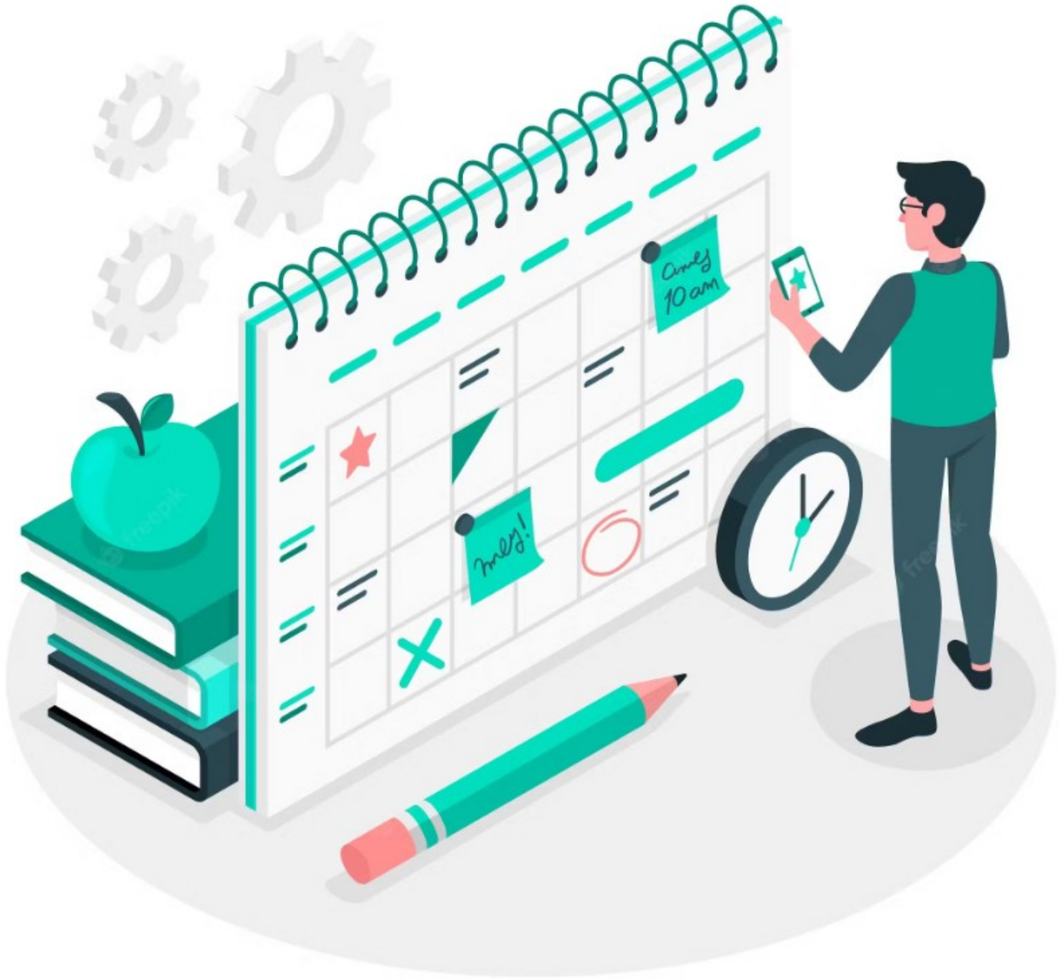
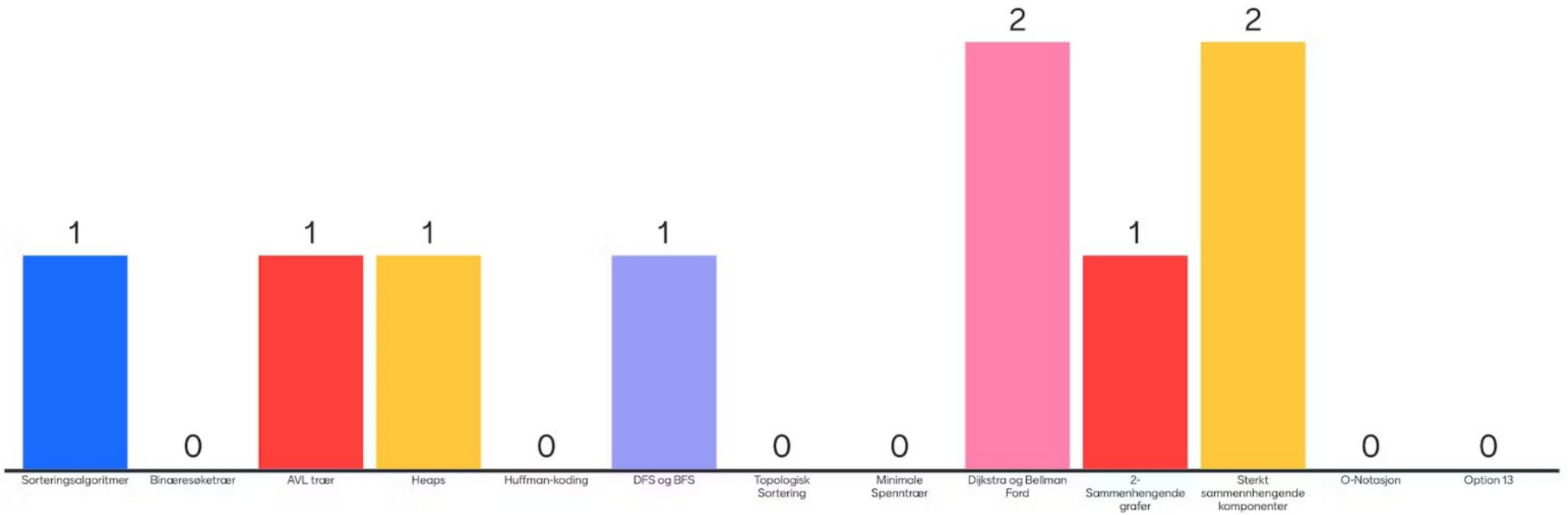
IN2010 - Gruppe 4

Uke 8: Repetisjon



Dagens Plan

- → Repetisjon
- → Repetisjonsoppgaver

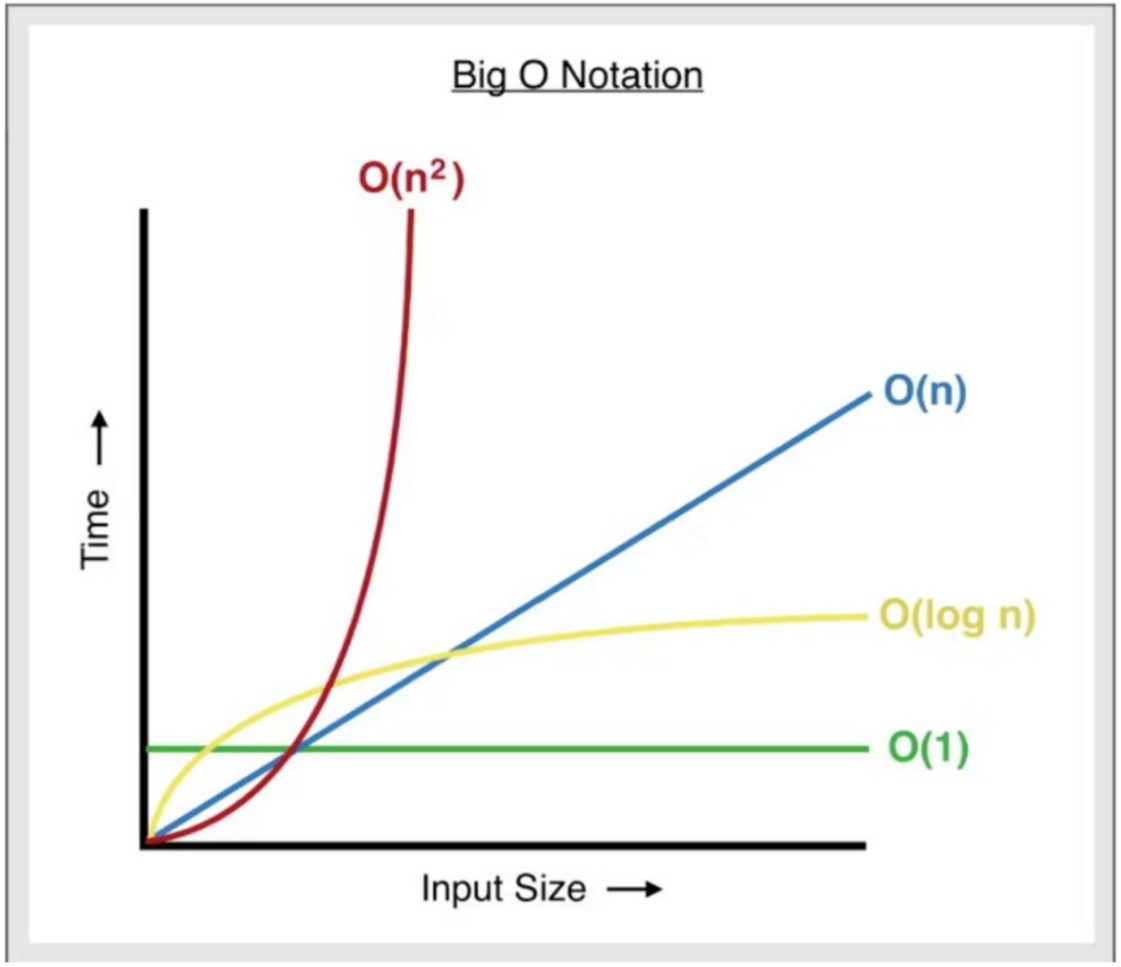


Resultat fra Skjema

Top Tema

- O Notasjon
- DFS og BFS
- Pseudokode

O Notasjon



O-Notasjon

- Hvor vanskelig er et problem/algoritme å løse?
- AKA: Hvor mange steg trenger man for å løse problemet?
- Eksempel: Hvor mange steg må man gjøre for å finne en vilkårlig person i klassen?(Rett-Frem søk)

Hvordan å finne kjøretidskompleksitet?

- Identifiser iterative steg(Løkker og Rekursive kall)
- Finne avhengigheter(Er det prosesser som skjer i løkker og rekursive kall)
- Blir andre funkjsoner kalt på? Hva er kjøretiden til de

Algorithm 1: Prims algoritme

```
1  Procedure Prim(G)
2      initialize T as empty tree, Q as empty heap
3      for each vertex u in G do
4          D[u] = ∞
           // Q contains (Node, Edge), where Edge has lowest weight
            of edges from T to Node, and uses D[u] to compare
5          Q.add(((u, None), D[u]))
6      pick v ∈ V, set D[v] = 0
7      while Q not empty do
8          (s, e) = Q.removeMin()
9          add s and e to T
           // check neighbors of s in Q
10         for edge a = (s, z) with z in Q do
11             if w(a) < D[z] then
12                 D[z] = w(a)
13                 Change entry of z in Q to ((z, a), D[z])
14     return T
```

Eksempel

- Vi har en Heap insert på for loop, som er O(log(|V|)), som gjøres maksimalt |V| ganger
- While loop itererer også |V| antall ganger med en heap remove som er O(log(|V|))
- Til slutt har vi en for løkke som gjøres maksimalt |E| ganger
- Summere alle O operasjonene som gir
- O(|V|log(|V|) + |V|log(|V|)|E|)
- = O(|V|log(|V|)(1 + |E|)
- = O(|V| |E|log(|V|))

O-Notasjon Lifehack Regler

Når det er ledd mellom plusstegn, så kan man ta hensyn til det leddet som vokser raskest

```
void printAllNumbersThenAllPairSums(int arr[], int size)
{
    for (int i = 0; i < size; i++)
    {
        printf("%d\n", arr[i]);
    }

    for (int i = 0; i < size; i++)
    {
        for (int j = 0; j < size; j++)
        {
            printf("%d\n", arr[i] + arr[j]);
        }
    }
}
```

Eksempel: O(n^3 + n^2 + n) = O(n^3) fordi n^3 vokser raskest når n øker :)

Konstanter og koffesienter kan man se bort ifra, eller redusere ned til 1

Eksempel 1: O(50n) = O(1n) = O(n)
Eksempel 2: O(1000) = O(1)

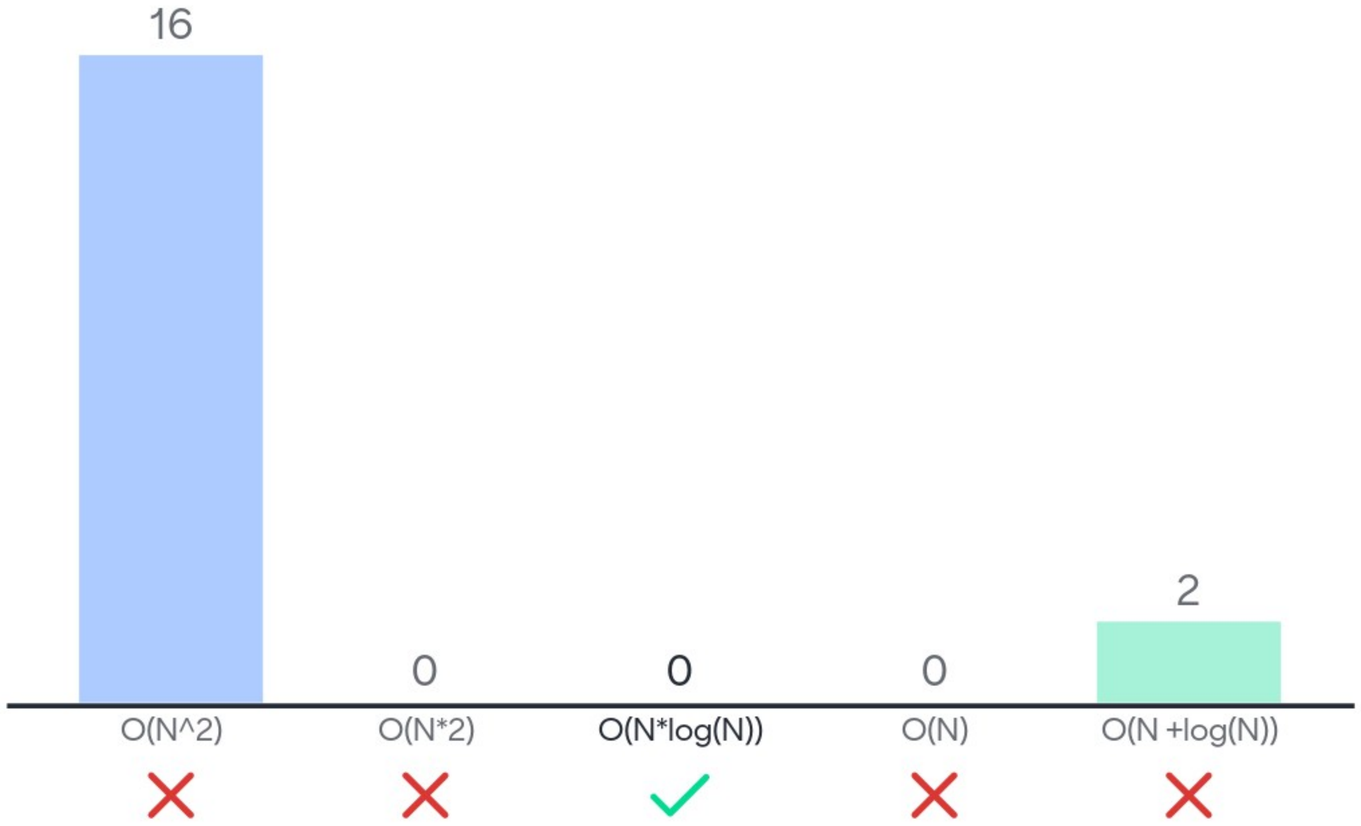
```
void printFirstItemThenFirstHalfThenSayHi100Times(int arr[], int size)
{
    printf("First element of array = %d\n",arr[0]);

    for (int i = 0; i < size/2; i++)
    {
        printf("%d\n", arr[i]);
    }

    for (int i = 0; i < 100; i++)
    {
        printf("Hi\n");
    }
}
```

Variabler kan ganges sammen for å gjøre det enklere å se hva som vokser raskest

Eksempel: O((n+15)(n+20)) = O(n^2 + 20n + 15n + 1520) = O(n^2) ettersom at n^2 vokser raskest

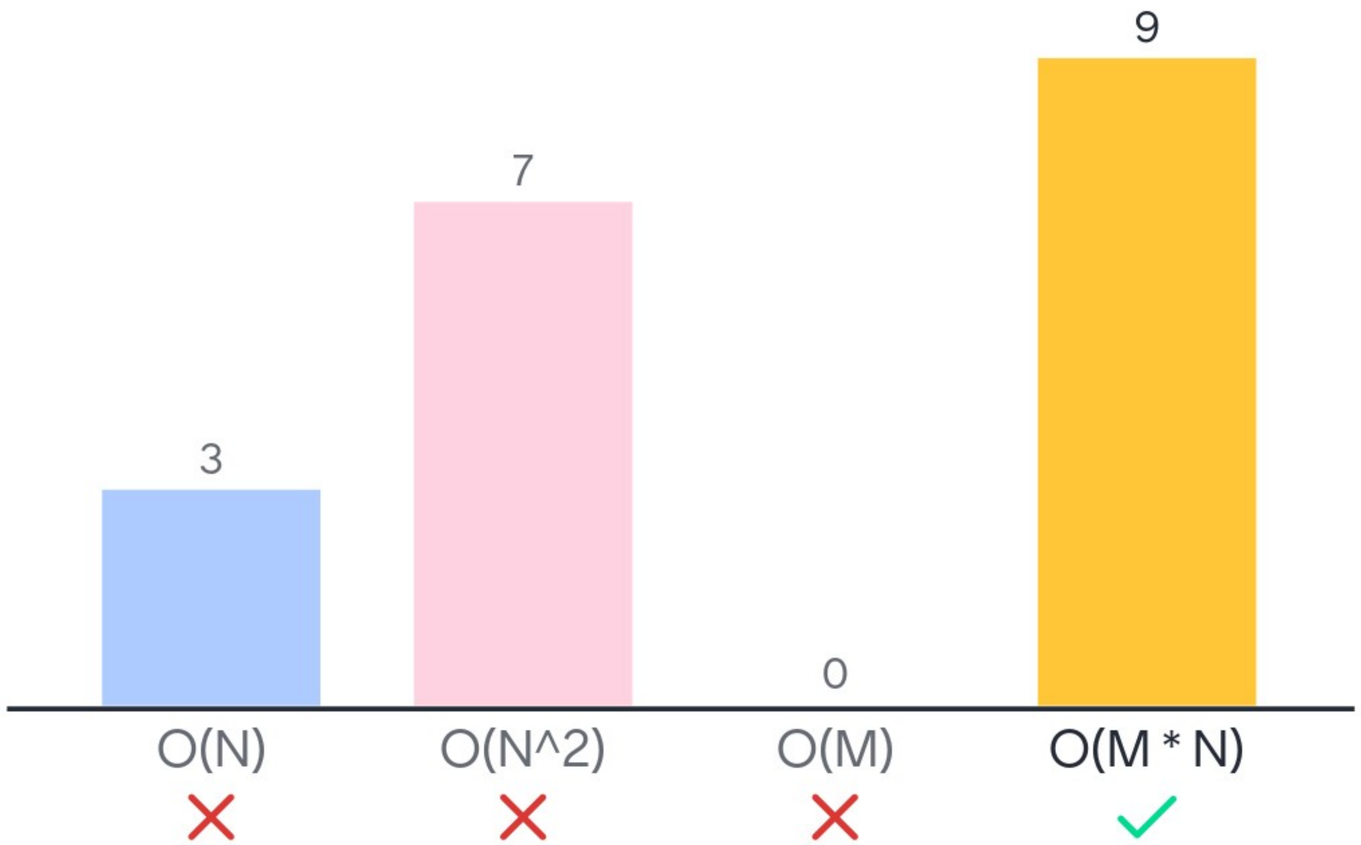
Dette eksempelet kan også løses ved å se bort fra konstantene, ettersom at n vokser raskest i begge parantesene. Da ser løsningen slik ut:
O((n+15) x (n+20))= O((n) x (n)) = O(n^2)

Hva er kjøretiden



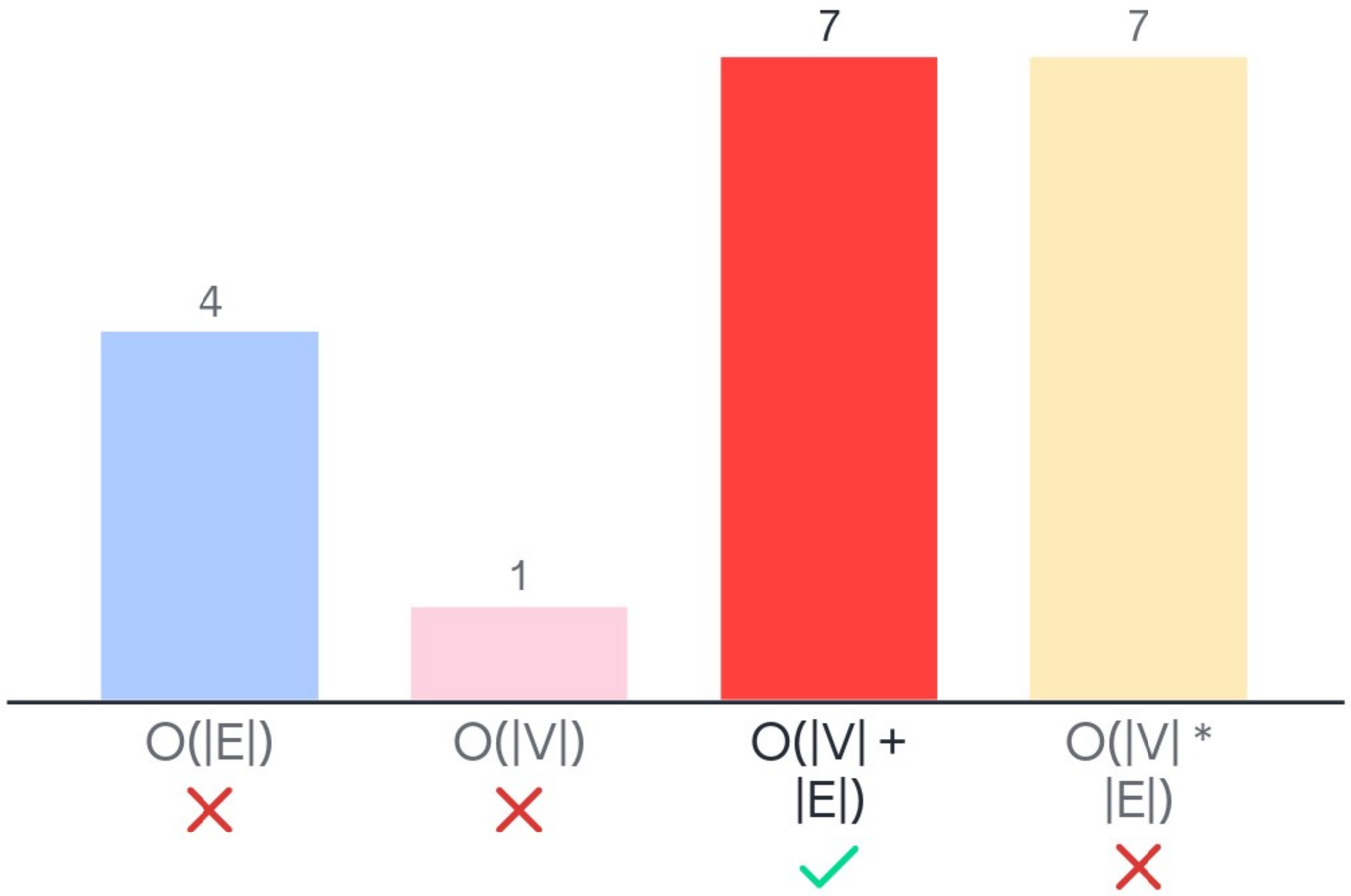

```java
// Hva er kjøretiden?
for(int i=1; i<=n; i*=2){
  for(int j=1; j<=i; j++){
    System.out.println("Hei");
  }
}
```

Hva er kjøretiden?



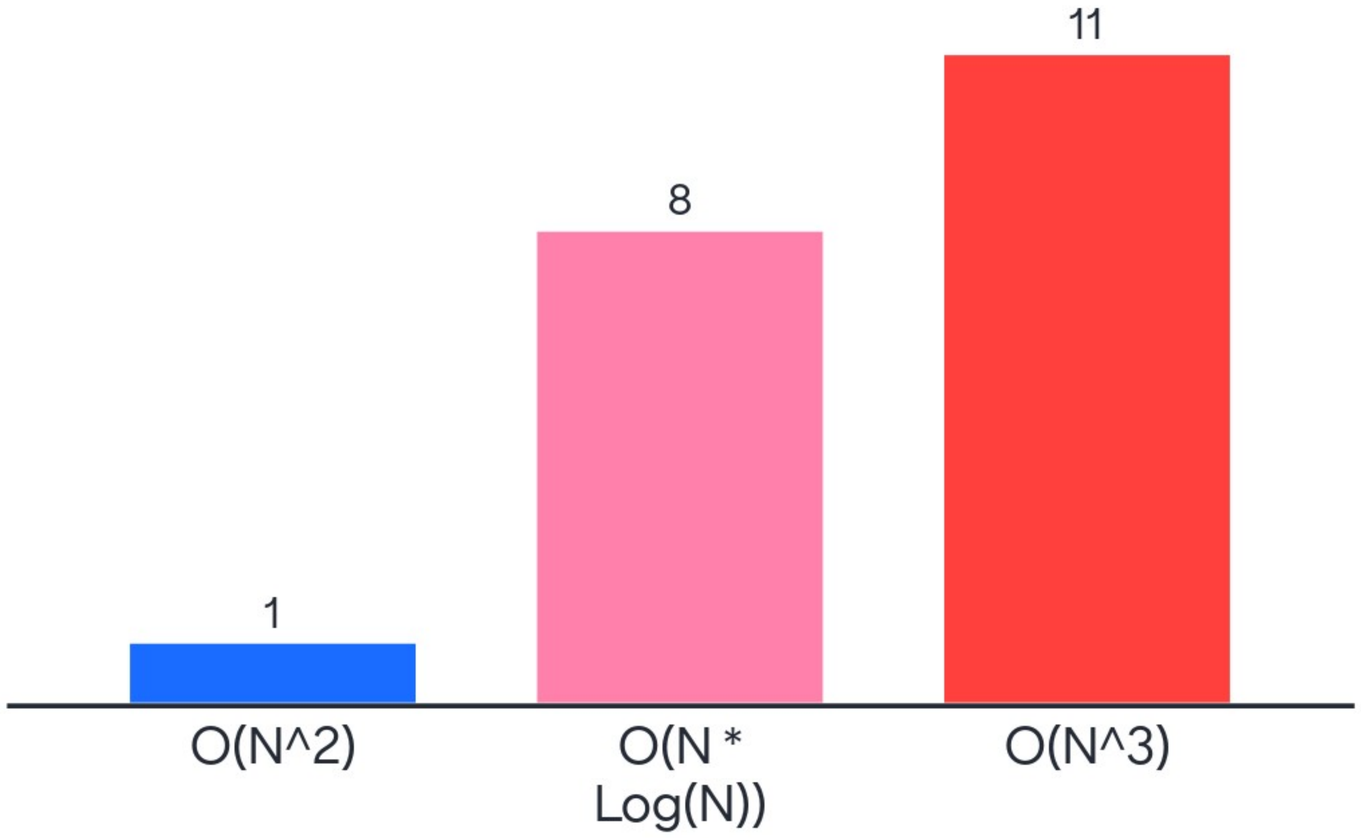

```java
class GFG{

// Driver Code
public static void main (String[] args)
{
    int a = 0;
    int b = 0;
    int N = 4;
    int M = 5;

    // Nested loops
    for(int i = 0; i < N; i++)
    {
        for(int j = 0; j < M; j++)
        {
            a = a + j;

            // Print the current
            // value of a
            System.out.print(a + " ");
        }
        System.out.println();
    }
}
}
```

Hva er kjøretiden?



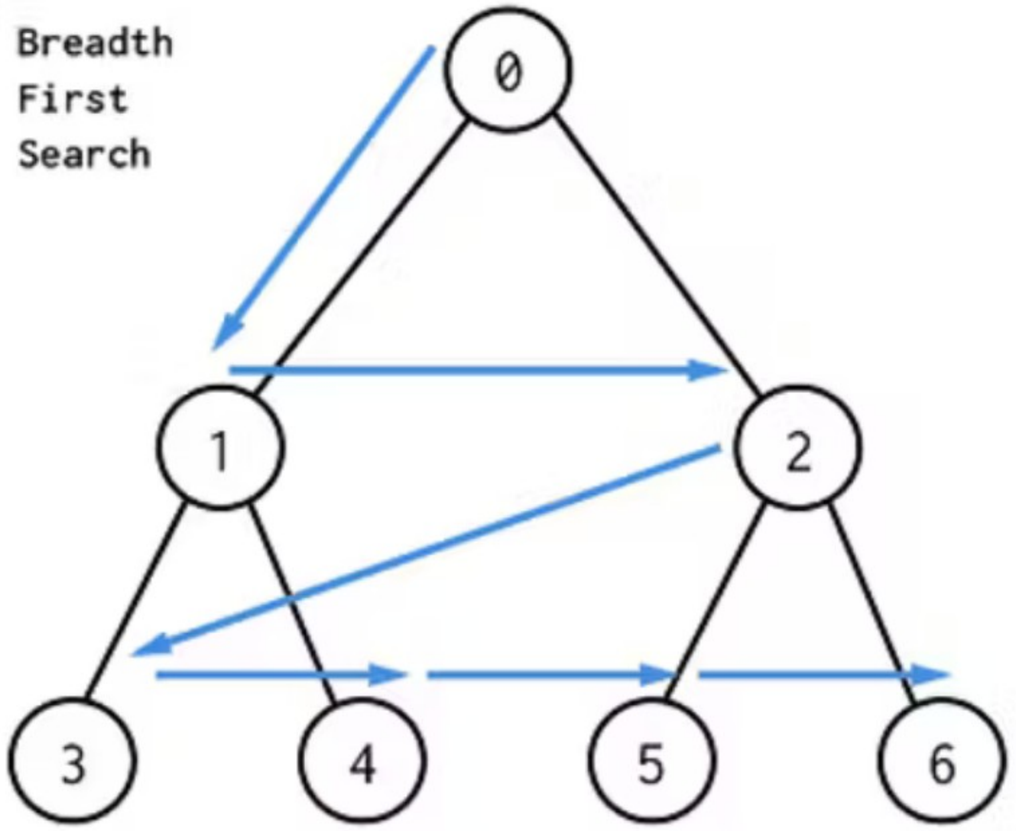

```python
def dfs_rec(G, s, visited, result):
    _, E, _ = G
    result.append(s)
    visited.add(s)
    for v in E[s]:
        if v not in visited:
            dfs_rec(G, v, visited, result)
    return result
```

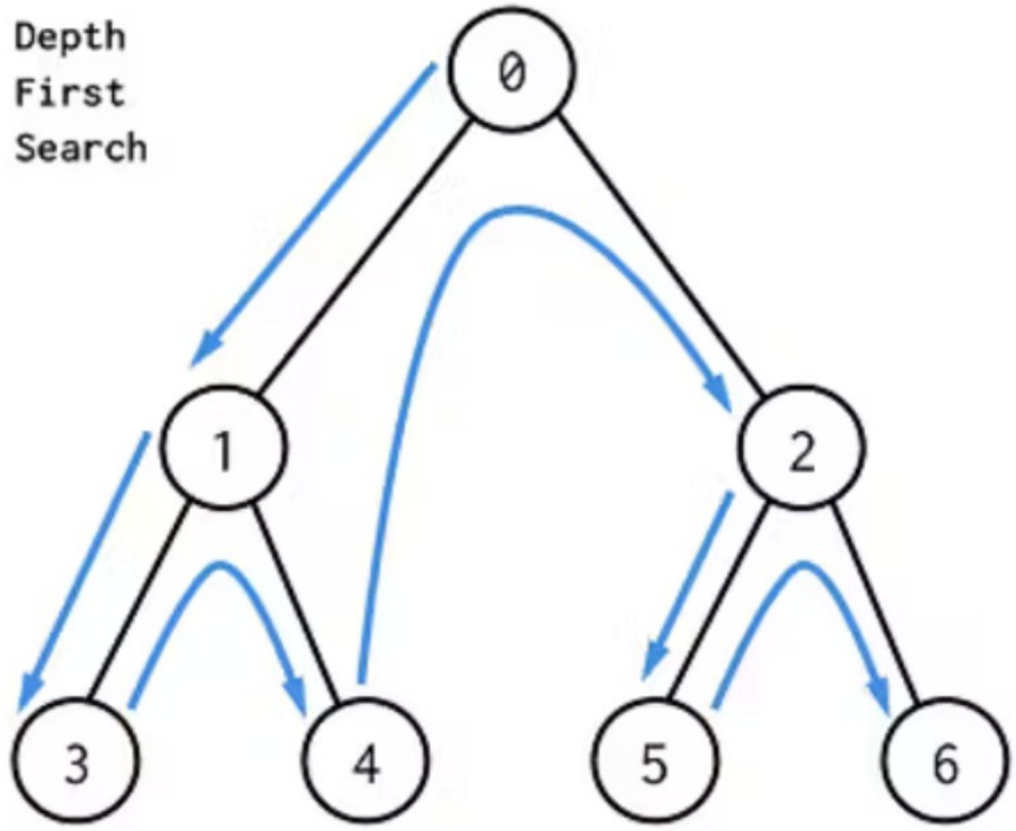
Hva er kjøretiden?




```
int x = 0;

for(int i=0; i<n; i++){
    for(int j=0; j<i^2; j++){
      x = x + j
    }
}
```

BFS og DFS





DFS VS BFS

```python
def DFSVisit(G, s, visited):
    stack = [s]
    while len(stack) != 0:
        u = stack.pop()
        if u not in visited:
            visited.add(u)
            print(u.name)
            for kanter in u.getEdges():
                stack.append(kanter.getOther(u))

def DFSFull(G):
    visited = set()
    for v in G[0]:
        if v not in visited:
            DFSVisit(G, v, visited)
```

```python
def BFSVisit(G, s, visited):
    queue = [s]
    visited.add(s)
    print(s.name)
    while len(queue) != 0:
        u = queue.pop(0)
        for kanter in u.getEdges():
            v = kanter.getOther(u)
            if v not in visited:
                print(v.name)
                visited.add(v)
                queue.append(v)



def BFSFull(G):
    visited = set()
    for v in G[0]:
            if v not in visited:
                BFSVisit(G, v, visited)
```

Kode

Pseudokode

Hva er pseudokode?

- En uformel beskrivelse av et program
- Ikke strenge Syntaktiske Regler
- Ikke ren kode
- Men er heller ikke kun naturlig språk
- Steg for steg beskrivelse av programmet

```
x_curr← Produce initial solution()
T←T0
while stopping criteria not true  do
    for i=1 to n
          x_cand←Produce a random solution
          if f(x_cand) < f(x_curr) then
               x_ curr ←x_cand
          else
              'Metropolis Algorithm
              x Produce a random number between (0,1)
              if   x < p ( T, x_curr, x_cand) then
                   x_curr ← x_cand
              end if
          end if
    end
    update(T)
end while
```

Eksempel

```
1:  procedure BAD_SORT(x)
2:      y ← {}
3:      while len(x) > 0 do
4:          winner ← x[0]
5:          for Integer k ← 0; k < len(x); k++ do
6:              if x[k] < winner then
7:                  winner ← x[k]
8:              end if
9:          end for
10:         Push the winner onto y
11:         Remove the winner from x
12:     end while
13:     return y
14: end procedure
```

Eksempel

Mentimeter

Pause

Felles Oppgaver

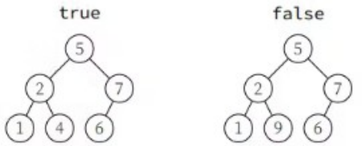
Er binærtreet et søketre? 10 poeng

Anta at du er gitt et binærtre B med unike heltall. Hvis v er en node i det binære treet, så gir

- v.`element` heltallet som er lagret i noden
- v.`left` venstre barn av v
- v.`right` høyre barn av v

Vi ønsker å sjekke om det binære treet også er et binært *søketre*. Under finner du spesifikasjonen for algoritmen, og to eksempler på trær som henholdsvis bør gi `true` og `false`.

```
Input: Rotnoden v av et binærtre B
Output: Returnerer true hvis binærtreet er et binært søketre,
        false ellers
1 Procedure CheckBST(v)
  | // ···
```



Det finnes flere gode løsninger på dette problemet. Følgende kan være behjelpelig når du tenker på en løsning:

- Du kan anta at du har prosedyrer `FindMin` og `FindMax` som henholdsvis finner minste og største tall i det binære treet. Siden treet ikke er garantert å være et binært søketre (eller balansert) så vil disse prosedyrene ha lineær tid.
- Det kan være lurt å dele algoritmen opp ved å lage en hjelpeprosedyre.

(a) Skriv ned egenskapen et binærtre må ha for å kunne kalles et binært *søketre*.

(b) Fullfør prosedyren over. Lavere kjøretidskompleksitet er mer poenggivende.

(c) Oppgi kjøretidskompleksiteten på algoritmen din med hensyn til antall noder i binærtreet.

Eksamen 2022

Finn par som summerer til x — 10 poeng

Du er gitt et array med unike heltall A og et heltall x. Du skal skrive en prosedyre som skriver ut alle par av heltall y og z i arrayet, der $y + z = x$. Tallet på en gitt posisjon i arrayet kan maksimalt inngå i én utskrift. Rekkefølgen parene (y, z) skrives ut i spiller ingen rolle, og rekkefølgen innad i parene spiller heller ingen rolle.

Input: Et array A av heltall, og et heltall x
Output: Skriver ut alle par $y, z \in$ A slik at $y + z = x$

```
1 Procedure FindSummands(A, x)
  | // ···
```

For eksempel skal et kall på `FindSummands`$([0, 2, 4, 6, 8, 10], 10)$ skrive ut parene $(0, 10)$, $(2, 8)$ og $(4, 6)$.

For begge deloppgaver: Gi pseudokode for en algoritme som løser problemet **og** oppgi kjøretidskompleksiteten på algoritmen (den trenger ikke begrunnes). Lavere kjøretidskompleksitet er mer poenggivende.

(a) Skriv en prosedyre `FindSummands` som beskrevet over, med antagelsen om at A er *sortert fra minst til størst*.

(b) Skriv en prosedyre `FindSummands` som beskrevet over, men du kan *ikke* anta at A er sortert. Hint: Du kan anta $O(1)$ for innsetting, oppslag og sletting i hashbaserte datastrukturer.

Eksamen 2021

Whops!-oppgjør — 10 poeng

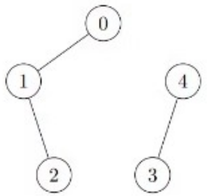
Ansatte på Institutt for informatikk har vært på instituttseminar hvor det har sklidd helt ut med utlån av små beløp mellom hverandre. De skal alle gjøre opp for seg ved å overføre penger via tjenesten Whops!. Dessverre har det oppstått et virvar av interne stridigheter mellom de ansatte, hvor mange har endt opp med å blokkere hverandre på Whops!, som gjør det vanskelig å få oppgjøret til å gå opp. Administrasjonen ser fortvilet på situasjonen, og skjønner at dette problemet vil oppstå igjen i årene fremover, hvor det også skal ansettes vanvittig mange. De har rutiner på plass for å få oversikt over hvor mye hver ansatt har lagt ut eller skylder totalt. Nå ber de om hjelp fra studentene i IN2010 (som på dette tidspunktet har blitt eksperter på algoritmer og datastrukturer) til å finne en generell måte å sjekke om oppgjøret kan gjennomføres gjennom Whops! eller ikke, som også vil skalere etterhvert som Institutt for Informatikk får ubegripelig mange ansatte.

Du setter flittig i gang, og formaliserer problemet som et grafproblem. Du lar $G = (V, E)$, der hver ansatt være representert ved en node $v \in V$, og lar det være en (urettet) kant $\{u, v\} \in E$ mellom to ansatte dersom de ikke har blokkert hverandre på Whops!. For hver ansatt $v \in V$ kan du slå opp i en tabell T, der $\mathsf{T}[v]$ er et heltall som angir hvor mye v skylder eller har lagt ut totalt. Dersom $\mathsf{T}[v]$ er negativ, betyr det at v skylder penger, men dersom $\mathsf{T}[v]$ er positiv betyr det at v har lagt ut penger. Du vet at summen av alle tallene i T er nøyaktig 0. Merk at det betyr at dersom alle kunne utveksle penger mellom hverandre gjennom Whops! (muligens via andre ansatte) ville oppgjøret alltid gått opp.

Du skal gi en algoritme som tar G og T som input og svarer **true** hvis oppgjøret kan gjøres gjennom Whops!, og **false** dersom det ikke er mulig.

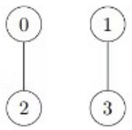
Her er to eksempler på hvordan en G og T kan se ut. For eksempelet med G_1 og T_1 kan oppgjøret gjøres gjennom Whops!, men for G_2 og T_2 er det ikke mulig å gjøre oppgjøret gjennom Whops!.

G_1



v	$\mathsf{T}_1[v]$
0	500
1	−200
2	−300
3	−50
4	50

G_2



v	$\mathsf{T}_2[v]$
0	300
1	−250
2	0
3	−50

Eksamen 2021

Whops!-logger **10 poeng**

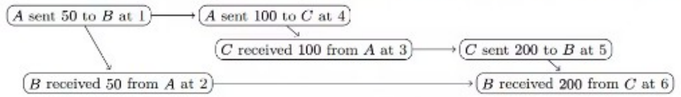
Tjenesten Whops! lar brukere overføre penger mellom hverandre gjennom en app. De prøver i så stor grad som mulig å unngå å lagre informasjon om brukerene sine sentralt, og lar heller informasjonen lagres på brukeren sin egen mobiltelefon.

En gang i blant har de behov for å anskaffe en komplett logg av overføringer for en periode for å kunne feilsøke systemene sine. Da henter de inn *lokale* logger fra noen utvalgte mobiltelefoner i systemet. En lokal logg er en liste av hendelser, der hver hendelse beskriver enten at et beløp er sent eller mottatt. I tillegg inneholder hver hendelse et tidsstempel som forteller når beløpet ble sendt eller motatt. For enkelhets skyld er tidsstemplene representert som positive heltall. Her er et lite eksempel på tre logger fra mobiltelefoner A, B og C.

A	B	C
A sent 50 to B at 1	B received 50 from A at 2	C received 100 from A at 3
A sent 100 to C at 4	B received 200 from C at 6	C sent 200 to B at 5

Dessverre er tidsstemplene fra mobiltelefonene upålitelige. De observerer ofte at et mottak av et beløp forekommer *før* beløpet ble sendt, i følge tidsstemplene. I eksempelet ovenfor mottar C et beløp på 100 *før* A har sendt beløpet til C.

For å approksimere en riktig logg lar de hver hendelse fra de lokale loggene være noder i en rettet graf $G = (V, E)$. Dersom to hendelser u og v kommer direkte etter hverandre i en lokal logg, så finnes det en rettet kant (u, v) i grafen. For hver hendelse u som representerer at et beløp er sendt, så finnes det en rettet kant (u, v) der v er hendelsen for det korresponderende mottaket av det beløpet. Her er grafen for eksempelet ovenfor:



Den ønskede komplette loggen for dette eksempelet skal se slik ut:

A sent 50 to B at 1
B received 50 from A at 2
A sent 100 to C at 4
C received 100 from A at 3
C sent 200 to B at 5
B received 200 from C at 6

Hver node v kan aksessere tidsstempelet fra loggen ved $v.\texttt{ts}$. Skriv en algoritme som tar rettet graf $G = (V, E)$ som input, og skriver ut alle noder i V i en rekkefølge slik at:

- Dersom det finnes en kant fra u til v så skal u skrives ut før v.
- u skal skrives ut før v hvis $u.\texttt{ts} < v.\texttt{ts}$, så lenge det ikke bryter med det første kravet.

Eksamen 2021

Spørsmål

0 questions
0 upvotes