti millioner elementer med flyttal med ukjent distribusjon.

D

Lister med rundt hundre tusen elementer med flyttal med uniform distribusjon.

Oppgave 2 - Quicksort

Denne oppgaven tar for seg QUICKSORT algoritmen, som er listet under sammen med hjelpefunksjonen PARTITION.

```
PARTITION(A, p, r)
  x = A[r]
  i = p − 1
  for j = p to r − 1
      if A[j] ≤ x
          i = i + 1
          exchange A[i] with A[j]
  exchange A[i + 1] with A[r]
  return i + 1

QUICKSORT(A, p, r)
  if p < r
      q = PARTITION(A, p, r)
      QUICKSORT(A, p, q − 1)
      QUICKSORT(A, q + 1, r)
```

A

Vis hvordan algoritmen ville kjørt på følgende input.

$$[4, 3, 6, 9, 5, 11, 8, 7, 1, 2]$$

B

Hva er poenget med å ha en randomisert utgave av QUICKSORT? Kan du komme på et annet alternativ for å oppnå tilsvarende effekt?

3 Datastrukturer (20%)

I denne delen skal du vise at du har oversikt over ulike datastrukturene vi har gjennomgått i kurset. Delen består av to oppgaver med mindre deloppgaver.

Oppgave 1 - Hashtabeller

Denne oppgaven tar for seg hashfunksjoner og hashtabeller.

A

Beskriv hva en hashfunksjon gjør og hva som kjennetegner en god hashfunksjon.

B

Forklar når det er passende å ta i bruk hashtabeller fremfor direkteadresserte tabeller.

C

Gi en kort beskrivelse av to teknikker for hvordan man kan håndtere kollisjoner i en hashtabell.

Oppgave 2 - Søketrær

Denne oppgaven dreier som de ulike variantene av søketrær vi har gått gjennom i kurset.

A

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

$$[10, 16, 4, 1, 5, 11, 2, 19, 12, 7, 14]$$

B

Forklar forskjellen mellom binære søketrær og rød-svarte søketrær. Trekk inn de aspektene ved rød-svarte trær som skiller seg fra binære trær og prøv å forklar

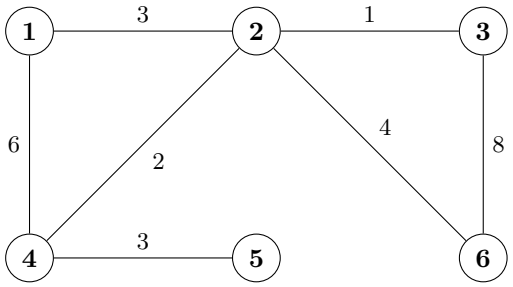
hvordan de benyttes til å oppnå en forbedring.

C

B-trær er en tredje variant av søketrær. Forklar hvordan de fungerer og hvordan de skiller seg fra de to andre variantene.

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 - Søk

Vi har gjennomgått algoritmer for både dybde-først og bredde-først søk i grafer. I denne oppgaven skal du vise uten å følge en spesiell algoritme at du har forstått hvordan slike søk utføres.

A

Utfør et bredde-først søk med 1 som startnode. Du skal vise fremgangen til søket ved å bygge det resulterende bredde-først treet inkrementelt i.e du skal vise hvert steg i byggeprosessen.

B

Utfør et dybde-først søk med 1 som startnode. Du skal vise fremgangen til søket ved å bygge det resulterende dybde-først treet inkrementelt.

C

Forklar hvordan et dybde-først søk kan brukes som et grunnlag for å sortere en graf topologisk.

Oppgave 2 - Minimum Spenntrær

KRUSKAL og PRIM algoritmen for å bygge minimum spenntrær er begge 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. Hjelpefunksjonene INSERT, DECREASE-KEY og EXTRACT-MIN er operasjoner for å henholdsvis legge til et element i en prioritetskø, for å minke nøkkelen til et element allerede i køen, og for å hente ut minste element.

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

```
PRIM(G, w, r)
  Q = ∅
  for each u ∈ G.V
      u.key = ∞
      u.π = NIL
      INSERT(Q, u)
  DECREASE-KEY(Q, r, 0)        // r.key = 0
  while Q ≠ ∅
      u = EXTRACT-MIN(Q)
      for each v ∈ G.Adj[u]
          if v ∈ Q and w(u, v) < v.key
              v.π = u
              DECREASE-KEY(Q, v, w(u, v))
```

A

Vis hvordan KRUSKAL algoritmen ville kjørt på grafen gitt over. Du skal vise fremgangen til algoritmen ved å inkrementelt legge til kanter til det som til slutt vil bli det resulterende spenntreet.

B

Vis hvordan PRIM algoritmen ville kjørt på grafen gitt over. Du skal også her vise fremgangen ved å vise hvordan kanter legges til inkrementelt, men her kan det også være fordelaktig å holde orden på sorteringsnøkkelen til hver node (noden v har nøkkel $v.key$ i pseudokoden). Velg 1 som rotnode.