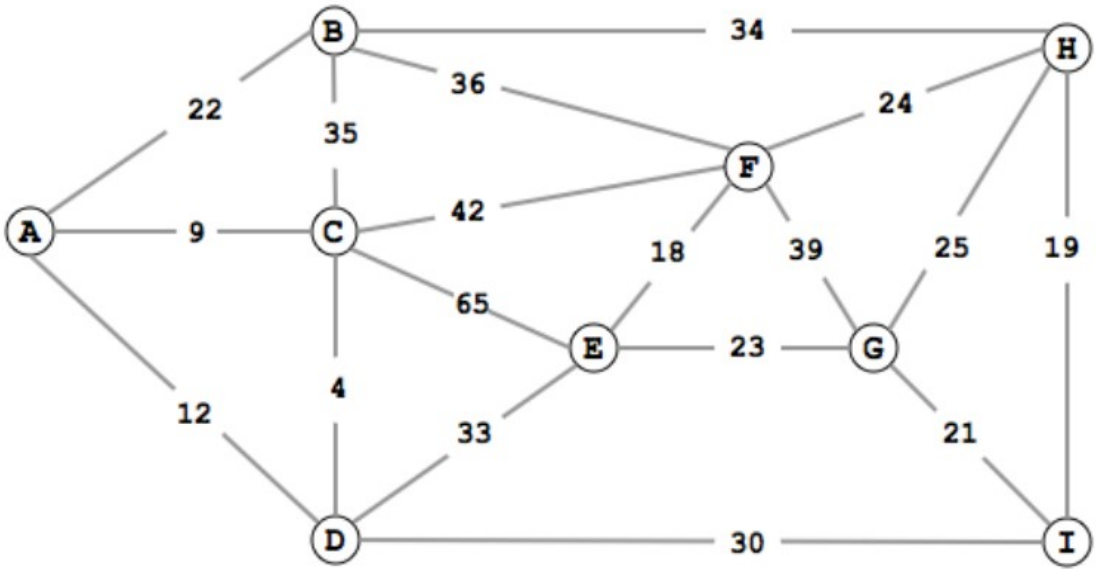
Exercise 1

Find a *minimum spanning tree* for graph below using Prim's algorithm. Your answer must be a list of edges, E-G. D-E, A-C etc., which shows in which order the algorithm will establish connections between vertices.



Could you end up with a different answer if Kruskal's algorithm were applied? Explain your answer.

C-D, A-C, A-B, D-I, H-I, G-I, E-G, E-F

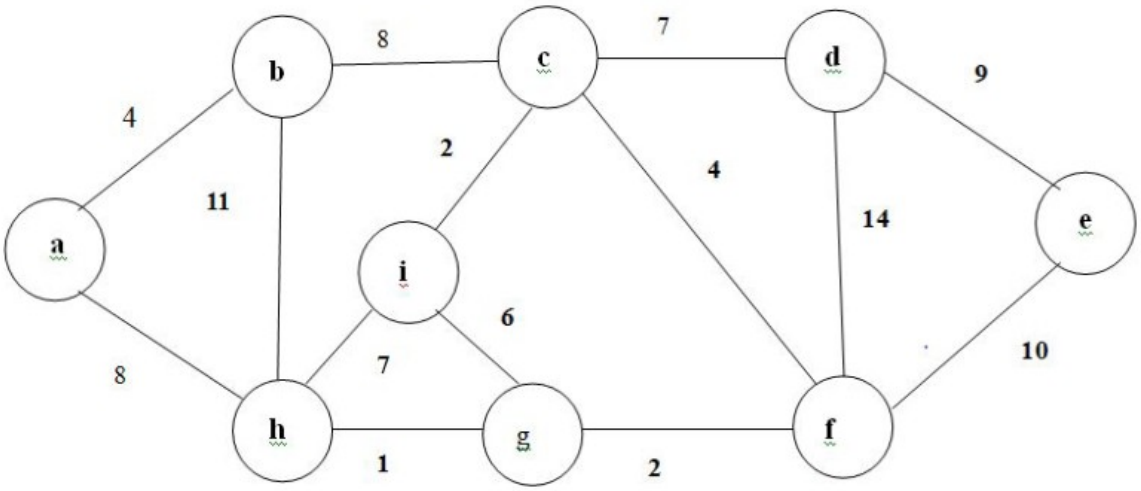
Ja jeg vil argumentere for at du godt kunne ende op med et andet træ hvis du havde brugt Kruskal fordi du ikke er tvunget til at edgesne skal være forbundet til det allerede eksisterende træ, men i stedet bare altid tager de edges med lavest kost.

C-D, A-C, E-F, H-I, G-I, A-B, E-G, D-I

Vi kommer frem til det samme træ, men edgesne bliver fundet i en anden rækkefølge.

Exercise 2

Find a *minimum spanning tree* for below graph using Kruskal's algorithm. Your answer must be a list of edges, e.g. d-e, a-c etc., which shows in which order the algorithm will establish connections between vertices.



g-h, c-i, f-g, a-b, c-f, c-d, a-h, d-e

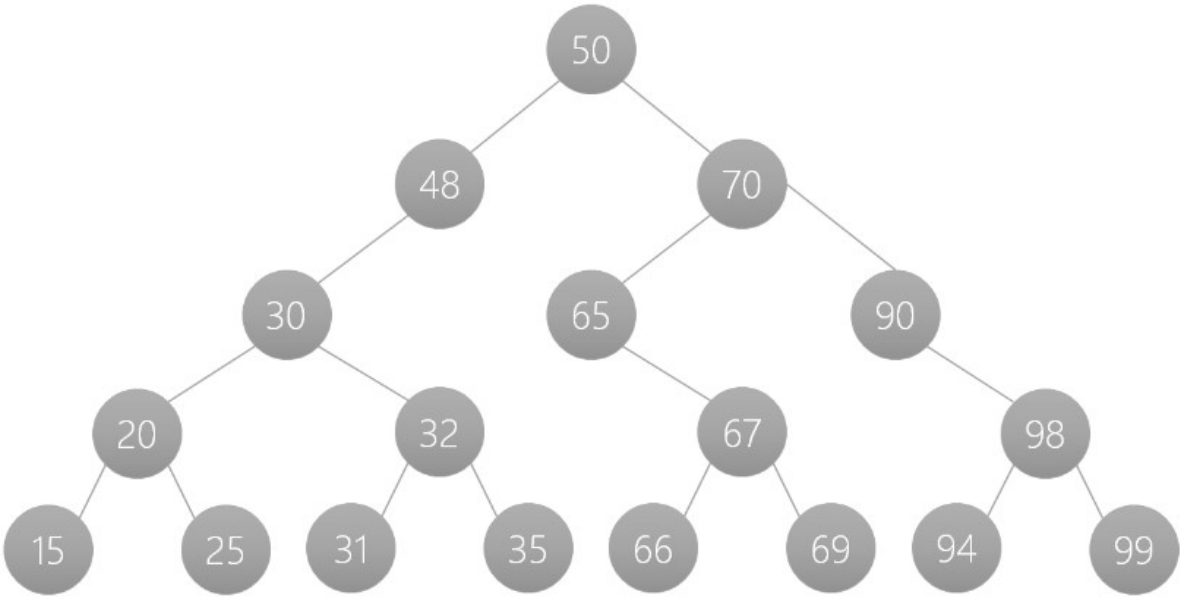
Et andet spanning tree ville potentielt kunne være fundet med prim. Det er i hvert fald 100% sikker at edgesne ville blive sluttet i en anderledes rækkefølge i og med prim altid tager og forbinder den edge med den laveste cost, som stadig er i forbindelse med det allerede eksisterende træ.

g-h, f-g, c-f, c-i, c-d, a-h, a-b, d-e

Det lader til at være det samme træ, men edgesne bliver fundet i en anden rækkefølge som forslået.

Exercise 4

The figure below is a binary search tree:



List the order in which the nodes will be visited in an in-order and a level order traversal.
Which simple data structure can preferably be used for a level order traversal and why?
What is the internal path length of the tree, and what other characteristics does the tree have? Explain your answers.

I in-order traversal bliver det venstre barn besøgt først, efterfulgt af parent noden og til sidst det højre barn. Derved kommer rækkefølgen til at se således ud:

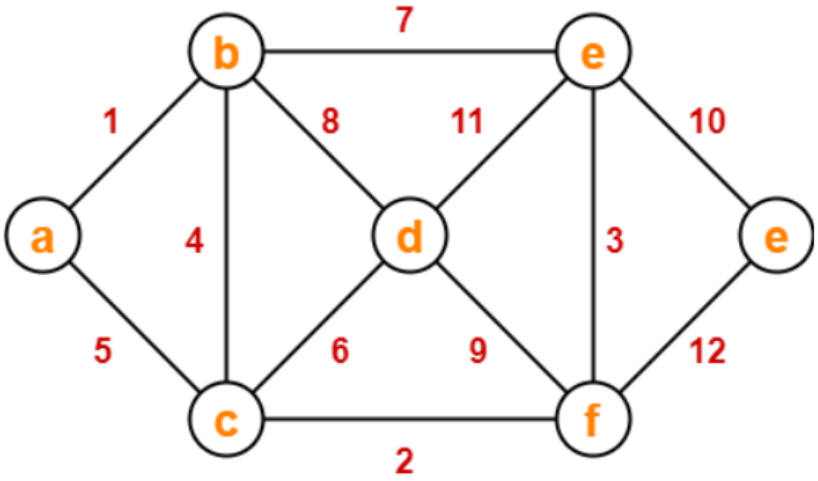
15,20,25,30,31,32,35,48,50,65,66,67,69,70,90,94,98,99

I level-order traversal bliver alle noder på dybde d processeret før noderne på dybde d+1. Derved kommer rækkefølgen noderne bliver besøgt i til at se således ud:

50,48,70,30,65,90,20,32,67,98,15,25,31,35,66,69,94,99

Exercise 5

Find a *minimum spanning tree* for the graph below using Prim's algorithm. Your answer must be a list of edges, fx d-e, a-c etc., which show the order in which the algorithm will establish connections between the nodes (vertices). Also give the size of the tree (the sum of the weights in the tree).



a-b, b-c, c-f, e-f, c-d, e-g

1+4+2+3+6+10 = 26

Det er muligt at der ville være fundet et andet træ ved brug af kruskal fordi den altid forbinder den edge med lavest kost uden at tænke på om den er forbundet til det allerede eksisterende træ. Under alle omstændigheder ville edgesne findes i en anden rækkefølge:

a-b, c-d, e-f, b-c, c-d, e-g

Det samme træ, men en anden rækkefølge af edges.

Exercise 6

The table below is a hash table. The hash function is X % tableSize, and the unique key af the element is the number of the letter in the alphabet. Thus the letter E has the key 5, and the letter O has the key 15. For collision resolution *quadratic probing* is used.

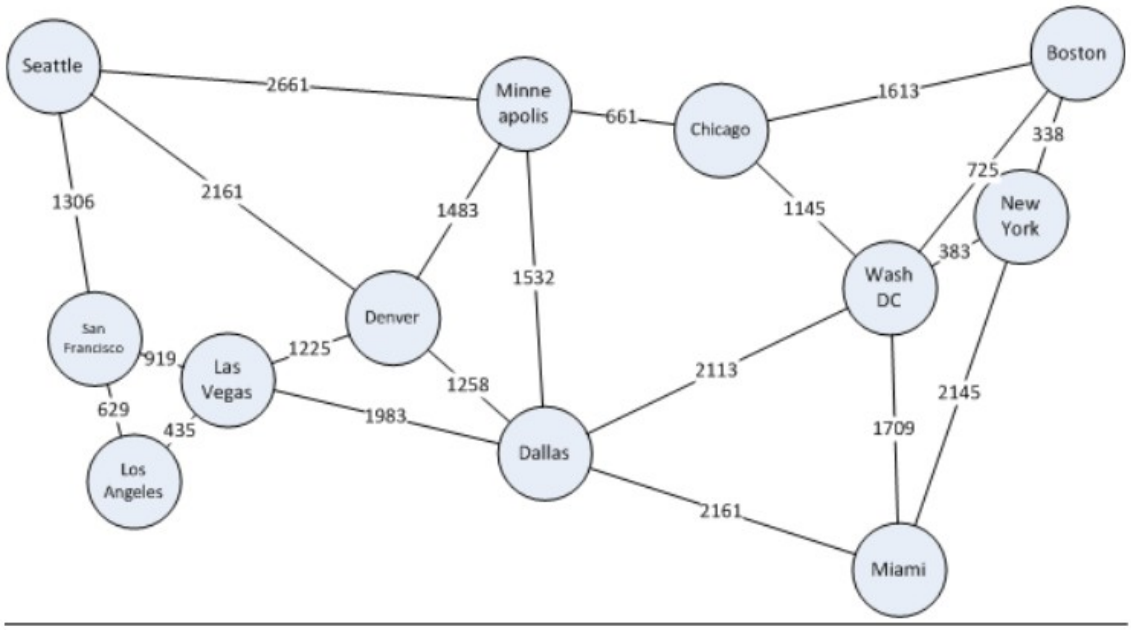
Index	Value
0	
1	A
2	W
3	C
4	O
5	E
6	
7	
8	S
9	
10	

The load factor of the table is above 0.5 and a rehashing should be performed. Show the table after the rehashing has been performed.

Når der rehashes fordobles tabelstørrelsen (til nærmeste primtal) det vil altså sige at tabelstørrelsen nu bliver til 23.

0	W
1	A
2	
3	C
4	
5	E
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	O
16	
17	
18	
19	S
20	
21	
22	

Exercise 8



Apply Dijkstra's shortest path algorithm to above graph with Seattle as starting point.

The answer must give a short explanation of how the algorithm operates and a list of used edges naming the two vertices (nodes) involved, for example

Minneapolis-Dallas
Chicago-Wash DC
New York-Boston
etc.

Der startes med at laves forbindelse til alle de byer der kan nås fra Seattle.

Seattle-San Francisco, Seattle-Denver, Seattle-Minneapolis

Vi eksplorerer nu fra San Francisco fordi det er der der er den korteste distance til.

San Francisco-Las Vegas, San Francisco-Los Angeles

Vi tager nu og eksplorerer fra Los Angeles da det er her der er den korteste distance til.

Los Angeles-Las Vegas

Der laves ingen ny forbindelse for distance ikke er kortere. Der eksploreres nu fra Denver.

Denver-Las Vegas, Denver-Minneapolis, Denver-Dallas

Der laves forbindelse til Dallas fra Denver. Nu eksploreres der fra Las Vegas.

Las Vegas-Denver, Las Vegas-Dallas.

Ingen nye forbindelser laves. Minneapolis eksploreres.

Minneapolis-Denver, Minneapolis-Dallas, Minneapolis-Chicago

Chicago eksploreres.

Chicago-Boston, Chicago-Washington DC

Dallas udforskes.

Dallas-Miami, Dallas-Minneapolis, Dallas-Washington DC

Washington DC udforskes.

Washington DC-Boston, Washington DC-New York, Washington DC-Miami

Der laves forbindelse til New York. New York udforskes. Ingen nye forbindelser laves

Miami udforskes og ingen nye forbindelser laves.

By	Kendt	Distance	Tætteste by
Seattle	T	0	0
Minneapolis	T	2661	Seattle
Chicago	T	3322	Minneapolis
Boston	F	4935	Chicago
San Francisco	T	1306	Seattle
Las Vegas	T	2225	San Francisco
Denver	T	2161	Seattle
Washington DC	T	4467	Chicago
New York	T	4850	Washington DC
Los Angeles	T	1935	San Francisco
Dallas	T	3419	Denver
Miami	T	5580	Dallas

Exercise 10

What is the Big-Oh time complexity of the following method? Please explain your answer.

```
public static int myMethod( int N )
{
   int x = 0;
   for (int i = 0; i < N; i++)
        for (int j = 0; j < N/2; j++)
            for (int k = 1; k < N;)
            {
                x++;
                k *= 2;
            }
   return x;
}
```

Det yderste loop kører N gange. Det næste loop kører N/2 gange. Det sidste loop må køre log(N) gange fordi k bliver fordoblet for hver gang. Det vil altså sige $N \cdot \frac{N}{2} \cdot \log(N)$. Dermed bliver store O tidskompleksiteten:

$$O(N^2 \log N)$$

Exercise 12

Solve the following recurrence. All relevant steps must be shown leading to a Big-Oh answer.

T(N) = T(N-1) + (N-1)

Der telescopes på ovenstående ligning:

$$T(N-1) = T(N-2) + (N-2)$$

$$T(N-2) = T(N-3) + (N-3)$$

...

$$T(2) = T(1) + 1$$

Alle ligningerne lægges sammen:

$$T(N) + T(N-1) + T(N-2) + \ldots + T(2) = T(N-1) + (N-1) + T(N-2) + (N-2) + T(N-3) + (N-3) + \ldots + T(1) + 1$$

Det vil sige:

$$T(N) = T(1) + \sum_{i=1}^{N-1} i$$

$$T(N) = T(1) + N - 1$$

Dermed bliver store o tidskompleksiteten:

$$T(N) = O(1) + O(N)$$

$$= O(N)$$

Altså kan det konkluderes at store O tidskompleksiteten er $O(N)$.

Exercise 14

Consider the hash-table below

Index	Value
0	D
1	H
2	V
3	
4	
5	P
6	
7	X
8	E
9	
10	

Indexes 3, 4, 6, 9 and 10 are vacant. What happens if an element hashing to index 1 is to be inserted? The strategy used for collision resolution is quadratic probing.

Der er ikke plads på index et så der bliver probet og lagt 1*1 til index 1 altså index 2. Der er heller ikke plads på index 2 så der lægges 2*2 til index 1 i stedet, der kigges altså på index 5. På index 5 er der heller ikke plads, så der lægges i stedet 3*3 til index 1, der kigges på index 10. Denne er tom og elementet indsættes her.

Exercise 15

What is the Big-Oh time complexity of the following method? Please explain your answer.

```
public static int myMethod1( int[] arr )
{
            int x = 0;
            for (int i = 0; i < arr.length/2; i++)
                        for (int j = 0; j < arr.length; j++)
                                    for (int k = 0; k < arr.length; k++)
                                    {
                                                x++;
                                                if (k==arr.length/2)
                                                            break;
                                    }
            return x;
}
```

The attribute *length* denotes the number of elements in the array.

Det første loop køres N/2 gange, det næste loop køres N gange og det sidste loop køres N/2 gange. Det vil altså sige $\frac{N}{2} \cdot N \cdot \frac{N}{2}$. Dermed bliver store O tidskompleksiteten:

$$O\left(N^3\right)$$

Exercise 17

A quadratic probing hash table is used to store 2.500 `String` (C++: `string`) objects each 6 characters long. Determine the minimum size of the table and the storage needed (in bytes) in order to avoid unresolvable collisions.

Som udgangspunkt kræves der dobbelt så meget plads i hashtabellen som det antal data man ønsker at lagre. Det vil altså sige at minimumstørrelsen for tabellen bør være 5000 pladser. I forhold lagerpladsen skal der lagres 6 karakterer pr. index, én karakter fylder 8 bit (antaget at det er ASCII karakterer), altså er det 6*8 bit = 48 bit eller 6 bytes pr. index i hashtabellen. Det vil sige at alt i alt skal hashtabellen bruge 6*5000 bytes hvilket er 30 KB.

Exercise 19

Solve the following recurrence. All relevant steps must be shown leading to a Big-Oh answer.

T(N) = T(N/2) + N

$$T\left(\frac{N}{2}\right)=T\left(\frac{N}{4}\right)+\frac{N}{2}$$

$$T\left(\frac{N}{4}\right)=T\left(\frac{N}{8}\right)+\frac{N}{4}$$

...

$$T(2)=T(1)+2$$

De ovenstående ligninger lægges sammen:

$$T(N)+T\left(\frac{N}{2}\right)+T\left(\frac{N}{4}\right)+\ldots+T(2)=T\left(\frac{N}{2}\right)+N+T\left(\frac{N}{4}\right)+\frac{N}{2}+T\left(\frac{N}{8}\right)+\frac{N}{4}+\ldots+T(1)+2$$

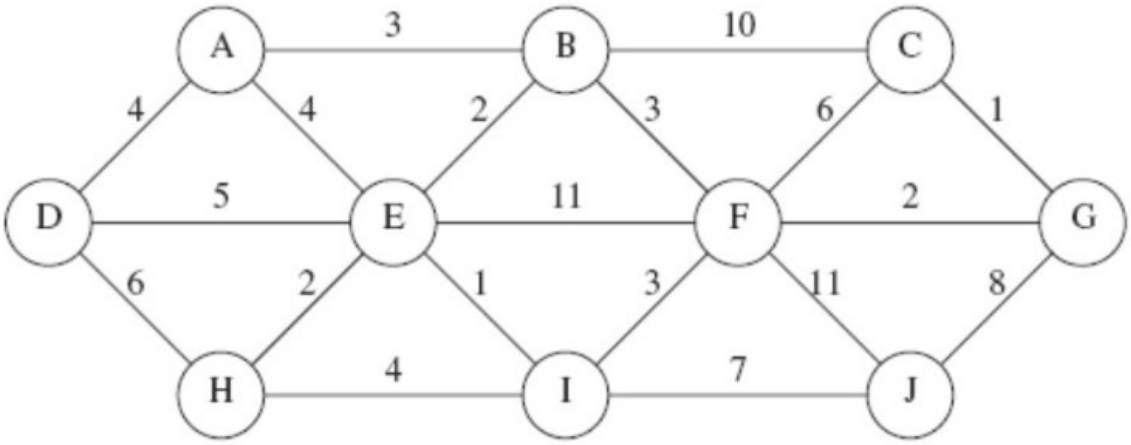
$$T(N)=T(1)+N\sum_{i=0}^{k}\frac{1}{2^i}$$

Dette betyder:

$$T(N)=O(1)+O(N)$$

Altså er tidskompleksiteten $O(N)$.

Exercise 20



Find a minimum spanning tree of the above graph using Prim's algorithm. Your answer must be a list of edges, e.g. E-I, F-J etc., showing the order in which the algorithm will establish connections between vertices.

C-G, F-G, B-F, B-E, E-I, E-H, A-B, A-D, I-J

Med Kruskal edgesne blive fundet på følgende måde:

C-G, E-I, B-E, E-H, F-G, A-B, B-F, A-D, I-J

Exercise 23

What is the Big-Oh time complexity of the following method? Please explain your answer.

```
public static int myMethod( int[] arr )
{
     int x = 0;
     for (int i = 0; i < arr.length; i++)
          for (int j = 0; j < arr.length/2; j++)
              for (int k = 0; k < arr.length; k++)
              {
                  x++;
                  if (k==1)
                      break;
              }
     return x;
}
```

The attribute *length* denotes the number of elements in the array.

Det yderste loop kører N gange, det næste loop kørere N/2 gange, det inderste loop kører 2 gange, fordi der breakes når k er lig 1. Det vil altså sige $N \cdot \frac{N}{2} \cdot 2$. Dermed bliver tidskompleksiteten:

$$O\left(N^2\right)$$

Opgave 4 (15 %)

Nedenstående figur viser en tom hopscotch hash tabel:

Indeks	Værdi	Hop
20		0000
21		0000
22		0000
23		0000
24		0000
25		0000
26		0000
27		0000
28		0000
29		0000
30		0000
31		0000

Opgaven går ud på at indsætte følgende elementer i tabellen og opdatere 'hoppen' (the hop) i overensstemmelse hermed:

Værdi	Hasher til indeks
A:	29
B:	24
C:	31
D:	22
E:	24
F:	29
G:	23
H:	22

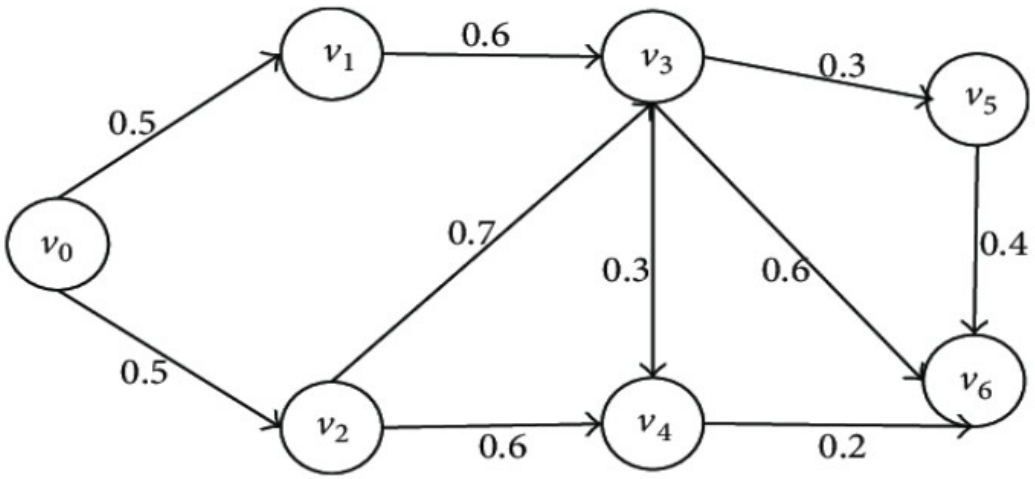
Det antages, at loadfaktoren og rehashing er irrelevant.
Forklar med nogle få ord, i hvilke situationer hopscotch hashing er at foretrække fremfor collision resolution med anvendelse af quadratic probing.

Indeks	Værdi	Hop
20		
21		
22	D	1100
23	H	0001
24	B	1100
25	E	0000
26	G	0000
27		
28		
29	A	1100

30	F	0000
31	C	1000

I nogle arkitekturer er nærheden af data vigtig. Med hopscotch sørger vi for at data der hasher til samme indeks ligger tæt på det indeks, altså opnår vi en nærhed. Med quadratic probing kan vi risikere at dataen ligger langt væk fra indekset det hasher til.

Opgave 5 (15 %)



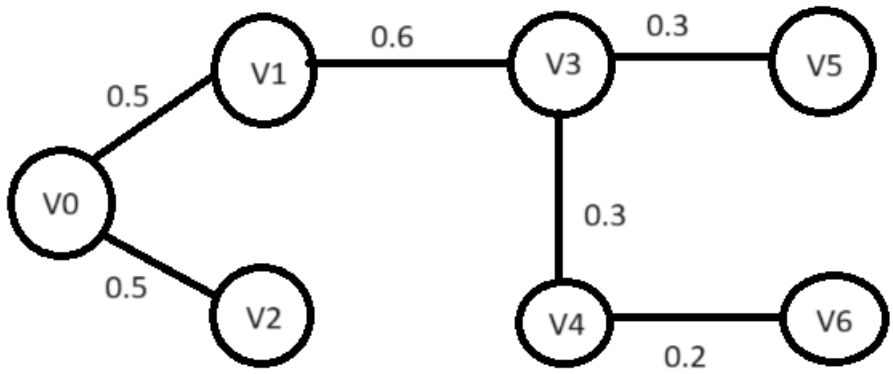
Nedenstående tabel viser startkonfigurationen for en traversering af ovenstående graf ved anvendelse af Dijkstras algoritme. Vis slutkonfigurationen for en traversering af grafen startende i vertex v_0.

v	known	d_v	p_v
v_0	false	0	0
v_1	false	∞	0
v_2	false	∞	0
v_3	false	∞	0
v_4	false	∞	0
v_5	false	∞	0
v_6	false	∞	0

Hvordan ville et minimum spanning tree for grafen se ud, hvis den var undirected? Angiv hvilken algoritme, der er anvendt og træets vægt.

V	Known	Dv	Pv
V0	T	0	0
V1	T	0.5	V0
V2	T	0.5	V0
V3	T	1.1	V1
V4	T	1.1	V2
V5	T	1.4	V3
V6	T	1.3	V4

Minimum spanning træet for den undirected graf ville se således ud:



Der er brugt Prim's algoritme og edgesne ville være fundet i følgende rækkefølge:

V4-V6, V3-V4, V3-V5, V1-V3, V0-V1, V0-V2

Vægten af træet ville være:

0.2+0.3+0.3+0.6+0.5+0.5=2.4

Opgave 2 (10 %)

Nedenfor er vist indholdet af et array.

{0,5,11,17,12,68,21,38,18,15,81,93,22,23,65,41,66,59,17,16}

En prioritetskø kan implementeres som et array, hvor indeks 0 ikke benyttes.

Kan arrayet repræsentere en prioritetskø? Forklar dit svar.

Forklar derudover hvorfor en prioritetskø kan implementeres i et simpelt array med positive heltal.

I en prioritetskø kan en forældres børn findes ved at gange indekset af forælderen i med 2, dette giver venstre barn og højre barn findes ved 2*i +1. I tilfælde af en MinHeap prioritetskø skal hver forælder være mindre end dens bør.

5<11, 17

11<12, 68

17<21, 38

12<18, 15

68<81, 93

21<22, 23

38<65, 41

18<66, 59

15<17, 16

Da alle forældre er mindre end deres børn som vist ovenover, kan dette array godt repræsentere en MinHeap prioritetskø.

En prioritetskø er også det som man vil kalde for et komplet binært træ. Alle binær træer kan gemmes som arrays, da det er en række af forskellige data.

Opgave 3 (20 %)

Hvad er Store O (Big-Oh) tidskompleksiteten for metoden myMethod1? Begrund dit svar.

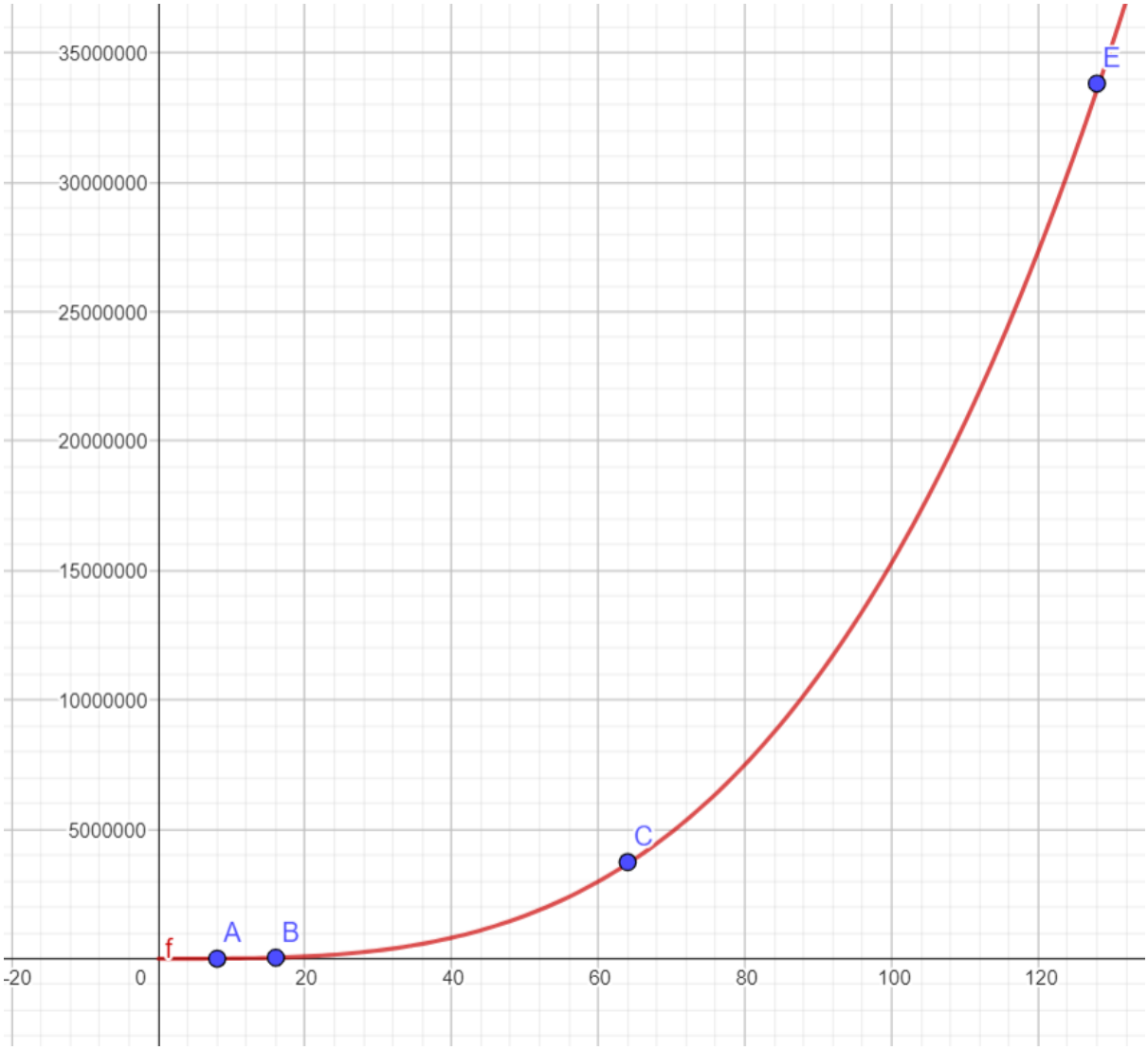
```
int myMethod1(int x)
{
    int y = 0;
    for (int i = 0; i <= x; i++)
    {
        if (x < 256)
            y += myMethod2(x*x);
        else
            y += myMethod2(x/2);
    }
    return y;
}
```

```
int myMethod2(int x)
{
    for (int i = 0;i <= x; i++)
    {
        if (x < 256)
            y += myMethod3(x);
        else
            y += myMethod3(x*2);
    }
    return y;
}
int myMethod3(int x)
{
    int y = 0;
    for (int i = x; i > 0; i/=2)
        y++;
    return y;
}
```

Der er to korrekte, forskellige svar: ét hvor parameteren x er større end eller lig med 256, og ét hvor parameteren er mindre end 256.
Det kan stærkt anbefales at afvikle mindst to eksperimenter med koden, fx ét hvor parameteren er 256, og ét hvor parameteren er mindre end 256, fx 64. Derudover bør du analysere koden for at få bekræftet det estimat, du når frem til med eksperimenterne.

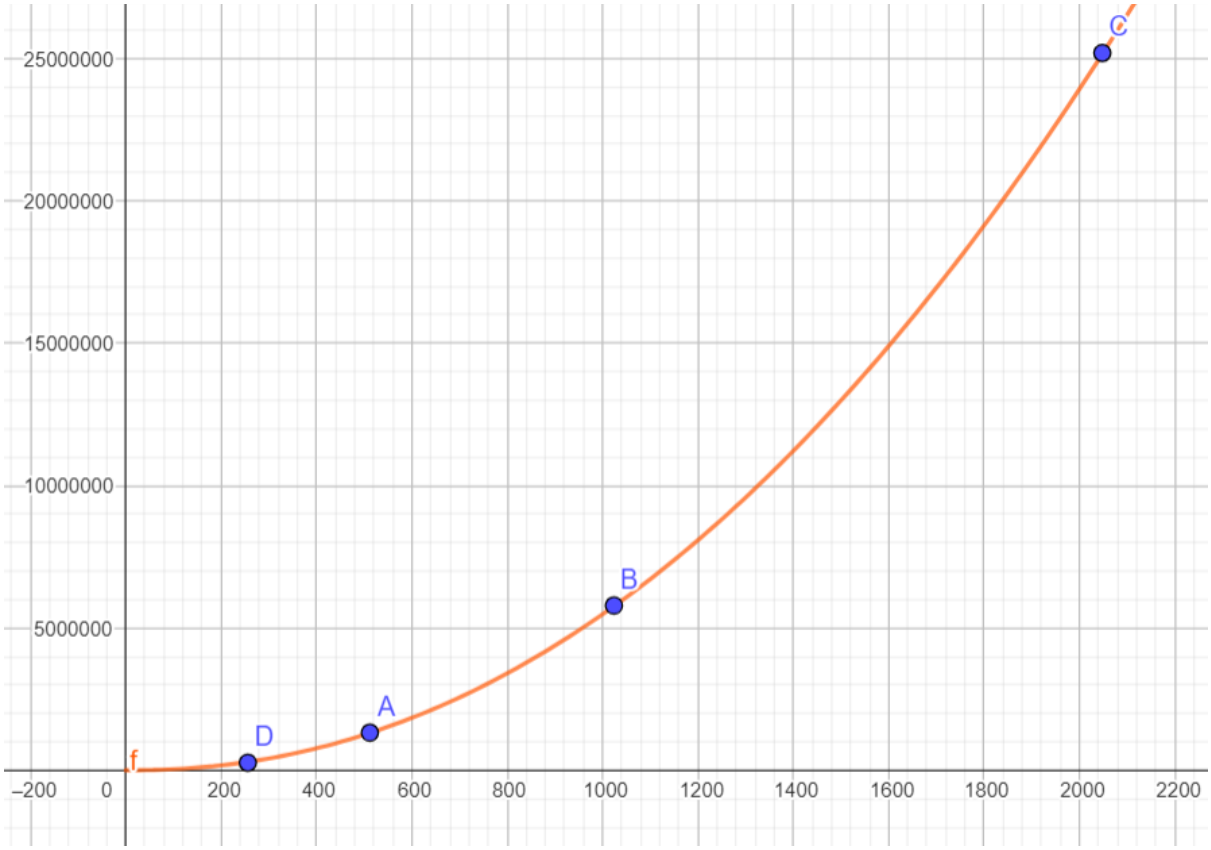
Koden er blevet analyseret. Det yderste loop kører x gange. Det næste loop når x er under 256 kører x^2 gange. Det sidste loop kører log(x) gange fordi der divideres med 2 pr. iteration. Det vil altså sige tidskompleksiteten er omkring $O(N^3 \log N)$. Forsøger er her lavet for x = 8, 16, 64, 128. Med en lille skalar justering på 2, passer denne kompleksitet meget godt på dataen. Dermed kan det konkluderes at når x er under 256 bliver store O tidskompleksiteten:

$$O(N^3 \log N)$$



Det yderste loop kører stadig x gange. Det næste loop når x er større end 256 bliver kaldt med x/2 og kører derved x/2 gange. Antages det at x stadig er større end 256 bliver sidste loop kaldt med 2*x og kører derved log(2*x) gange. Altså er et gæt på tidskompleksiteten $O\left(\frac{N^2}{2}\log 2\,N\right)$. Forsøg med x = 256, 512, 1024, 2048 bliver holdt oppe imod dette gæt og det ser ud til at passe meget godt på dataen. Derved konkluderes det altså at tidskompleksiteten for metoden når x er større end 256 er:

$$O\left(N^2 \log N\right)$$



Opgave 4 (10 %)

Nedenfor er vist en række elementer, som er indsat i hopscotch tabellen længere nede. Elementerne er indsat i alfabetisk rækkefølge, altså først indsættes A, herefter B, efterfulgt af C etc.

Opgaven går ud på at finde to fejl i hopscotch-tabellen. Der kan forekomme to typer af fejl: værdien står i det forkerte indeks, eller hoppen er opdateret forkert i forhold til værdiernes placering.

Der kan ses bort fra loadfaktor og rehashing.

Værdi	Hasher til indeks
A:	51
B:	54
C:	49
D:	52
E:	54
F:	53
G:	55
H:	49

Indeks	Værdi	Hop
47		0000
48		0000
49	C	1100
50	H	0001
51	A	1000
52	D	1000
53	F	1000
54	B	1100
55	E	0100
56	G	1000
57		0000
58		0000

Alle værdierne er blevet placeret på de korrekte indekser, men hoppen er blevet opdateret forkert på indeks 50 og indeks 56. På indeks 50 står der 0001, hvilket fortæller at F skulle hashe til 50 men dette er ikke tilfælde, der burde i stå 0000, fordi ingen værdierne i blokken hasher til 50. På indeks 56 står der 1000, hvilket betyder at G burde hashe til 56, det er ikke rigtigt, der burde stå 0000 fordi ingen værdier hasher til 56.

Opgave 7 (10 %)

Nedenfor ses et binært søgetræ.

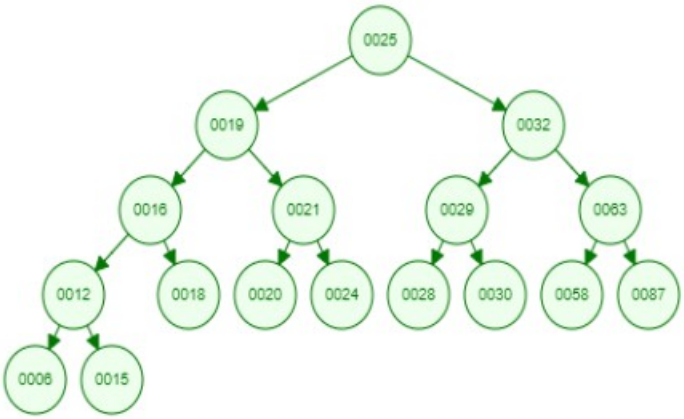
Hvilke karakteristika kendetegner dette træ?

Hvad er træets højde og internal path length?

Tegn træet efter at du har udført følgende operationer i den angivne rækkefølge:

Operation 1: delete 29
Operation 2: insert 22
Operation 3: delete 12
Operation 4: insert 60
Operation 5: delete 21

Har træets karakteristika ændret sig?



En karakteristika ved dette binære søgetræ er at det er komplet. Det er det fordi alle niveauer er fyldt ud på nær det sidste, hvor der dog bliver fyldt op fra højre.

Træets højde er 4. Internal path length er givet som summen af dybderne for samtlige noder for højre og venstre subtræ set fra roden:

$$D(N) = D(i) + D(N - i - 1) + N - 1$$

$$D(i) = 16$$

$$D(N - i - 1) = 10$$

$$N - 1 = 16$$

$$D(N) = 16 + 10 + 16$$

$$= 42$$