Informasjon

Du måtte klare 8 av 18 oppgaver for å få øvingen godkjent.

Finner du feil eller mangler eller har forslag til forbedringer, ta gjerne kontakt!

Oppgave 2

$\log_a a^n = n$ og $\log_2 1024 = 10$ er riktig, siden $\log_a x = n$ er det samme som at $x = a^n$.

Kommentar:
Det er mulig å tenke at $\log_a b = \frac{\log_c a}{\log_c b}$ skal være riktig, men dette stemmer ikke. Det er derimot inversen, $\log_a b = \frac{\log_c b}{\log_c a}$, som er et gyldig uttrykk. Se for eksempel her for en forklaring av hvorfor.

Oppgave 3

$a - 1$ er riktig, siden $a \bmod n = r$ er det samme som at $a = xn + r$, hvor $0 \leqslant r < n$. Dermed har vi at $a = xn + 1$, og $a - 1 = xn$.

Oppgave 4

konstant, logaritmisk, lineær, linearitmisk, kvadratisk, kubisk, eksponentiell, faktoriell er riktig rekkefølge.

Kommentar:
Det kan hjelpe å skrive ut de forskjellige kjøretidsklassene med asymptotisk notasjon. For den riktige rekkefølgen blir dette $\Theta(1), \Theta(\lg n), \Theta(n), \Theta(n \lg n), \Theta(n^2), \Theta(n^3), \Theta(2^n), \Theta(n!)$. Da blir det plutselig lettere å sammenligne klassene. Det kan også hjelpe å tegne inn de forskjellige klassene i en graf, slik som, for eksempel, Wikipedia har gjort her.

Oppgave 5

$f(n) = \mathrm{O}(g(n))$, $f(n) = \Omega(g(n))$ og $f(n) = \Theta(g(n))$ er riktig, siden

$$f(n) = \lg n^{\lg 5} = \lg 5 \lg n = \lg 5^{\lg n} = g(n)$$

Oppgave 6

$g(n) = \mathrm{O}(f(n))$ og $f(n) = \omega(g(n))$ er riktig.

Kommentar:
I praksis er det kjent at $n^2 = \mathrm{O}(n^2 \lg n + n + n \lg n)$ og at $n^2 \lg n + n + n \lg n = \omega(n^2)$, men det er selvfølgelig mulig å vise dette også.

Vi kan se at $n^2 \lg n + n + n \lg n > n^2 \lg n > n^2$ for $n > 2$. Dette betyr at $0 \leqslant g(n) \leqslant f(n)$ for alle $n > 2$ ($c = 1$), og vi har av definisjonen til O-notasjon at $g(n) = \mathrm{O}(f(n))$.

For å vise at $f(n) = \omega(g(n))$, må vi vise at $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$.

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{n^2 \lg n + n + n \lg n}{n^2} = \lim_{n\to\infty} \lg n + \frac{1}{n} + \frac{\lg n}{n} = \infty + 0 + 0 = \infty$$

Så av definisjon er $f(n) = \omega(g(n))$.

Oppgave 7

$\Theta(n^3)$ er riktig.

Kommentar:
Her består den asymptotiske notasjonen av to deler. En $\mathrm{O}(n^2)$ del og en $\Theta(n^3)$ del. Vi kan se at $\Theta(n^3)$ delen vil dominere $\mathrm{O}(n^2)$ delen, siden $\mathrm{O}(n^2)$ maksimalt kan ha størrelsesorden n^2, mens $\Theta(n^3)$ alltid har størrelsesorden n^3. Dermed kan vi forenkle bort $\mathrm{O}(n^2)$ delen uten å tape noen presisjon. Det vil si at vi har like tette grenser i det forenklede uttrykket.

Oppgave 8

$\Omega(n^3)$ er riktig.

Kommentar:
Her består den asymptotiske notasjonen av to deler. En $\Omega(n^2)$ del og en $\Theta(n^3)$ del. Til forskjell fra forrige oppgave så dominerer ikke $\Theta(n^3)$ delen $\Omega(n^2)$ delen. $\Omega(n^2)$ forteller oss at vi har et uttrykk med størrelsesorden på minst n^2, men at det ikke er noe som hindrer denne delen fra å ha størrelsesorden på for eksempel n^{100}. Dermed må vi ha med dette videre i forenklingen. Vi vet også at $\Theta(n^3)$ garanterer at størrelsesordenen til kjøretiden til algoritmen alltid er minst n^3. Det vil si at det eneste vi vet om kjøretiden er at $\Omega(n^2)$ tilsier at den kan være hva som helst med størrelseorden større eller lik n^2 og $\Theta(n^3)$ forteller oss at den alltid har et ledd med størrelsesorden n^3. Dermed har vi ikke mistet noe informasjon ved å forenkle til $\Omega(n^3)$, som sier at vi alltid har størrelsesorden n^3 eller større.

Oppgave 9

$\Omega(1) + \mathrm{O}(n!)$, $\Omega(n \lg n) + \Theta(n^2)$, $\Omega(n^2) + \mathrm{O}(n^4)$, $\Omega(n \lg n) + \mathrm{O}(n^4)$ og $\Omega(n) + \Theta(n^2) + \mathrm{O}(n^4)$ er riktig.

Kommentar:
Her er det viktig å tenke på hva det betyr at et asymptotisk uttrykk er likt et annet. Det vil si at for hver mulige tolkning av de anonyme funksjonene på venstresiden av likehetstegnet så må man kunne velge tolkninger av de anonyme funksjonene på høyresiden av likehetstegnet slik at venstresiden blir lik høyresiden. Altså, venstresiden må være minst like presis som høyresiden og høyresiden må alltid inneholde venstresiden. Det står mer detaljert om disse definisjonene på denne FAQ-posten om asympotisk notasjon.

Det er lett å tenke at $\Omega(1) + \mathrm{O}(n!)$ ikke kan være gyldig siden dette uttrykket er så generelt. Men, alle tolkninger av $\Theta(n^2) + \mathrm{O}(n^4) + \Omega(\lg n)$ finnes også i $\Omega(1) + \mathrm{O}(n!)$. Det er absolutt ikke en veldig nyttig omskriving, men viser hvor generell definisjonen av asymptotisk notasjon er.

Oppgave 10

«Sammenhengen mellom den generelle kjøretiden og $g(n)$ kan beskrives med asymptotisk notasjon, men ingen av uttrykkene i denne oppgaven stemmer.» er riktig.

Kommentar:
Den gjennomsnittlige kjøretiden til en algoritme er kun kjøretiden man i gjennomsnitt vil få gitt et tilfeldig plukket input. Den gir derfor ikke en nøyaktig beskrivelse av den generelle kjøretiden til en algoritme. Det kan finnes input som har bedre, dårligere eller like god kjøretid som den gjennomsnittlige kjøretiden. Derfor kan vi ikke på generell basis si at den generelle kjøretiden er bundet av $g(n)$ direkte. Men, den gjennomsnittlige kjøretiden forteller oss noe indirekte om generelle kjøretiden til algoritmen. Den forteller oss at kjøretiden i verste tilfelle ikke kan være bedre enn den gjennomsnittlige kjøretiden og at kjøretiden i beste tilfelle ikke kan være dårligere enn den gjennomsnittlige kjøretiden. Altså, kan man beskrive den generelle kjøretiden ved hjelp av uttrykkene $\mathrm{O}(\Omega(g(n)))$ og $\Omega(\mathrm{O}(g(n)))$. Det vil si, kjøretiden til algoritmen er bundet nedenfra av noe som er bundet av $\mathrm{O}(g(n))$ og ovenfra av noe som er bundet av $\Omega(g(n))$.

Oppgave 11

$g(n) = \omega(n^2)$ og $h(n) = \Omega(n)$ er riktig.

Kommentar:
Basert på den asympotiske notasjonen her er det umulig å si noe nøyaktig om kjøretiden til algoritmen i beste og verste tilfelle. Dette skyldes at vi har dette $\Omega(n)$-leddet som ikke har noen øvre grense (hvert ledd kan tenkes på som separat). Dermed kan det hende at algoritmen har en faktisk kjøretid på, for eksempel, $\Theta(n^{100})$ eller $\Theta(n^2 \lg n)$. Det eneste vi dermed kan si noe om er hva den nedre grensen på kjøretiden i beste og verste tilfelle kan være. Vi vet at denne er $\Omega(n^2 \lg n)$, grunnet garantien gitt av leddet $\Theta(n^2 \lg n)$.

Oppgave 12

$O(n^2)$ er riktig, siden INSERTION-SORT har en kjøretid på $\Omega(n)$ i beste tilfelle og en kjøretid på $O(n^2)$ i verste tilfelle.

Oppgave 13

```python
def insertion_sort(A):
    for i in range(1, len(A)):
        while i > 0 and A[i] < A[i - 1]:
            # Python triks for å bytte om to variabler
            A[i], A[i - 1] = A[i - 1], A[i]
            i -= 1
    return A
```

Oppgave 14

4 er riktig, siden det da vil være igjen 1 fyrstikk. Siden den andre spilleren må plukke minst 1 fyrstikk, så garanterer dette at vi vinner.

Oppgave 15

7 er riktig.

Kommentar:
Siden vi har flere enn 8 fyrstikker, så er det ikke like lett å se hvorfor 7 garanterer oss å vinne. Det er mulig å teste alle mulige varianter å spille på, og velge den som gir oss muligheten til å vinne uavhengig av hvordan den andre spilleren plukker. Dette blir tungvindt, selv med så få fyrstikker som 32. For vår variant av Nim, så finnes det faktisk en generell formell som kan si hvor mange fyrstikker vi må plukke for å garantere at vi vinner. Denne formelen er som følger:

$$(n-1) \bmod 8$$

Hvis formelen gir ut 0 som svar, så betyr det at uansett hvor mange fyrstikker vi plukker, så kan vi ikke garantere å vinne.

Denne formelen kan virke litt tilfeldig, med fungerer egentlig på samme måte som strategien vi hadde brukt hvis det var 8 eller færre fyrstikker igjen. I det tilfellet, valgte vi fyrstikker slik at det var kun 1 igjen. Denne formelen prøver å la det være igjen et multiplum av 8 fyrstikker, pluss en ekstra fyrstikk. Dette, da vi deretter alltid kan garantere at det er et multiplum av 8 fyrstikker pluss en ekstra fyrstikk igjen på den andre spillerens tur. Det gjør vi ved å plukke $8-k$ fyrstikker, hvor k er antall fyrstikker den andre spilleren plukket. Fortsetter vi slik, ender vi til slutt opp med at det er 1 fyrstikk igjen på den andre spillerens tur, og vi vinner dermed. Dette er kun mulig, siden den andre spilleren kan plukke mellom 1 og 7 fyrstikker. Hadde antall som kan plukkes vært annerledes, måtte vi endret strategi.

Siden den andre spilleren kan følge samme strategi, kan vi ikke garantere at vi vinner hvis det er et multiplum av 8 fyrstikker, pluss en igjen på turen vår. Altså, hvis $(n-1) \bmod 8 = 0$. I den situasjonen, ville den andre spilleren alltid vinne ved perfekt spill.

Oppgave 16

«Det er ikke sikkert jeg vinner, uansett hvor mange jeg tar.» er riktig, siden $345 - 1 \bmod 8 = 0$.

Oppgave 17

```python
def take_pieces(n_pieces):
    take = (n_pieces - 1) % 8
    if take != 0:
        return take
    # Hvis vi ikke kan garantere et vinn, plukk 1
    return 1
```

Kommentar:
Koden kan forenkles med et Python triks

```python
def take_pieces(n_pieces):
    # Returnerer (n_pieces - 1) % 8 hvis det ikke er 0, ellers 1
    return (n_pieces - 1) % 8 or 1
```

Merk: For dette problemet er det ikke mulig å teste alle mulige varianter, selv ikke for små verdier. Det var mulig å løse oppgaven uten å finne formelen, men det krevde noen form for husking av tidligere regnet ut svar. Man kunne, for eksempel, anvende *memoisering* eller utregning med iterasjon (fra 1 og oppover), men disse teknikkene blir ikke dekket før forelesning 6.

Oppgave 18

Hvis $(n-1) \bmod 8 \neq 0$ kan vi garantere at vi vinner. Dette er den samme formelen som ble utledet i forklaringen av oppgave 13. Beviset, ved induksjon, følger:

Grunntilfelle $n = 1$ ($0 \bmod 8 = 0$): Du må plukke en fyrstikk, og taper derfor alltid.
Grunntilfelle $n = 2$ ($1 \bmod 8 \neq 0$): Du kan plukke en fyrstikk, og da er det kun en igjen. Den andre spilleren taper.
Grunntilfelle $n = 3$ ($2 \bmod 8 \neq 0$): Du kan plukke to fyrstikker, og da er det kun en igjen. Den andre spilleren taper.

$\vdots$

Grunntilfelle $n = 8$ ($7 \bmod 8 \neq 0$): Du kan plukke syv fyrstikker, og da er det kun en igjen. Den andre spilleren taper.

Induksjonstilfelle $(k-1) \bmod 8 \neq 0$, $k > 8$: Anta at formelen gjelder for $1 \leqslant n < k$. Da kan vi plukke $(k-1) \bmod 8$ fyrstikker, siden $(k-1) \bmod 8 \neq 0$, så $(k-1) \bmod 8 \in \{1, 2, 3, 4, 5, 6, 7\}$.

I tillegg er $k - ((k-1) \bmod 8) > 1$, så formelen gjelder for $k - ((k-1) \bmod 8)$. Av definisjon er $(k - (k-1) \bmod 8 - 1) \bmod 8 = 0$, så det følger at vi kan garantere at vi vinner.
Induksjonstilfelle $(k-1) \bmod 8 = 0$, $k > 8$: Anta at formelen gjelder for $1 \leqslant n < k$. Uansett hvor mange fyrstikker vi plukker, $i \in \{1, 2, 3, 4, 5, 6, 7\}$, så er følgende sant $(k - i - 1) \bmod 8 \neq 0$. Siden $k - i > 1$, så sier formelen at den andre spilleren kan med perfekt spill garantere at hen vinner. Dermed kan vi ikke for $(k-1) \bmod 8 = 0$ garantere at vi vinner.

Merk: Det er mulig, og helt riktig, å heller bruke den alternative formelen, $n \bmod 8 \neq 1$. Fremgangsmåten i beviset vil være den samme, med kun noen mindre forskjeller i argumentene.

Oppgave 19

$\mathrm{O}(n^2)$, $\mathrm{O}(nk)$ og $\Omega(n + k^2)$ er riktig.

Kommentar:
Den oppgaven kan løses ved å telle operasjoner, slik som CLRS (læreboken) gjør for Insertion-Sort på side 26 (se gjerne *Analysis of Insertion sort*, s. 24 i CLRS). Hvis vi gjør dette, så ser vi at kjøretiden til K-Largest blir $\mathrm{O}(nk)$ i verste tilfelle. Dette skyldes at for-løkken på linje 4 vil kjøre n ganger, og for hver av disse n gangene kan while-løkken på linje 8 kjøres k ganger. Husk også at $\mathrm{O}(nk) = \mathrm{O}(n^2)$, siden $k \leqslant n$.

For svaret $\Omega(n + k^2)$, kan man ikke se på kjøretiden i verste tilfelle. Da må man heller se på kjøretiden i beste tilfelle, som ikke alltid er like lett å regne ut. K-Largest vil alltid iterere over A, og derfor må kjøretiden være bundet av minst $\Omega(n)$. Denne delen av kjøretiden vil alltid overskygge for-løkkene på linjene 2 og 12, og derfor kan vi ignorere disse for-løkkene her. I for-løkken på linje 4, vil if-en på linje 5 være sann for minst k siffere i A. Dette skyldes at b er fylt med nuller til å starte med. Hvis de k største nummerene i A er de k første nummerene i A, så vil denne if-en kjøres kun k ganger. I tillegg, vil while-løkken for hvert nummer kunne iterere opptil $k-1$ ganger. Vi vet at de første k nummeren som blir håndtert må i hvert fall bruke $k-1, k-2, \ldots, 0$ iterasjoner, siden $a_i > 0$ og B er originalt fylt med nuller. Dermed vil de første k nummerene i snitt kreve minst $\frac{k-1}{2}$ iterasjoner. Det vi si at i det beste tilfellet har vi minst $k\frac{k-1}{2}$ iterasjoner av while-løkken. Dermed vil kjøretiden i beste tilfelle ikke kunne være bedre enn $\Omega(n + k^2)$. Denne argumentasjonen dekker alle deler av algoritmen, derfor vil dette være kjøretiden i beste tilfelle. Spesielt, så vet vi at dette tilfellet vil inntreffe hvis A er sortert i synkende rekkefølge.

Oppgaver som handler om å finne enten de k største eller minste tallene i en liste har blitt gitt flere ganger på tidligere eksamener. Senest ved konten sommeren 2020 ble det gitt en slik oppgave.

Oppgave 20

Vi ønsker å vise at K-Largest finner summen av de k største elementene i listen. Grunnet linjer 11, 12 og 13 i algoritmen, vet vi at dette kun stemmer hvis B består av de k største tallene i A etter at løkken over linjer 4 til og med 10 er utført. Vi vet også at linjer 1, 2 og 3 garanterer for at B består av k nuller når vi kommer til linje 4. Vi må altså vise at løkken på linjer 4 til og med 10 gjør om B fra å bestå av k nuller til de k største tallene i A.

Løkkeinvariant: Etter iterasjon i gjelder følgende

(i) B består av de $\min(k, i)$ største tallene i A[1..i], samt $\max(k - i, 0)$ nuller.

(ii) B er sortert i stigende rekkefølge.

Grunntilfelle i = 1: Før iterasjon 1, vil B bestå av k nuller. Siden $a_i > 0$, vil A[1] være større enn B[1] og linjer 6 til og med 10 vil utføres. B vil dermed bestå av A[1] og $k - 1$ nuller, siden linje 6 bytter ut B[1] med A[1]. Siden linje 9 kun bytter om på rekkefølgen på elementer i B, gjelder (i) etter iterasjon 1. I while-løkken på linjer 8, 9 og 10, har vi til å starte med at A[i] = B[1] = B[$j-1$]. Så lenge B[$j - 1$] > B[j], så bytter løkken om B[$j - 1$] og B[j]. Dermed vil A[1] = B[$j - 1$] alltid gjelde på linje 8 og 9. Siden B[2..i] består av nuller før løkken, sørger løkken for at A[1] blir plassert i slutten av listen. Dermed er B sortert i stigende rekkefølge etter iterasjon 1 og (ii) gjelder.

Induksjonstilfelle $1 < i \leqslant k$: Anta at løkkeinvarianten gjelder etter iterasjon $i - 1$, da vil (i) og (ii) være gyldige for $i - 1$ på starten av iterasjon i. Siden $i \leqslant k$, finnes det minst en null i B. Dermed vil A[i] > B[1] og linjer 6 til og med 10 vil bli utført. Linje 6 bytter ut B[1] med A[i]. B[1] må være en null, og dermed vil det være $k - i$ nuller igjen i B, samt i tall fra A[1..i] i B, og (i) holder etter iterasjon i. Følger man samme logikk som i grunntilfellet over, vet man at A[i] = B[$j - 1$] på linjer 8 og 9. Siden, B[2..i] er sortert i stigende rekkefølge, vil resultatet av løkken på linje 8, 9 og 10 være at A[i] plasseres i B, slik at alle elementene før er mindre enn A[i] og alle elementene etter er større enn eller lik A[i]. Siden rekkefølgen på de andre elementene i B ikke endrer seg, vil B forbli sortert i stigende rekkefølge. Dermed holder (ii) etter iterasjon i.

Induksjonstilfelle $k < i \leqslant n$: Anta at løkkeinvarianten gjelder etter iterasjon $i - 1$, da vil (i) og (ii) være gyldige for $i - 1$ på starten av iterasjon i. Siden $i > k$ finnes det ingen nuller i B. Da finnes det to muligheter, (a) A[i] $\leqslant$ B[1] eller (b) A[i] > B[1]. For (a), så må B allerede inneholde de k største elementene i A[1..i], siden B er sortert og består av de k største elementene i A[1..($i - 1$)]. Dermed vil ikke A[i] være et av de k største elementene i A. I dette tilfellet, så vil algoritmen ikke gjøre noe mer og (i) og (ii) vil derfor holde etter iterasjon i. For (b), så vil A[i] være et av de k største elementene i A[1..i] og linje 6 vil sørge for at det minste av elementene i B blir byttet ut med A[i]. Dermed vil B bestå av de k største elementene i A[1..i], og (i) holder. På samme måte som i induksjonstilfellet $1 < i \leqslant k$, så vil løkken på linjer 8, 9 og 10 plassere A[i] i B, slik at B er sortert i stigende rekkefølge. Dermed holder også (ii) etter iterasjon i.

Merk: For å slippe å argumentere for while-løkken på linjer 8, 9 og 10 for hvert tilfelle, kunne man vist egenskapene til denne seperat før man begynte å vise den ytre løkken.

Oppgave 21

Denne oppgaven har mange løsninger, ikke alle vil være inkludert i løsningsforslaget.

En av de mest naturlige løsningene, er en algoritme som ligner på K-Largest, men ikke sørger for at B er sortert. En slik algoritme vil generelt være tregere enn K-Largest, da den krever at man alltid går over hele B for hver verdi i A. Altså kjøretiden til algoritmen vil være $\Theta(nk)$.

Et videre alternativ er å sortere listen først, for så å summere de k siste elementene i A. Ved bruk av Insertion-Sort til sortering, vil man kunne få en kjøretid på $\Theta(n)$ i beste tilfelle. Denne kjøretiden er bedre enn $\Theta(n + k^2)$, som er kjøretiden til K-Largest i beste tilfelle, men med Insertion-Sort vil kjøretiden bli $\Theta(n^2)$ i verste tilfelle, som er dårligere enn for K-Largest, som har $\Theta(nk)$ som kjøretid i verste tilfelle.

Hvis vi anvender andre sorteringsalgoritmer enn Insertion-Sort, kan vi oppnå en bedre kjøretid.

Hvis vi bruker MERGE-SORT (forelesning 3 og 2.3.1 i CLRS), vil vi få en kjøretid på $\Theta(n \lg n)$. Avhengig av verdien til k vil kjøretiden være bedre eller verre enn for K-LARGEST.

Det er mulig å oppnå en bedre kjøretid enn K-LARGEST, uten å anvende sortering. Dette kan gjøres ved å velge et tilfeldig element i listen, for så å dele A i en del som inneholder elementer som er større enn dette elementet og en del med de som er mindre enn dette elementet (antar for øyeblikket at det ikke finnes duplikater). Da ender man opp med to deler av A, *lower* og *upper*, separert av det elementet man valgte tidligere, element i. Hvis $i = n - k$, så kan man summere tallene i *upper*, da dette er de k største tallene i A i usortert rekkefølge. Ellers, hvis $i < n - k$ utfører man operasjonen på nytt på *upper*. Hvis $i > n - k$, så utfører man operasjonen på *lower*, men leter nå etter $k - (n - i)$ tall. Når man finner disse $k - (n - i)$ tallene, så summerer man disse sammen med *upper*. Denne prosessen fortsetter rekursivt til man har funnet de k største tallene. Hvis man velger helt tilfeldig, kan man vise at $\Theta(n)$ er både kjøretid i beste tilfelle og gjennomsnittlig kjøretid, mens kjøretiden i verste tilfelle er $\Theta(n^2)$. Dette er en modifikasjon av RANDOMIZED-SELECT fra boken (forelesning 4 og 9.2 i CLRS). Det finnes også mer informasjon om en slik algoritme her.

Det er mulig å velge et bedre elementet å dele listen på enn ved å plukke et tilfeldig element. Anvender man heller algoritmen SELECT (forelesning 4 og 9.3 i CLRS) ender opp med en kjøretid på $\Theta(n)$. I stedet for å plukke et tilfeldig element, finner denne noe som kalles «*median-of-medians*». Dette er en tilnærming til den faktiske medianen, og sørger for at man deler listen i to deler av omtrent samme størrelse.

Oppgave 22

$\Theta(n^2)$ er riktig.

Kommentar:
Algoritmen iterer $\lfloor \frac{n}{2} \rfloor$ ganger, og den i-te gangen itereres det over $n - 2i + 2$ elementer. Det vil si totalt:

$$\sum_{i=1}^{\lfloor \frac{n}{2} \rfloor} n - 2i + 2 = \left(n - \left\lfloor \frac{n}{2} \right\rfloor + 1\right) \left\lfloor \frac{n}{2} \right\rfloor \geqslant \left(\frac{n}{2} + 1\right) \left\lfloor \frac{n}{2} \right\rfloor = \Theta(n^2)$$

Dette betyr ikke at svaret er $\Theta(n^2)$, grunnet bruken av $\geqslant$. Det betyr kun at den generelle kjøretiden er $\Omega(n^2)$, men man kan på samme måte vise at

$$\left(n - \left\lfloor \frac{n}{2} \right\rfloor + 1\right) \left\lfloor \frac{n}{2} \right\rfloor \leqslant \left(\frac{n}{2} + 1\right) \left\lfloor \frac{n}{2} \right\rfloor + 1 = \Theta(n^2)$$

Merk: Det er mange riktig alternativer, men $\Theta(n^2)$ er det mest nøyaktige.

Oppgave 23

Det er ikke mulig å direkte vise sorteringen etter hver iterasjon. I steden for kan vi vise følgende løkkeinvariant.

Løkkeinvariant: Etter iterasjon i gjelder følgende:

(i) De i første elementene i A er sortert i stigende rekkefølge

(ii) De i siste elementene i A er sortert i stigende rekkefølge

(iii) $a_i \leqslant a_{n-i+1}$

(iv) $a_i \leqslant \min(a_{i+1}, a_{i+2}, \ldots, a_{n-i})$ og $\max(a_{i+1}, a_{i+2}, \ldots, a_{n-i}) \leqslant a_{n-i+1}$

Vi vil senere se at denne løkkeinvarianten garanterer for at A er sortert etter $\lfloor \frac{n}{2} \rfloor$ iterasjoner.

Grunntilfelle $i = 1$: DUAL-SORT plukker det minste og største elementet i midten av A, og bytter ut a_i og a_{n-i+1} henholdsvis med disse. I iterasjon 1 vil det si at a_1 blir det minste elementet i A og a_n blir det største elementet i A. Dermed må (iii) holde, da enten er $a_1 < a_n$ eller så består A av kun like elementer. Videre, må også (iv) holde, da a_1 er det minste elementet i A og a_n er det største elementet i A. Siden $i = 1$ vil alltid (i) og (ii) holde.

Induksjonstilfelle $1 < i \leqslant \lfloor \frac{n}{2} \rfloor$:
Anta at løkkeinvarianten holder etter iterasjon $i - 1$. Slik DUAL-SORT er definert, vil algoritmen i iterasjon i finne det minste og største elementet i $\langle a_i, a_{i+1}, \ldots a_{n-i+1} \rangle$. Deretter vil a_i og a_{n-i+1} bli byttet ut med disse henholdsvis. Altså må (iii) gjelde, da vi vet at $a_i < a_{n-i+1}$ eller at alle elementene i $\langle a_i, a_{i+1}, \ldots, a_{n-i+1} \rangle$ er like. På samme måte må også (iv) gjelde, da a_i er det minste elementet i denne delen av A og a_{n-i+1} er det største elementet.

Siden løkkeinvarianten holder for iterasjon $i - 1$, tilsier (iv) at etter iterasjon i må $a_{i-1} \leqslant a_i$, siden vi vet at $a_{i-1} \leqslant \min(a_i, a_{i+1}, \ldots, a_{n-i+1})$. Siden de $i - 1$ første elementene i A er sortert i stigende rekkefølge, og $a_{i-1} \leqslant a_i$, så må også de i første elementene i A være sortert i stigende rekkefølge. Altså holder (i). På samme måte, kan vi vise at $a_{n-i+1} \leqslant a_{n-i+2}$ og dermed siden (ii) sier at de $i - 1$ siste elementene i A er sortert i stigende rekkefølge må også de i siste elementene i A være sortert i stigende rekkefølge. Dermed holder også (ii) og løkkeinvarianten holder.

Nå som induksjonstilfellet har blitt vist, vet vi at etter $\lfloor \frac{n}{2} \rfloor$ iterasjoner gjelder (i), (ii), (iii) og (iv). Når n er et partall, betyr dette av de $\frac{n}{2}$ første elementene i A er sortert i stigende rekkefølge og at de $\frac{n}{2}$ siste elementene i A er sortert i stigende rekkefølge. Siden $a_{\frac{n}{2}} \leqslant a_{\frac{n}{2}+1}$, betyr det at A er sortert i stigende rekkefølge. Når n er et oddetall, derimot, vet vi at de først og siste $\frac{n-1}{2}$ elementene i A er sortert i stigende rekkefølge. Grunnet (iv), vet vi at $a_{\frac{n-1}{2}} \leqslant a_{\frac{n-1}{2}+1} \leqslant a_{\frac{n-1}{2}+2}$. Av dette følger det at hele A må være sortert i stigende rekkefølge. Dermed sorterer DUAL-SORT A i stigende rekkefølge for vilkårlige A.

Merk: I teorien bør man vise at DUAL-SORT også fungerer for lister med 0 eller 1 elementer. Siden DUAL-SORT ikke itererer før det er minst 2 elementer i listen, vil den ikke gjøre noe for lister med 0 eller 1 elementer, og derfor fungere helt fint i slike tilfeller.

Oppgave 25

For eksamensoppgavene, se løsningsforslaget til den gitte eksamenen.

For oppgaver i CLRS, så finnes det mange ressurser på nettet som har fullstendige eller nesten fullstendige løsningsforslag på alle oppgavene i boken. Det er ikke garantert at disse er 100% korrekte, men de kan gi en god indikasjon på om du har fått til oppgaven. Det er selvfølgelig også mulig å spørre studassene om hjelp med disse oppgavene.

Et eksempel på et ganske greit løsningsforslag på CLRS, laget av en tidligere doktorgradsstudent ved Rutgers, finnes her.