

Algoritmer og datastrukturer

Kondisjonstest

Repetisjonsoppgaver - **LF**

Onsdag 6. oktober 2004

Dette oppgavesettet er for det meste ment å drille på konsepter. Det er ikke representativt for *vanskelighetsgraden* på eksamen, men klarer du hele dette settet stiller du i alle fall sterkt! :-)

Disse repetisjonsoppgavene er ganske mange. Det er derfor mye å lese gjennom, men vi håper du blir tilsvarende drillet i dette stoffet.

Hvis du ikke rekker gjennom alle, er det jo bare å kose seg en annen gang :-)

PS! Bakerst er det med formelark.

Oppgave 1 - Designmetoder

Deloppgave 1a

Alles favoritt, tegn streker! Dra streker mellom designmetode og algoritmer som hovedsaklig benytter seg av designmetoden. Det skal være to streker fra hver designmetode, med mindre du velger å se på algoritmene på en annen måte en vi har valgt i LF. Det er ikke nødvendigvis åpenbart hvilken designmetode som passer best til å beskrive en algoritme.

Rå makt (brute force)	Flettesortering (merge sort)
	Sekvensielt søk (sequential search)
Splitt-og-hersk (divide and conquer)	Dybde-først-søk (depth-first search)
	Boblesortering (bubble sort)
Krymp-og-hersk (decrease and conquer)	Topologisk sortering (topologic sort)
	Binærsøk

Merk: Det er ikke nødvendigvis entydig for enhver algoritme hvilken designmetode som skal ha kreditt for den.

Rå makt: Sekvensielt søk og boblesortering.

Splitt-og-hersk: Binærsøk (kan med fordel også ses på som krymp-og-hersk, siden halvparten av elementene forkastes) og flettesortering.

Krymp-og-hersk: Topologisk sortering og dybde-først-søk.

I tillegg kan vi ha følgende alternative syn:

Sekvensielt søk: Krymp-og-hersk (vi fjerner ett element av gangen: decrease-by-a-constant)

Boblesortering: Krymp-og-hersk (vi fjerner ett eller flere elementer av gangen: variable-size-decrease), men dette er ikke en så klar krymp-og-hersk som sekvensielt søk/binærsøk

Binærsøk: Krymp-og-hersk (halvpartene av elementene fjernes av gangen: decrease-by-a-constant-factor)

Deloppgave 1b

Stryk det som ikke er en designmetode. Tenk kort gjennom hva som kjennetegner hver designmetode. (Du får velge selv om du vil tenke på de designmetodene som ikke er gjennomgått ennå)

- Forgren-og-begrens (branch-and-bound)
- Splitt-og-hersk (divide and conquer)
- Kombinér eksponensvekst (combine exponentiation)
- Endre-og-hersk (transform and conquer)
- Tilbakesporing (backtracking)

- Grådighet (greedy technique)
- Rå makt (brute force)
- Krymp-og-hersk (decrease and conquer)
- Dynamisk programmering (dynamic programming)
- Tid-plass-avveining (space-time tradeoffs)

Kombinér eksponensvekst er ikke en designmetode (og ikke noe annet heller). Kort beskrivelse av designmetoder, i bokas rekkefølge:

Rå makt (brute force): Sjekk alle løsninger.

Splitt-og-hersk (divide and conquer): Del opp inndata i flere delproblemer som løses hver for seg.

Krymp-og-hersk (decrease and conquer): Se på ett antall elementer fra inndata og eliminer elementer som mulige løsninger når de har blitt sett på.

Endre-og-hersk (transform and conquer): Forhåndsprosesser inndata før problemet løses (f.eks. sortér).

Tid-plass-avveining (space-time tradeoffs): Bruk ekstra plass for å tilby en kjappere løsning, som regel etter en engangskostnad for plass (f.eks. indeksering) eller bruk lenger beregningstid for å spare plass (f.eks. hvis data er for stor til å indekseres).

Grådighet (greedy technique): Velg det elementet som er umiddelbart best (f.eks. minspenntre).

Dynamisk programmering (dynamic programming): Beregn løsningen av delproblemer for å deretter å bruke disse beregningene til å løse hovedproblemet.

Tilbakesporing (backtracking): Tidsbesparende forbedring av dybde-først-søk som baserer seg på å bare evaluere de løvnode som potensielt kan gi en gyldig løsning. Søket ned til enhver løvnode blir avbrutt så fort en intern node på veien oppdager at løvnode umulig kan gi en gyldig løsning. Dette fører i praksis til at bare deler av søketreet utforskes, noe som ofte forbedrer kjørtiden betydelig. Binærsøk kan ses på som et ekstremt tilfelle av tilbakesporing.

Forgren-og-begrens (branch and bound): Som tilbakesporing, men istdf. gyldige barn må man overestimere en verdi for den teoretisk beste etterkommeren og sammenligne med hittil beste løsning. Er overestimert dårligere enn hittil beste, tilbakesporer man.

Deloppgave 1c

Hvilken av designteknikkene rå makt, splitt-og-hersk og krymp-og-hersk brukes i denne koden?

```
A = [2, 5, 1, 7, 9, 20, 10, 3]

def finn(A, tall):
    print A
    if len(A) == 0:
        return None
    elif len(A) == 1:
        if A[0] == tall:
            return 0
        else:
            return None
    else:
        midt = len(A) // 2
        res1 = finn(A[:midt], tall)
        res2 = finn(A[midt:], tall)

        if res1 != None:
            return res1
        else:
            if res2 != None:
                return midt + res2
            else:
                return None

i = finn(A, 7)
if i == None:
    print "Fant ikke tallet"
else:
    print "Funnet i posisjon", i
```

Splitt-og-hersk (divide and conquer).

Deloppgave 1d

Enn denne? Her er det ikke et entydig svar.

```
A = [12, 3, 2, 1, 0, 8, 4]
finn = 2
for i in range(len(A)):
    if A[i] == finn:
        print "Funnet i posisjon", i
```

Rå makt (brute force) og krymp-og-hersk (decrease and conquer). Begge er like riktige.

Deloppgave 1e

Og denne?

```
A = [1, 5, 3, 0, 8, 6]
B = []
for X in A:
    if len(B) == 0:
        B = [X]
```

```

else:
    i = 0
    for Y in B:
        if X < Y:
            B.insert(i, X)
            X = None
            break
        i += 1
    if X != None:
        B.append(X)

print B

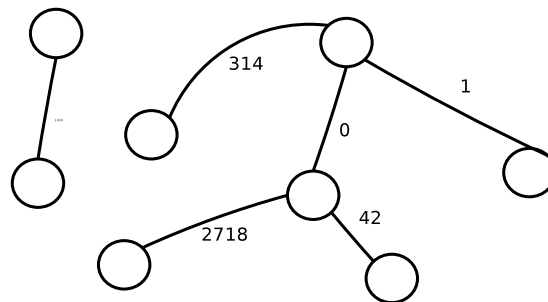
```

Krymp-og-hersk (decrease and conquer).

Oppgave 2 - Datastrukturer

I denne oppgaven skal du finne navnet på forskjellige datastrukturer. En del av strukturene har også en del lett gjenkjennelige egenskaper. Se om du kan finne alle som nevnes i LFet :)

Deloppgave 2a



Dette er en en vektet skog, en type urettet graf. Skogen består av to frie trær (i motsetning til rota trær).

Deloppgave 2b

```

class struct:
    def __init__(self):
        self.values = [None]
        self.i = -1
        self.n = 1

    def add(self, object):
        self.i += 1
        if self.i == self.n:

```

```

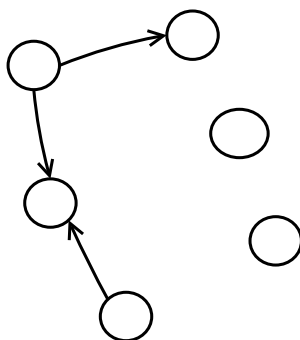
        self.n *= 2
        new_values = [None] * self.n
        for j in range(0, self.i):
            new_values[j] = self.values[j]
        self.values = new_values
        self.values[self.i] = object

def remove(self):
    object = self.values[self.i]
    self.i -= 1
    if self.i < self.n // 4:
        self.n //= 2
        new_values = [None] * self.n
        for j in range(0, self.i + 1):
            new_values[j] = self.values[j]
        self.values = new_values
    return object

```

Dette er en stakk (stack). Dette kalles også en LIFO-kø (last in, first out).

Deloppgave 2c



Dette er en rettet (directed), uvektet (unweighted), ikke sammenhengende (unconnected) graf. Merk at dette ikke er en skog, fordi retningen på kantene gjør at du ikke kan lage noe gyldig tre av de fire nodene som har kanter mellom seg. Grafen er sparse (har få kanter ifht. noder).

Deloppgave 2d

```

size = 23

class struct:
    def __init__(self):
        self.keys = [None] * size
        self.values = [None] * size

    def add(self, key, object):
        i = calc(key)
        if i >= 0:
            if self.keys[i] == None:
                self.keys[i] = [key]
                self.values[i] = [object]
            else:
                for j in range(0, len(self.keys[i])):
                    if self.keys[i][j] == key:

```

```

        self.values[i][j] = object
        break
    else:
        self.keys[i].append(key)
        self.values[i].append(object)

def retrieve(self, key):
    i = calc(key)
    if i >= 0:
        if self.keys[i] == None:
            return None
        else:
            for j in range(0, len(self.keys[i])):
                if self.keys[i][j] == key:
                    return self.values[i][j]

def calc(key):
    if key.__class__ == (1).__class__:
        i = key % size
    elif key.__class__ == "a".__class__:
        i = 0
        c = 1
        for d in key:
            i += ord(d) # ord('a') = 97, chr(ord(d)) = d, chr(97) = 'a'
        i %= size
    else:
        print "Unsupported type"
        return -1
    return i

```

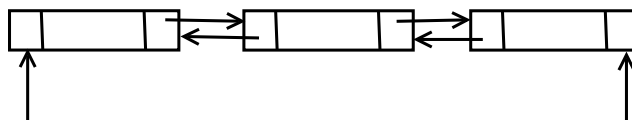
Dette er en hash-tabell (hash table). Merk at dette er en veldig dårlig implementasjon, som neppe vil oppnå konstant tid for retrieve()-funksjonen i gjennomsnitt.

Deloppgave 2e

```
[[0, 1, 0, 1], [1, 1, 1, 0], [0, 1, 1, 0], [0, 0, 0, 1]]
```

Dette er en to-dimensjonal tabell. Siden alle radene er like lange (4), er det en matrise. Siden det er like mange kolonner som rader (4), kan dette godt være nabomatriserepresentasjonen av en graf. Matrisen inneholder bare tallene 0 og 1, så dette kan godt være en uvektet graf.

Deloppgave 2f



Dette er ei dobbelt-lenket liste.

Deloppgave 2g

```
class struct:
```

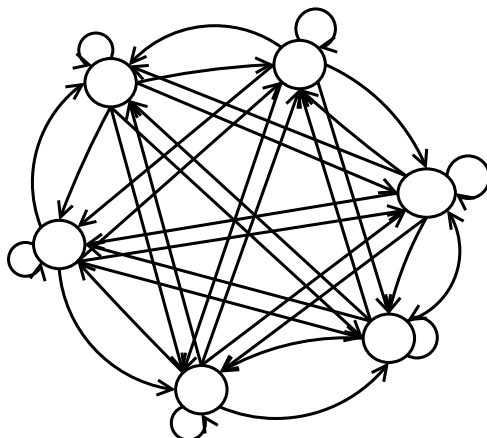
```
def __init__(self):
    self.values = []

def add(self, object):
    self.values.append(object)

def remove(self):
    object = self.values[0]
    self.values = self.values[1:]
    return object
```

Dette er en kø (queue). Dette kalles også en FIFO-kø (first in, first out).

Deloppgave 2h



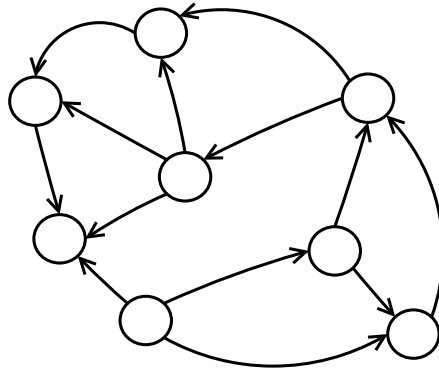
Dette er en graf. Grafen er veldig tett (dense), faktisk komplett (complete). Den inneholder løkker (loops). At den er komplett vil si at det går kanter mellom alle noder. Hvorvidt alle noder må ha løkker (kanter til seg selv) for at en graf skal være komplett, er et definisjonsspørsmål. Vanligvis kalles en graf komplett, selv om den ikke inneholder noen løkker (det antas som regel at det ikke finnes løkker).

Deloppgave 2i

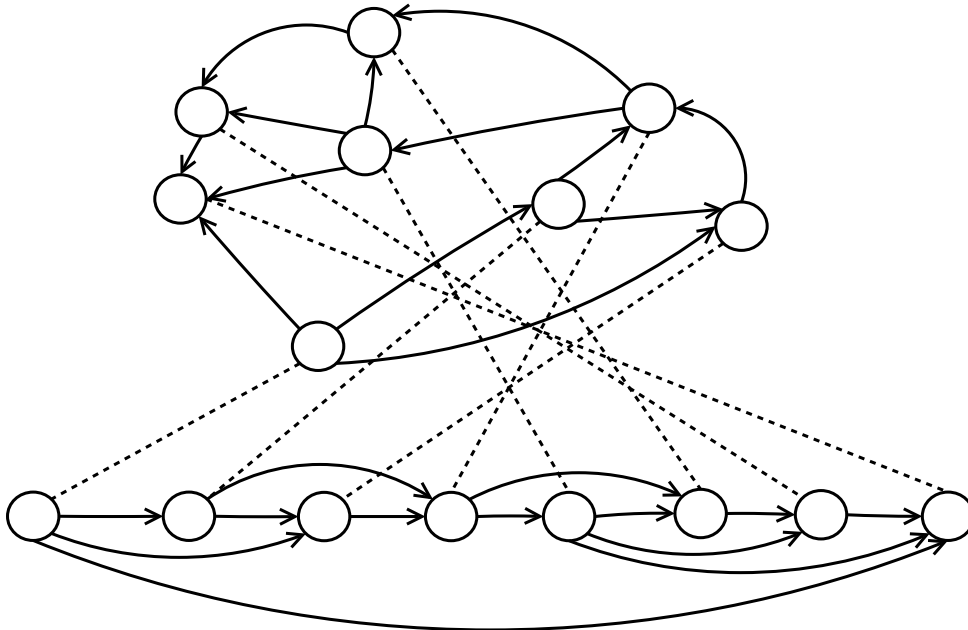
[1, 2, 0, 13, 26]

Dette er en array (tabell). Den inneholder bare ikke-negative heltall.

Deloppgave 2j



Dette er en graf, nærmere bestemt en DAG (directed, acyclic graph). Dette betyr at den er rettet, og er asyklisk (ikke har noen sykler), og ikke inneholder løkker. Denne DAG'en er uvektet. Rette, asykliske grafer kan sorteres topologisk. Dvs. at man legger alle nodene på en linje bortover, slik at alle kantene peker i samme retning. En DAG kan ha flere gyldige topologiske sorteringer, her viser vi én:



Oppgave 3 - Algoritmer

Deloppgave 3a

Fyll ut tabellen. Angi kompleksiteten (Kompl.) for worst case, i asymptotisk notasjon. Bruk den beste worst case-kompleksiteten fra boka, dersom det finnes flere versjoner av algoritmen.

I problem-kolonnen skal du fylle ut hva slags type problemer algoritmen løser. Denne typen er en eller flere av:

- Sortering (sorting)

- Søkning (searching)

- Strengprosessering (string processing)

- Grafproblemer (graph problems)

- Kombinatorisk problem (combinatorial problem)

- Geometrisk problem (geometric problem)

- Numerisk problem (numerical problem)

Algoritme	Designmetode	Kompl.	Problem
Quicksort			
SequentialSearch2			
Binary search			
BruteForceStringMatch	Brute force		
Selection sort			
BFS			
Mergesort			
BruteForceClosestPoints	Brute force		
Bubble sort			
Insertion sort			
DFS			

Vi angir her de generelle kjøretidene i Kompl.-kolonnen. For å få worst-case-kjøretidene er det selvfølgelig (ut i fra definisjonene av O , Θ og worst-case) bare å bytte ut alle forekomster av O med Θ .

Algoritme	Designmetode	Kompl.	Problem
Selection sort	Brute force	$\Theta(n^2)$	Sortering
Bubble sort	Brute force	$O(n^2)^{[*]}$	Sortering
SequentialSearch2	Brute force	$O(n)$	Søking
BruteForceStringMatch	Brute force	$O(n \cdot m)$	Strengprosessering, søking
BruteForceClosestPoints	Brute force	$\Theta(n^2)$	Geometrisk problem, kombinatorisk problem
Mergesort	Divide-and-conquer	$\Theta(n \log n)$	Sortering
Quicksort	Divide-and-conquer	$O(n^2)$	Sortering
Binary search	Divide-and-conquer Decrease-and-conquer (constant factor)	$O(\log n)$	Søking
Insertion sort	Decrease-and-conquer	$O(n^2)$	Sortering
DFS	Decrease-and-conquer (constant)	$\Theta(V ^2)$ eller $\Theta(V + E)^{[**]}$	Grafproblem, søking
BFS	Decrease-and-conquer (constant)	Som DFS	Grafproblem, søking

Brute force og decrease-and-conquer (decrease-by-a-constant) kan ofte begge være naturlige forklaringer på designmetoden for en algoritme, og hvilken som forklarer algoritmen blir litt avhengig av hvilken innfallsvinkel man har.

Det er ikke alltid mulig å kunne kategorisere en algoritme utvetydig etter designmetode (noe dere forhåpentligvis har sett i dette oppgavesettet). Poenget er å ha peiling på hva de forskjellige designmetodene går ut på, slik at dere kan bruke dem til å konstruere algoritmer vha. hovedidéene.

[*] Pseudokoden i boka er $\Theta(n^2)$, men med endringen på side 101 blir den $\Theta(n)$ i best case.

[**] For hhv. nabomatrise-representasjon og lenket liste-representasjon. Merk at hvis vi kan stoppe søket når vi finner en bestemt verdi, er kjøretiden O istdf. Θ

Deloppgave 3b

Hvilken algoritme er dette?

```
def algoritme(G):
    (V, E) = G
    Vt = [V[0]]
    Et = []
    E.sort()
    for i in range(1, len(V)):
        for (w, (v, u)) in E:
            if v in Vt and u not in Vt:
                Vt.append(u)
```

```

        Et.append((w, (v, u)))
        break
    return Et

```

Prims minimum-spenntre-algoritme, omtrent som beskrevet i boka.
Den kan testes med følgende kode:

```

class Node(".__class__"):
    def __init__(self, name):
        self.name = name
    def __str__(self):
        return self.name

a = Node('a')
b = Node('b')
c = Node('c')

V = [a, b, c]
E = [(1, (a, c)), (1, (c, a)), (10,(b,a)), (10,(a,b)),
      (5,(c,b)), (5,(b,c))]

print algoritme((V,E))

```

Deloppgave 3c

Hvilken algoritme er dette?

```

count = 0

def algoritme(G):
    global count
    count = 0
    (V, E) = G
    for v in V:
        v.m = 0
    for v in V:
        if v.m == 0:
            funksjon(V, E, v)

def funksjon(V, E, v):
    global count
    count += 1
    v.m = count
    q = [v]
    while len(q) > 0:
        v = q.pop(0)
        for w in V:
            if w.m == 0 and (v, w) in E:
                count += 1
                w.m = count
                q.append(w)

```

Bredde-først-søk, omtrent som beskrevet i boka.
Den kan testes med følgende kode:

```
class Node("".__class__):
    def __init__(self, name):
        self.name = name
    def __str__(self):
        return self.name

a = Node('a')
b = Node('b')
c = Node('c')
d = Node('d')

V = [a, b, c, d]
E = [(a,b), (b, c), (a,d)]

algoritme((V, E))
for v in V:
    print v, 'ble nummer', v.m
```

Oppgave 4 - Analyse

Deloppgave 4a

Beskriv 1000 forskjellige funksjoner som er med i $\Omega(n)$.

Eksempel 1: $n - i$, der $i \in [0, 1000)$.
Eksempel 2: n^i , der $i \in [1, 1000]$ (i kan ikke være 0 nå).
Eksempel 3: $n \log_i n$, der $i \in [2, 1001]$ ($\log_1 n$ er ikke definert).
Eksempel 4: n^i , der $i \in [1, 500]$ og $n^i \log n$, der $i \in [1, 500]$.

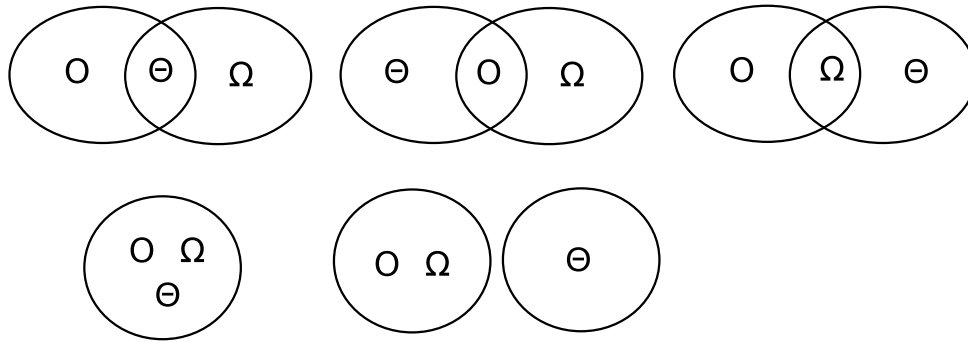
Deloppgave 4b

Beskriv 1000 forskjellige funksjoner som er med i $\Theta(n)$.

Eksempel 1: $n - i$, der $i \in [0, 1000)$.

Deloppgave 4c

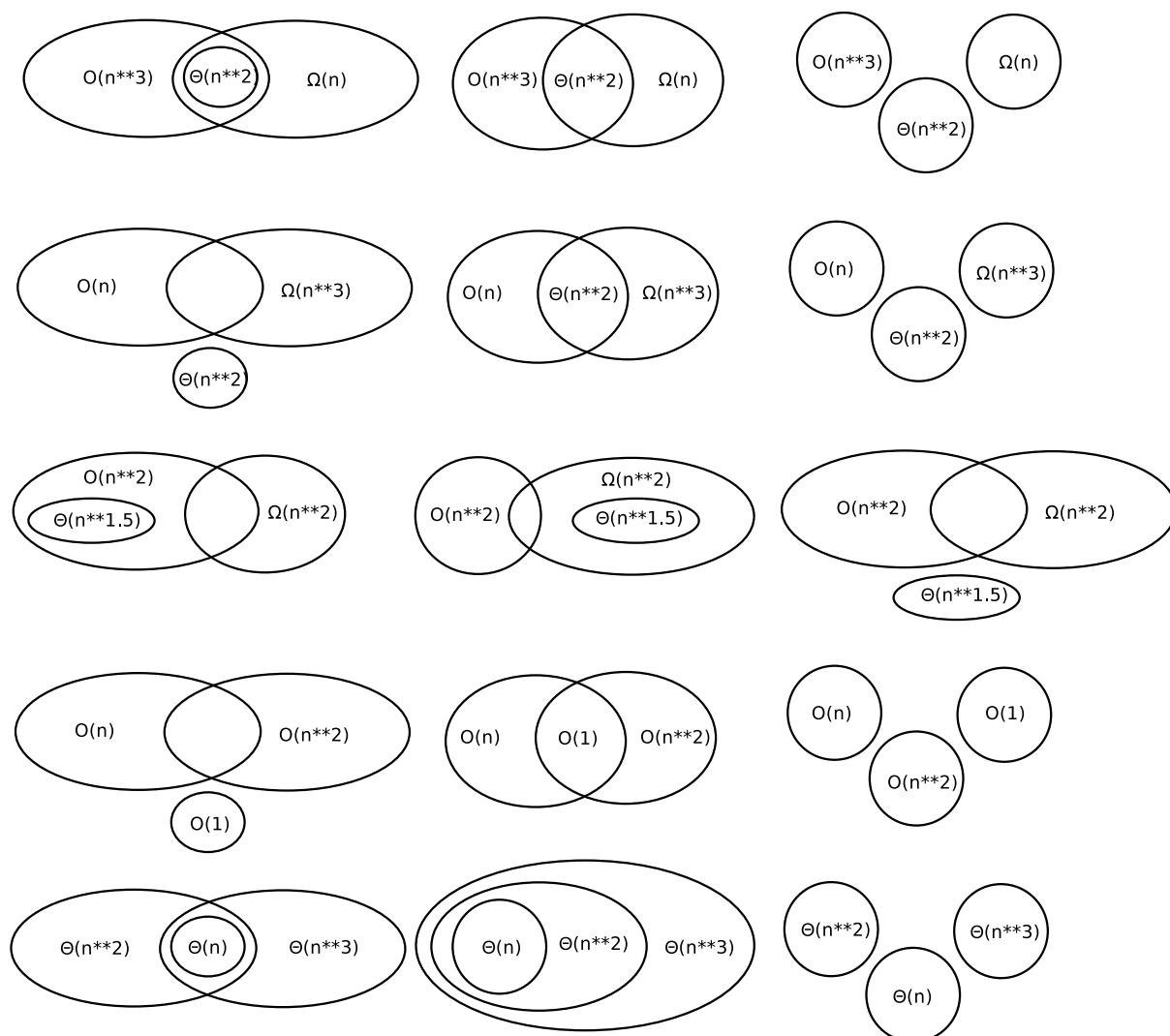
Hvilken figur er riktig?



Den øverste figuren til venstre er riktig. De andre er ikke engang i nærheten.
 En kuriositet:
 $O(n) = \Theta(n) + o(n)$ og $\Omega(n) = \Theta(n) + \omega(n)$.
 $o(n)$ representerer alle funksjoner som er asymptotisk mindre enn (ikke lik) n og $\omega(n)$ tilsvarende alle funksjoner som er asymptotisk større enn n .

Deloppgave 4d

Hvilke figurer er riktige?



Rad 1: Figuren til venstre er riktig.
 $\Theta(n^2) \subset O(n^3), \Theta(n^2) \subset \Omega(n), \Theta(n^2) \subset O(n^3) \cap \Omega(n)$

Rad 2: Figuren til høyre er riktig.
 $O(n) \cap \Theta(n^2) = \emptyset, O(n) \cap \Omega(n^3) = \emptyset, \Theta(n^2) \cap \Omega(n^3) = \emptyset$

Rad 3: Figuren til venstre er riktig.
 $O(n^2) \cap \Omega(n^2) \neq \emptyset, O(n^2) \cap \Omega(n^2) \subset O(n^2), O(n^2) \cap \Omega(n^2) \subset \Omega(n^2),$
 $\Theta(n^{1.5}) \subset O(n^2), \Theta(n^{1.5}) \cap \Omega(n^2) = \emptyset$

Rad 4: Ingen figurer er riktige.
 $O(1) \subset O(n) \subset O(n^2)$

Rad 5: Figuren til høyre er riktig.
 $\Theta(n^a) \cap \Theta(n^b) = \emptyset, a \neq b$

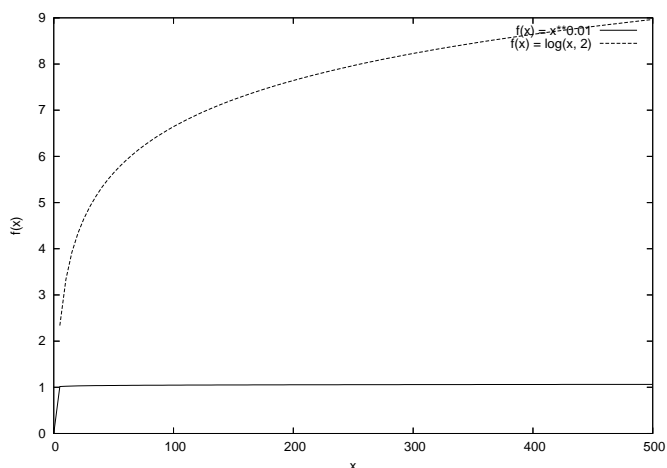
Deloppgave 4e

Noen kjappe ja/nei-spørsmål:

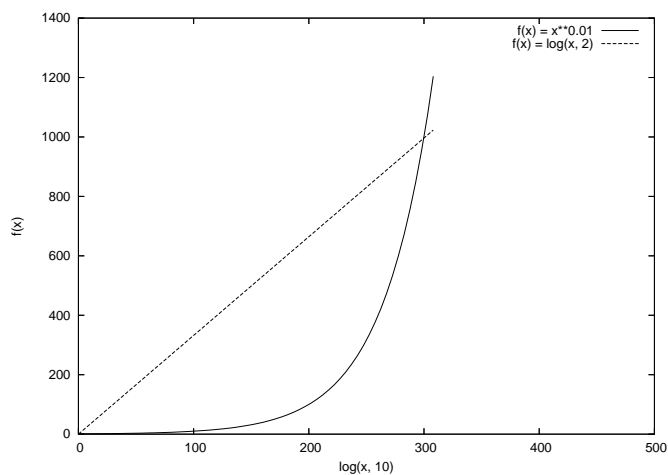
Er $O(n^{1.000000001}) \subseteq O(n)$?

Nei, $n^{1.000000001}$ er asymptotisk større enn n .

Er $O(n^{0.01}) \subseteq O(\log n)$? Merk at nå er n opphøyd i noe som er mindre enn 1. Dette er grafen for små n :



Nei, $\log n$ er asymptotisk mindre; $n^{0.01} > \log n, n > 10^{350}$.



Deloppgave 4f

Gi eksempel på en funksjon som er...

- konstant (constant)
- logaritmisk (logarithmic)

- lineær (linear)
- polynomisk (polynomial)
- n-log-n (n-log-n)
- kvadratisk (quadratic)
- kubisk (cubic)
- eksponensiell (exponential)
- faktoriell (factorial)

konstant	1, 42
logaritmisk	$\log n, \log_{42} n$
lineær	$n, 2n, \frac{n}{2}$
polynomisk	$n, n \log n, n^2, n^{1024}, f(n) : f(n) \in O(n^k), k = \text{konstant}$
n-log-n	$n \log n, 2n \log_{13} n$
kvadratisk	n^2
kubisk	n^3
eksponensiell	$2^n, 3^n$
faktoriell	$n!$

Deloppgave 4g

Beskriv uendelig mange polynomiske funksjoner vha. 4 tegn.

$O(n)$ eller $\Theta(n)$ eller $\Omega(n)$.

Oppgave 5 - Rekurrenser

Dette er en blå kopi av rekurrensoppgavene fra oppgave 2 på øving 4. Eneste unntak er deloppgave c, som er gjort litt vanskeligere. Finn nøyaktige svar av rekurrensene i hver deloppgave ved hjelp av backward substitution og noen av formlene fra formelarket.

Hint: Ikke løs flere oppgaver når backward substitution og de tilhørende matematiske triksene sitter i fingrene :)

Deloppgave 5a

$$T(n) = T(n/2) + 1, T(1) = 0, \text{ der } n = 2^i, i \in \{0, 1, 2, \dots\}$$

$$\begin{aligned} T(n) &= T(n/2) + 1 \\ &= T(n/4) + 2 \\ &= T(n/8) + 3 \\ &= T(n/16) + 4 \end{aligned}$$

Gjetter:

$$\begin{aligned} &= T(n/2^i) + i \\ T(n = 2^i) &= T(2^i/2^i) + i \\ &= T(1) + i \\ &= i \\ &= \log_2 n \end{aligned}$$

Beviser ved induksjon:

$$\text{Antar: } T(n) = \log_2 n$$

Setter inn:

$$\begin{aligned} T(1) &= \log_2 1 = 0, \text{ ok} \\ T(n) &= T(n/2) + 1 \\ &= \log_2 n/2 + 1 \\ &= \log_2 n - \log_2 2 + 1 \\ &= \log_2 n - 1 + 1 \\ &= \log_2 n, \text{ ok} \end{aligned}$$

Deloppgave 5b

$$T(n) = 2T(n/2) + 1, T(1) = 0, \text{ der } n = 2^i, i \in \{0, 1, 2, \dots\}$$

$$\begin{aligned}
T(n) &= 2T(n/2) + 1 \\
&= 2(2T(n/4) + 1) + 1 \\
&= 4T(n/4) + 2 + 1 \\
&= 4(2T(n/8) + 1) + 2 + 1 \\
&= 8T(n/8) + 4 + 2 + 1 \\
&= 8(2T(n/16) + 1) + 4 + 2 + 1 \\
&= 16T(n/16) + 8 + 4 + 2 + 1
\end{aligned}$$

Gjetter:

$$\begin{aligned}
&= 2^i T(n/2^i) + \sum_{j=0}^{i-1} 2^j \\
&= 2^i T(n/2^i) + (1 - 2^i)/(1 - 2) \\
&= 2^i T(n/2^i) + 2^i - 1 \\
T(n = 2^i) &= 2^i T(n/2^i) + 2^i - 1 \\
&= 2^i T(n/n) + n - 1 \\
&= 2^i T(1) + n - 1 \\
&= 0 + n - 1 \\
&= n - 1
\end{aligned}$$

Beviser ved induksjon:

$$\text{Antar: } T(n) = n - 1$$

Setter inn:

$$\begin{aligned}
T(1) &= 1 - 1 = 0, \text{ ok} \\
T(n) &= 2T(n/2) + 1 \\
&= 2(n/2 - 1) + 1 \\
&= n - 2 + 1 \\
&= n - 1, \text{ ok}
\end{aligned}$$

Deloppgave 5c

$$T(n) = 2T(n/3) + 1, T(1) = 1, \text{ der } n = 3^i, i \in \{0, 1, 2, \dots\}$$

$$\begin{aligned}
T(n) &= 2T(n/3) + 1 \\
&= 4T(n/9) + 3 \\
&= 8T(n/27) + 7 \\
&= 16T(n/81) + 15
\end{aligned}$$

gjetter at

$$= 2^i T(n/3^i) + \sum_{j=0}^{i-1} 2^j$$

som er lik

$$\begin{aligned}
&= 2^i T(n/3^i) + 2^i - 1 \\
T(n = 3^i) &= 2^i T(3^i/3^i) + 2^i - 1 \\
&= 2^i T(1) + 2^i - 1 \\
&= 2 \cdot 2^i - 1
\end{aligned}$$

så litt logaritmetriksing

$$\begin{aligned}
3^i &= n \\
i &= \log_3 n \\
2^i &= 2^{\log_3 n} \\
&= n^{\log_3 2}
\end{aligned}$$

som tilsammen gir

$$\begin{aligned}
T(n) &= 2 \cdot n^{\log_3 2} - 1 \\
&\approx 2 \cdot n^{0.63} - 1
\end{aligned}$$

Beviser ved induksjon:

$$\begin{aligned}
T(1) &= 2 \cdot 1^{\log_3 2} - 1 = 1, \text{ ok} \\
T(n) &= 2T(n/3) + 1 \\
2 \cdot n^{\log_3 2} - 1 &= 2\left(2 \cdot \left(\frac{n}{3}\right)^{\log_3 2} - 1\right) + 1 \\
&= 2\left(2 \cdot \frac{n^{\log_3 2}}{3^{\log_3 2}} - 1\right) + 1 \\
&= 2\left(2 \cdot \frac{n^{\log_3 2}}{2} - 1\right) + 1 \\
&= 2 \cdot n^{\log_3 2} - 1, \text{ ok}
\end{aligned}$$

Deloppgave 5d

$$T(n) = 3T(n/3) + 1, T(1) = 0, \text{ der } n = 3^i, i \in \{0, 1, 2, \dots\}$$

$$\begin{aligned}
T(n) &= 3T(n/3) + 1 \\
&= 3(3T(n/9) + 1) + 1 \\
&= 9T(n/9) + 3 + 1 \\
&= 9(3T(n/27) + 1) + 3 + 1 \\
&= 27T(n/27) + 9 + 3 + 1 \\
&= 27(3T(n/81) + 1) + 9 + 3 + 1 \\
&= 81T(n/81) + 27 + 9 + 3 + 1
\end{aligned}$$

Gjetter:

$$\begin{aligned}
&= 3^i T(n/3^i) + \sum_{j=0}^{i-1} 3^j \\
&= 3^i T(n/3^i) + (1 - 3^i)/(1 - 3) \\
&= 3^i T(n/3^i) + 3^i/2 - 1/2 \\
T(n = 3^i) &= 3^i T(n/3^i) + 3^i/2 - 1/2 \\
&= 3^i T(n/n) + n/2 - 1/2 \\
&= 3^i T(1) + n/2 - 1/2 \\
&= 0 + n/2 - 1/2 \\
&= n/2 - 1/2 \\
&= (n - 1)/2
\end{aligned}$$

Beviser ved induksjon:

$$\text{Antar: } T(n) = (n - 1)/2$$

Setter inn:

$$\begin{aligned}
T(1) &= (1 - 1)/2 = 0, \text{ ok} \\
T(n) &= 3T(n/3) + 1 \\
&= 3((n/3 - 1)/2) + 1 \\
&= n/2 - 3/2 + 1 \\
&= n/2 - 1/2 \\
&= (n - 1)/2, \text{ ok}
\end{aligned}$$

Deloppgave 5e

$$T(n) = T(n/2) + n, T(1) = 0, \text{ der } n = 2^i, i \in \{0, 1, 2, \dots\}$$

$$\begin{aligned}
T(n) &= T(n/2) + n \\
&= T(n/4) + n/2 + n \\
&= T(n/8) + n/4 + n/2 + n \\
&= T(n/16) + n/8 + n/4 + n/2 + n
\end{aligned}$$

Gjetter:

$$\begin{aligned}
&= T(n/2^i) + \sum_{j=0}^{i-1} n/2^j \\
&= T(n/2^i) + n \cdot \sum_{j=0}^{i-1} (1/2)^j \\
&= T(n/2^i) + n \cdot (1 - (1/2)^i)/(1 - 1/2) \\
&= T(n/2^i) + n \cdot (1 - 1/2^i)/(1/2) \\
&= T(n/2^i) + 2n - 2n \cdot 1/2^i \\
&= T(n/2^i) + 2n - 2n/2^i \\
T(n = 2^i) &= T(n/2^i) + 2n - 2n/2^i \\
&= T(n/n) + 2n - 2n/n \\
&= T(1) + 2n - 2 \\
&= 0 + 2n - 2 \\
&= 2n - 2
\end{aligned}$$

Beviser ved induksjon:

$$\text{Antar: } T(n) = 2n - 2$$

Setter inn:

$$\begin{aligned}
T(1) &= 2 \cdot 1 - 2 = 0, \text{ ok} \\
T(n) &= T(n/2) + n \\
&= 2(n/2) - 2 + n \\
&= n - 2 + n \\
&= 2n - 2, \text{ ok}
\end{aligned}$$

Deloppgave 5f

$$T(n) = 2T(n/2) + n, T(1) = 0, \text{ der } n = 2^i, i \in \{0, 1, 2, \dots\}$$

$$\begin{aligned}
T(n) &= 2T(n/2) + n \\
&= 2(2T(n/4) + n/2) + n \\
&= 4T(n/4) + 2n \\
&= 4(2T(n/8) + n/4) + 2n \\
&= 8T(n/8) + 3n \\
&= 8(2T(n/16) + n/8) + 3n \\
&= 16T(n/16) + 4n
\end{aligned}$$

Gjetter:

$$\begin{aligned}
&= 2^i T(n/2^i) + i \cdot n \\
T(n = 2^i) &= 2^i T(n/2^i) + i \cdot n \\
&= 2^i T(n/n) + i \cdot n \\
&= 2^i T(1) + i \cdot n \\
&= 0 + i \cdot n \\
&= 0 + \log n \cdot n \\
&= n \log n
\end{aligned}$$

Beviser ved induksjon:

$$\text{Antar: } T(n) = n \log n$$

Setter inn:

$$\begin{aligned}
T(1) &= 1 \cdot \log 1 = 0, \text{ ok} \\
T(n) &= 2T(n/2) + n \\
&= 2(n/2 \log n/2) + n \\
&= n \log n/2 + n \\
&= n \log n - n \log 2 + n \\
&= n \log n - n + n \\
&= n \log n, \text{ ok}
\end{aligned}$$

Oppgave 6 - Master-teoremet

Her kommer en del av de samme rekurensene en gang til. Nå skal oppgavene løses vha. *Master-teoremet*. Dersom Master-teoremets krav ikke er oppfylt, skal du svare at rekurrensen ikke kan løses vha. Master-teoremet, og løse rekurrensen ved hjelp av backward substitution.

Delppgavene som ikke lar seg løse med masterteoremet, krever ganske høy konsentrasjon for å løses med backward substitution. Det er godt mulig du kan bruke tiden på mer fornuftige ting enn å finne den nøyaktige løsningen på disse.

Deloppgave 6a

$$T(n) = T(n/2) + 1, T(1) = 0, \text{ der } n = 2^i, i \in \{0, 1, 2, \dots\}$$

Her er $a = 1$, $b = 2$ og $d = 0$, noe som gir $a = b^d$. Dette er andre tilfellet i Master-teoremet som gir kjøretiden $\Theta(n^0 \log n) = \Theta(\log n)$.

Deloppgave 6b

$$T(n) = 2T(n/2) + 1, T(1) = 0, \text{ der } n = 2^i, i \in \{0, 1, 2, \dots\}$$

Her er $a = 2$, $b = 2$ og $d = 0$, noe som gir $a > b^d$. Dette er tredje tilfellet i Master-teoremet som gir kjøretiden $\Theta(n^{\log_2 2}) = \Theta(n)$.

Deloppgave 6c

$$T(n) = 2T(n/3) + 1, T(1) = 1, \text{ der } n = 3^i, i \in \{0, 1, 2, \dots\}$$

Her er $a = 2$, $b = 3$ og $d = 0$, noe som gir $a > b^d$. Dette er tredje tilfellet i Master-teoremet som gir kjøretiden $\Theta(n^{\log_3 2}) \approx \Theta(n^{0.63})$.

Deloppgave 6d

$$T(n) = 2T(n/2) + n \log n, T(1) = 0, \text{ der } n = 2^i, i \in \{0, 1, 2, \dots\}$$

$n \log n \notin \Theta(n^d)$, så Master-teoremet kan ikke benyttes. Hvis man er villig til å løse på grensene, kan man skrive om til

$T(n) = 2T(n/2) + O(n^2)$, $T(1) = 0$, som gir $T(n) = O(n^2)$ med Master-teoremet.

Dette er som regel ikke ønskelig, da vi kan finne en mer nøyaktig grense vha. backward substitution, nemlig $T(n) = \Theta(n \log n \log n)$, $n \log n \log n = n \cdot (\log(n)) \cdot (\log(n))$:

$$\begin{aligned}
 T(n) &= \mathbf{2T(n/2)} + \mathbf{n \log n} \\
 &= 2(2T(n/4) + n/2 \log n/2) + n \log n \\
 &= 4T(n/4) + n \log n/2 + n \log n \\
 &= \mathbf{4T(n/4)} + \mathbf{2n \log n} - \mathbf{n} \\
 &= 4(2T(n/8) + n/4 \log n/4) + 2n \log n - n \\
 &= 8T(n/8) + n \log n/4 + 2n \log n - n \\
 &= \mathbf{8T(n/8)} + \mathbf{3n \log n} - \mathbf{n(1 + 2)} \\
 &= 8(2T(n/16) + n/8 \log n/8) + 3n \log n - n(1 + 2) \\
 &= 16T(n/16) + n \log n/8 + 3n \log n - n(1 + 2) \\
 &= \mathbf{16T(n/16)} + \mathbf{4n \log n} - \mathbf{n(1 + 2 + 3)}
 \end{aligned}$$

Gjetter:

$$\begin{aligned}
 &= 2^i T(n/2^i) + i \cdot n \log n - n \sum_{j=0}^{i-1} j \\
 &= 2^i T(n/2^i) + i \cdot n \log n - n \sum_{j=0}^{i-1} j \\
 &= 2^i T(n/2^i) + i \cdot n \log n - n(i(i+1)/2 - i) \\
 &= 2^i T(n/2^i) + i \cdot n \log n - n(i^2/2 - i/2) \\
 &= 2^i T(n/2^i) + i \cdot n \log n - n \cdot i^2/2 + n \cdot i/2 \\
 T(n = 2^i) &= nT(n/n) + i \cdot n \log n - n \cdot i^2/2 + n \cdot i/2 \\
 &= nT(1) + \log n \cdot n \log n - (n \log n \log n)/2 + (n \log n)/2 \\
 &= n \log n \log n - (n \log n \log n)/2 + (n \log n)/2 \\
 &= (n \log n \log n)/2 + (n \log n)/2 \\
 &= (n \log n \log n + n \log n)/2
 \end{aligned}$$

Beviser ved induksjon:

$$\text{Antar: } T(n) = (n \log n \log n + n \log n)/2$$

Setter inn:

$$T(1) = (1 \cdot \log 1 \log 1 + 1 \cdot \log 1)/2 = 0, \text{ ok}$$

$$\begin{aligned} T(n) &= 2T(n/2) + n \log n \\ &= 2((n/2 \log n/2 \log n/2 + n/2 \log n/2)/2) + n \log n \\ &= n/2 \log n/2 \log n/2 + n/2 \log n/2 + n \log n \\ &= n/2(\log n - \log 2)(\log n - \log 2) + n/2(\log n - \log 2) + n \log n \\ &= n/2(\log n \log n - \log n - \log n + 1) + n/2(\log n - 1) + n \log n \\ &= (n \log n \log n)/2 - n \log n + n/2 + (n \log n)/2 - n/2 + n \log n \\ &= (n \log n \log n)/2 - n \log n + n \log n + n/2 - n/2 + (n \log n)/2 \\ &= (n \log n \log n)/2 + (n \log n)/2 \\ &= (n \log n \log n + n \log n)/2, \text{ ok} \end{aligned}$$

Deloppgave 6e

$$T(n) = 3T(n/3) + 1, T(1) = 0, \text{ der } n = 3^i, i \in \{0, 1, 2, \dots\}$$

Her er $a = 3$, $b = 3$ og $d = 0$, noe som gir $a > b^d$. Dette er tredje tilfellet i Master-teoremet som gir kjøretiden $\Theta(n^{\log_3 3}) = \Theta(n)$.

Deloppgave 6f

$$T(n) = T(n/2) + n, T(1) = 0, \text{ der } n = 2^i, i \in \{0, 1, 2, \dots\}$$

Her er $a = 1$, $b = 2$ og $d = 1$, noe som gir $a > b^d$. Dette er første tilfellet i Master-teoremet som gir kjøretiden $\Theta(n^1) = \Theta(n)$.

Deloppgave 6g

$$T(n) = 2T(n/2) + n, T(1) = 0, \text{ der } n = 2^i, i \in \{0, 1, 2, \dots\}$$

Her er $a = 2$, $b = 2$ og $d = 1$, noe som gir $a = b^d$. Dette er andre tilfellet i Master-teoremet som gir kjøretiden $\Theta(n^1 \log n) = \Theta(n \log n)$.

Deloppgave 6h

$$T(n) = 2T(n-2) + n, T(0) = 0, \text{ der } n \in \{0, 1, 2, \dots\}$$

Rekurrensen kan *ikke* omskrives til formen $T(n) = aT(n/b) + \Theta(n^d)$, og Master-teoremet kan derfor ikke benyttes. Her må en benytte substitusjon. Okey, man kunne kanskje benyttet teoremene om linear second-order recurrences with constant coefficients i appendiks B, men disse ligger utenfor pensum (og det er ikke trivielt å løse side rekurrensen er inhomogen med en ikke-konstant funksjon $f(n)$). Løser ved hjelp av backward substitution:

$$\begin{aligned} T(n) &= = \mathbf{2T(n-2) + n} \\ &= 2(2T(n-4) + n-2) + n \\ &= \mathbf{4T(n-4) + n(1+2) - 4} \\ &= 4(2T(n-6) + n-4) + n(1+2) - 4 \\ &= \mathbf{8T(n-6) + n(1+2+4) - (4+16)} \\ &= 8(2T(n-8) + n-6) + n(1+2+4) - (4+16) \\ &= \mathbf{16T(n-8) + n(1+2+4+8) - (4+16+48)} \\ &= 16(2T(n-10) + n-8) + n(1+2+4+8) - (4+16+48) \\ &= \mathbf{32T(n-10) + n(1+2+4+8+16) -} \\ &\quad \mathbf{(4+16+48+128)} \end{aligned}$$

Gjetter:

$$\begin{aligned}
&= 2^i T(n - 2i) + n \sum_{j=1}^i 2^{j-1} - \sum_{j=2}^i 2^{j-1} 2(j-1) \\
&= 2^i T(n - 2i) + n \sum_{j=0}^{i-1} 2^j - \sum_{j=1}^i 2^{j-1} 2(j-1) \\
&= 2^i T(n - 2i) + n(2^i - 1) - 2 \sum_{j=1}^i 2^{j-1} (j-1) \\
&= 2^i T(n - 2i) + n(2^i - 1) - 2 \sum_{j=1}^i 2^{j-1} j + 2 \sum_{j=1}^i 2^{j-1} \\
&= 2^i T(n - 2i) + n(2^i - 1) - 2 \sum_{j=1}^i 2^{j-1} j + 2 \sum_{j=0}^{i-1} 2^j \\
&= 2^i T(n - 2i) + n(2^i - 1) - 2 \sum_{j=1}^i 2^{j-1} j + 2 \sum_{j=0}^{i-1} 2^j \\
&= 2^i T(n - 2i) + (n + 2)(2^i - 1) - 2 \sum_{j=1}^i 2^j / 2 \cdot j \\
&= 2^i T(n - 2i) + (n + 2)(2^i - 1) - \sum_{j=1}^i 2^j \cdot j \\
&= 2^i T(n - 2i) + (n + 2)(2^i - 1) - ((i - 1)2^{i+1} + 2) \\
&= 2^i T(n - 2i) + n2^i - n + 2 \cdot 2^i - 2 - 2i2^i + 2 \cdot 2^i - 2 \\
&= 2^i T(n - 2i) + 2^i(n + 2 - 2i + 2) - n - 2 - 2 \\
&= 2^i T(n - 2i) + 2^i(n - 2i + 4) - n - 4 \\
T(n = 2i) &= 2^i T(n - n) + 2^{n/2}(n - 2(n/2) + 4) - n - 4 \\
&= 2^i T(0) + 4 \cdot 2^{n/2} - n - 4 \\
&= 4 \cdot 2^{n/2} - n - 4
\end{aligned}$$

Beviser ved induksjon:

$$\text{Antar: } T(n) = 4 \cdot 2^{n/2} - n - 4$$

Setter inn:

$$\begin{aligned}
T(0) &= 4 \cdot 2^{0/2} - 0 - 4 = 4 - 4 = 0, \text{ ok} \\
T(n) &= 2T(n - 2) + n \\
&= 2(4 \cdot 2^{(n-2)/2} - (n - 2) - 4) + n \\
&= 8 \cdot 2^{n/2-1} - 2n + 4 - 8 + n \\
&= 8 \cdot 2^{n/2}/2 - n - 4 \\
&= 4 \cdot 2^{n/2} - n - 4, \text{ ok}
\end{aligned}$$

Oppgave 7 - Kjøretid - Finn formel

I denne oppgaven skal vi se på den asymptotiske oppførselen til noen enkle funksjoner. Du skal gi rekurrensen (eller en degenerert rekurrens) for funksjonen `funksjonX` av n med hensyn på basisoperasjonen, `gjorKonst()` eller `gjorNoe()`. Gi et så presist svar som mulig, men ikke løs ut evt. rekurrenser.

I neste oppgave skal du løse evt. rekurrenser, så da får du se om du har gjort riktig på noen av oppgavene her. Ikke tjuvtitt :-)

Anta at funksjonen `gjorKonst()` har kjøretid $\Theta(1)$ og at funksjonen `gjorNoe()` har kjøretid $O(n)$.

For å gjøre ting pent, anta at $n = 2^i, i \in 1, 2, 3, \dots$

Deloppgave 7a

```
def funksjonA(n):
    for i in range(0, n):
        for j in range(0, n):
            gjorKonst()
```

$$T(n) = n^2 \cdot \Theta(1) = \Theta(n^2)$$

Deloppgave 7b

```
def funksjonB(n):
    sum = 0
    if n > 1:
        sum += funksjonB(n//2)
        sum += funksjonB(n//2)
        for i in range(n):
            sum += i
            gjorKonst()

    return sum
```

$$\begin{aligned} T(n) &= 2T\left(\frac{n}{2}\right) + n \cdot \Theta(1) \\ &= 2T\left(\frac{n}{2}\right) + \Theta(n) \\ T(1) &= 0 \end{aligned}$$

Deloppgave 7c

```
from random import randint
```

```

def funksjonC(n):
    tall = [0] * n
    for i in range(0, n):
        tall[i] = randint(1, n)
        gjorKonst()

    for i in range(0, n):
        for j in range(0, n):
            maks = 0
            for k in tall:
                tmp = abs(tall[j] - k)
                if tmp > maks:
                    maks = tmp
            gjorKonst()
            tall[j] = maks

```

$$\begin{aligned}
 T(n) &= n \cdot \Theta(1) + n^3 \cdot \Theta(1) \\
 &= \Theta(n) \cup \Theta(n^3) \\
 &= \Theta(n^3)
 \end{aligned}$$

Merk at $\Theta(n) \cup \Theta(n^3) \neq \Theta(n^3)$, siden $\Theta(n) \not\subseteq \Theta(n^3)$. Faktisk er $\Theta(n) \cap \Theta(n^3) = \emptyset$. Men hva asymptotisk kjøretid angår, vil $\Theta(n^3)$ dominere $\Theta(n)$, så vi skriver

“ $\Theta(n) \cup \Theta(n^3) = \Theta(n^3)$ ”

som en kortform for

“ $T(n) = (\dots) = \Theta(n) \cup \Theta(n^3)$. Siden vi ser på den asymptotiske kjøretiden, vil leddet $\Theta(n^3)$ dominere $\Theta(n)$ og vi får $T(n) = \Theta(n^3)$.”

Deloppgave 7d

```

def funksjonD(n):
    if n > 0:
        funksjonD(n-1)
    gjorNoe()

```

$$\begin{aligned}
 T(n) &= T(n-1) + O(n) \\
 T(1) &= O(n)
 \end{aligned}$$

Deloppgave 7e

Hint: Bare sett opp rekurensen for worst case-kjøretid. Hvis denne kan løses og gi et svar med Θ -notasjon, er det bare å endre dette svaret til å bruke O -notasjon for å få en generell kjøretid.

```

from random import randint

def funksjonE(n):
    tall = randint(0, n - 1)
    print finn(tall, 0, n - 1)

def finn(tall, nedre, ovre):
    gjorKonst()
    gjett = (ovre + nedre) // 2
    if gjett == tall:

```

```

    return gjett
elif gjett > tall:
    return finn(tall, nedre, gjett - 1)
else:
    return finn(tall, gjett + 1, ovre)

```

Siden vi her skal finne et tilfeldig tall, vet vi ikke hva kjøretiden vil bli. Vi kan derimot regne ut kjøretiden, $f(n)$, for verste tilfelle (worst case), og angi den asymptotiske kjøretiden for algoritmen som $T(n) = O(f(n))$.

Hvis n er antall tall vi skal søke gjennom, er $n = u - l + 1$, $u = \text{ovre}$, $l = \text{nedre}$ i ett kall til `finn`. +1 fordi l og u er inkluderende grenser. I det neste rekursive kallet til `finn` vil vi ha

$$n_{rekursiv} = \begin{cases} 0 & \text{hvis } g = t \\ (g - 1) - (l) + 1 = g - l = \lfloor \frac{u+l}{2} \rfloor - l & \text{hvis } g < t \\ \stackrel{\text{worst case}}{=} \frac{u+l}{2} - l = \frac{u-l}{2} = \frac{n}{2} - \frac{1}{2} & \\ (u) - (g + 1) + 1 = u - g = u - \lfloor \frac{u+l}{2} \rfloor & \text{hvis } g > t \\ \stackrel{\text{worst case}}{=} u - (\frac{u+l-1}{2}) = \frac{u-l+1}{2} = \frac{n}{2} & \end{cases}$$

der $g = \text{gjett}$, $t = \text{tall}$ og alle verdiene er de gamle verdiene (unntatt $n_{rekursiv}$, selvsagt) Altså

$$\begin{aligned}
T_{\text{worst case}}(n) &= T(\max(n_{rekursiv})) \\
&= T(n/2) + \Theta(1) \\
T_{\text{worst case}}(1) &= \Theta(1)
\end{aligned}$$

At programmet vil kjøre til $n = 1$ i worst case ser vi når vi ser at det verste vil være om programmet kjører til `ovre = nedre`. Da er $n = u - l + 1 = u - u + 1 = 1$.

Vi kan nå gi den generelle rekurrensen for kompleksiteten til programmet:

$$\begin{aligned}
T(n) &\leq T(n/2) + \Theta(1) \\
T(1) &= O(1)
\end{aligned}$$

Oppgave 8 - Kjøretid - Løs rekurrens

I denne oppgaven skal vi løse rekurrensene fra hhv. deloppgaver b, d og e i forrige oppgave. Bruk den metoden du liker best :)

Deloppgave 8a

$$T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n), T(1) = 0$$

Som rekurrens-oppgave f, men svaret blir $\Theta(n \log n)$ istdf. $n \log n$.

Deloppgave 8b

$$T(n) = T(n-1) + O(n), T(1) = O(n)$$

Bruker backward substitution:

$$\begin{aligned} T(n) &= T(n-1) + O(1) \\ &= T(n-2) + O(1) + O(1) \\ &= T(n-3) + O(1) + O(1) + O(1) \\ &= T(n-4) + 4 \cdot O(1) \\ &= T(n-5) + 5 \cdot O(1) \end{aligned}$$

Gjetter:

$$\begin{aligned} &= T(n-i) + i \cdot O(1) \\ T(n=i-1) &= T(n-n+1) + (n-1) \cdot O(1) \\ &= T(1) + (n-1) \cdot O(1) \\ &= (n-1) \cdot O(1) \end{aligned}$$

Beviser ved induksjon:

$$\text{Antar: } T(n) = (n-1) \cdot O(1)$$

Setter inn:

$$\begin{aligned} T(1) &= (1-1) \cdot O(1) = 0 \cdot O(1) = 0, \text{ ok} \\ T(n) &= T(n-1) + O(1) \\ &= ((n-1)-1) \cdot O(1) + O(1) \\ &= (n-2+1) \cdot O(1) \\ &= (n-1) \cdot O(1) \end{aligned}$$

Deloppgave 8c

$$T(n) \leq T(n/2) + \Theta(1), T(1) = O(1)$$

(endret i forhold til utleverte oppgaver).

Som rekurrens-oppgave a, men svaret blir $O(\log_2 n)$ istdf. $\log_2 n$.
Dette regnes ut ved å løse $T(n) = T(n/2) + \Theta(1)$ istdf. $T(n) \leq T(n/2) + \Theta(1)$ og deretter bruke O -notasjon, som jo angir øvre grense, slik som \leq .
Eventuelt kan man jo svare $T(n) \leq \log_2 n$.

Oppgave 9 - Ja/Nei

Er den asymptotiske kjøretiden til en algoritme avhengig av datastrukturen som brukes?

Ja. F.eks. er Prims algoritme $O(V^2)$ med nabomatriser, mens den er $O(E \log V)$ med nabolister og heap.

Brukes fysiske datastrukturer til å implementere abstrakte datatyper?

Ja. F.eks. kan en kø (abstrakt) implementeres vha. en array (fysisk). Abstrakte datatyper beskriver et grensesnitt (interface).

Kan en graf inneholde flere trær?

Ja, da kalles den en skog (forest).

Kan et binærtre være ei lenket liste?

Ja, et fullstendig ubalansert binærtre degenerer til ei lenka liste.

Kan en graf ha flere kanter enn noder?

Ja, selvfølgelig.

Kan et tre ha flere kanter enn noder?

Nei, da må det være sykler, og det er ikke lenger et tre, men en mer generell graf. Dette svaret forutsetter at vi definerer at trær har urettede kanter.

Er $n^2 \in O(n)$?

Nei.

Er $n^2 \in \Omega(n)$?

Ja.

Er $n^2 \in \Theta(n)$?

Nei.

Er $3^n \in O(2^n)$?

Nei, 3^n vokser mye fortere enn 2^n .

Er $n! \in O(2^n)$?

Nei, $n!$ er jo $n \cdot (n - 1) \cdot (n - 2) \cdot (n - 3) \cdot \dots \cdot 1$, mens 2^n er bare $2 \cdot 2 \cdot 2 \cdot 2 \cdot \dots \cdot 2$ n ganger.

Formler

Her følger noen formler dere kan få bruk for på denne kondisjonstesten og ellers i livet. Du kan anse dette som en sketch på hva som forventes på eksamen, men vi ønsker *ikke* å binde oss ved f.eks. å love at den anbefalte kunnskapen vil bli oppgitt ved behov.

Forventet kunnskap

NB! Denne listen er ikke nødvendigvis¹ komplett.

$$\log x^y = y \log x$$

$$\log xy = \log x + \log y$$

$$\log \frac{x}{y} = \log x - \log y$$

$$\log_b x = \frac{\log_a x}{\log_a b} \text{ (eks. } \log_2 x = \frac{\log_e x}{\log_e 2} \text{)}$$

$$a^{\log_a x} = x^{\log_a a} \text{ (eks. } 2^{\log_2 n} = n \text{)}$$

$$\frac{a^n}{b^n} = \left(\frac{a}{b}\right)^n \text{ (eks. } \frac{1}{2^n} = \left(\frac{1}{2}\right)^n \text{)}$$

$$\sqrt{a \cdot b} = \sqrt{a} \cdot \sqrt{b} \text{ (eks. } x \cdot \sqrt{y} = \sqrt{x^2 \cdot y} \text{)}$$

$$\begin{aligned} \sum_{i=1}^n i \cdot x &= x \cdot \sum_{i=1}^n i \\ \sum_{i=1}^n (a_i + b_i) &= \sum_{i=1}^n a_i + \sum_{i=1}^n b_i \\ \sum_{i=1}^n i &= \sum_{i=1}^k i + \sum_{i=k+1}^n i, \text{ der } i \leq k < n \\ \text{(eks. } \sum_{i=1}^n (n-i) &= \sum_{i=0}^n (n-i) + n \text{)} \\ \sum_{i=k}^{l-1} c &= (l-k) \cdot c \end{aligned}$$

$$\begin{aligned} \sum_{i=1}^n i &= \frac{n(n+1)}{2} \approx \frac{1}{2}n^2 \\ \sum_{i=0}^n 2^i &= 2^{n+1} - 1 \end{aligned}$$

Master-teoremet:

Dersom

$$T(n) = aT(n/b) + \Theta(n^d), \quad n = b^k, \quad k = 1, 2, \dots,$$

$$T(1) = c,$$

der $a \geq 1, b \geq 2, c > 0, d \geq 0$ så

$$T(n) \in \begin{cases} \Theta(n^d) & \text{hvis } a < b^d \\ \Theta(n^d \log n) & \text{hvis } a = b^d \\ \Theta(n^{\log_b a}) & \text{hvis } a > b^d \end{cases}$$

Anbefalt kunnskap

$$a^{\log_b x} = x^{\log_b a}$$

$$\sum_{i=1}^n i^2 = \frac{n(n+1)(2n+1)}{6} \approx \frac{1}{3}n^3$$

$$\sum_{i=0}^n a^i = \frac{a^{n+1}-1}{a-1}, \text{ der } a \neq 1$$

Kursorisk:

$$\sum_{i=1}^n i2^i = (n-1)2^{n+1} + 2$$

¹Bonuskunnskap: "ikke nødvendigvis" \neq "nødvendigvis ikke" (spør Trygve om en eventuell utdyping av dette).