Algoritmer og Datastrukturer

<https://www.youtube.com/playlist?list=PL_c9BZzLwBRLpDEpYRFXKBN-2ZCsAx0ps>

Innhold

[1 Datastrukturer 2](#_Toc87362086)

[1.1 String 2](#_Toc87362087)

[1.2 List 3](#_Toc87362088)

[1.3 Linked List 5](#_Toc87362089)

[1.4 Double linked list 7](#_Toc87362090)

[1.5 Hash Tables og Dictionaries 9](#_Toc87362091)

[1.5.1 Python Dictionary 9](#_Toc87362092)

[1.5.2 C# 10](#_Toc87362093)

[1.6 Binære trær 11](#_Toc87362094)

[1.7 Stack og Queue 12](#_Toc87362095)

[2 Algoritmer 13](#_Toc87362096)

[2.1 Binary Search 13](#_Toc87362097)

[2.1.1 Iterative 13](#_Toc87362098)

[2.1.2 Recursive 14](#_Toc87362099)

[2.2 Binære trær lag for lag 15](#_Toc87362100)

[2.2.1 Iterative 15](#_Toc87362101)

[2.2.2 Recursive 16](#_Toc87362102)

[2.3 Binære trær dybde først 17](#_Toc87362103)

[2.3.1 Iterative 17](#_Toc87362104)

[2.3.2 Recursive 17](#_Toc87362105)

[2.4 Inorder, Preorder, Postorder 18](#_Toc87362106)

[2.4.1 inorder iterative 18](#_Toc87362107)

[2.4.2 inorder recursive 18](#_Toc87362108)

[2.4.3 preorder iterative 19](#_Toc87362109)

[2.4.4 preorder recursive 19](#_Toc87362110)

[2.4.5 postorder iterative 20](#_Toc87362111)

[2.4.6 postorder recursive 20](#_Toc87362112)

[class Node: 20](#_Toc87362113)

[1 Insertion sort 21](#_Toc87362114)

[1.1 Insertion sort iterative 21](#_Toc87362115)

[1.2 Insertion sort recursive 21](#_Toc87362116)

[2.5 Selection sort 22](#_Toc87362117)

[2.5.1 Selection sort iterative 22](#_Toc87362118)

[2.5.2 Selection sort recursive 22](#_Toc87362119)

[2.6 Merge sort 23](#_Toc87362120)

[2.7 Quicksort 25](#_Toc87362121)

[2.8 Heapsort 27](#_Toc87362122)

[3 Oppgave 29](#_Toc87362123)

# Datastrukturer

Datastruktur er hvordan data lagres. Dere har på VG1 lært om String, list, tuple, set og dictionary. Vi skal se på list litt dypere men starter med string siden det fungerer likt list på flere måter. Etter det skal vi se på Linked List, stacks og queues. Etter det kommer Hash Tables og Dictionaries og til slutt litt enkelt om Trestruktur

## String

Først og fremst tenker du nok på str eller string som en datatype. Og det er jo riktig. Men det er igså den enkleste datastrukturen vi har. En string kan adresseres fra 0 til lengde av string – 1.



String brukes til å lagre Unicode-tegn <https://en.wikipedia.org/wiki/List_of_Unicode_characters>.

En string kan endres i C# og Javascript, men ikke i Python .

Eksempel:

**Python**

melding = "Hallo Verden!"

print("String inneholder: ",melding)

teller = 1

for bokstav in melding:

print("Plassholder",teller,"i string er:",bokstav)

teller+=1

**C#**

using System;

namespace StringEksempel

{

class Program

{

static void Main(string[] args)

{

const string melding = "Hallo Verden!";

Console.WriteLine("String inneholder: "+melding);

int teller = 1;

foreach (char bokstav in melding)

{

Console.WriteLine("Plassholder "+teller+" i string er: "+bokstav);

teller++;

}

}

}

}

## List

List er en datastruktur som lagrer en rekke (sekvens) med verdier. Verdiene indekseres som i en string hver alle verdiene får en indeks

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| The | Ultimate | Answer | to | Life | , | The | Universe | and | Everything | is | 42 |

**Python**

<https://docs.python.org/3/tutorial/datastructures.html#more-on-lists>

<https://docs.python.org/3/library/array.html?highlight=array#module-array>

**Eksempel 1**

minListe = ["1IMA", "1IMB", "1IMC", "2ITA", "2ITB"]

print(minListe)

for verdi in minListe:

print(verdi)

**Eksempel 2**

# Viser arrays i Python sammenlignet med list.

# Array må typedeklareres, og kan bare inneholde en datatype

import array

minArrayInt = array.array('i', [6, 4, 9])

minArrayFloat = array.array('d', [8.5, 4.2, 5.3])

minListe = [1, 1.2, "Hallo Verden!"]

print ("Integer array : ", end =" ")

for i in range(len(minArrayInt)):

print (minArrayInt[i], end =" ")

print()

print ("Float array : ", end =" ")

for i in range(len(minArrayFloat)):

print (minArrayFloat[i], end =" ")

print()

print ("Liste : ", end =" ")

for i in range(len(minListe)):

print (minListe[i], end =" ")

print()

minArrayInt.insert(0,0)

minArrayInt.append(4)

print(minArrayInt)

minArrayFloat.insert(0,9.9)

minArrayFloat.append(4.5)

print(minArrayFloat)

minListe.insert(0,42)

minListe.append("Hallo igjen!")

print(minListe)

# minArrayFloat.insert(0,0)

# minArrayFloat.append(4)

# print(minArrayInt)

# minArrayInt.insert(0,9.9)

# minArrayInt.append(4.5)

# print(minArrayFloat)

# minArrayInt.insert(0,"Hallo igjen!")

# minArrayFloat.insert(0,"Hallo igjen!")

# minArrayInt.append("Hallo igjen!")

# minArrayFloat.append("Hallo igjen!")

# print(minArrayInt)

# print(minArrayFloat)

**C#**

Referanse

<https://www.tutorialsteacher.com/csharp/csharp-list>

<https://www.tutorialsteacher.com/csharp/array-csharp>

<https://www.educba.com/c-sharp-list-vs-array/>

Video

<https://www.youtube.com/watch?v=LBZqGtgxMjs>

using System;

using System.Collections.Generic;

class ListeEksempel

{

# Viser arrays i C# sammenlignet med list.

# All kode i C# må typedeklareres, og kan bare inneholde en datatype

# Arrays i C# har fast lengde dom må defineres, mens lister er dynamiske og kan vokse.

public static void Main(String[] args)

{

List<String> navneliste = new();

navneliste.Add("Per");

navneliste.Insert(0, "Anne");

navneliste.Insert(1, "Ole");

// navneliste.Add(42);

foreach (string navn in navneliste)

{

Console.WriteLine("Navn: "+ navn);

}

Console.WriteLine("\nAntall plasser i listen er: " + navneliste.Capacity);

Console.WriteLine("Antall elementer i listeb: " + navneliste.Count);

navneliste.Add("Lars");

navneliste.Add("Kari");

Console.WriteLine("Antall plasser i listen er: " + navneliste.Capacity);

Console.WriteLine("Antall elementer i listen er: " + navneliste.Count);

int[] tallArray = new int[5];

string[] navnArray = new string[3];

tallArray.SetValue(42, 0);

tallArray[3] = 1814;

navnArray.SetValue("Lars", 0);

navnArray[2] = "Anne";

Console.WriteLine("\n");

foreach (int tall in tallArray)

{

Console.WriteLine(tall);

}

foreach (string navn in navnArray)

{

Console.WriteLine(navn);

}

}

}

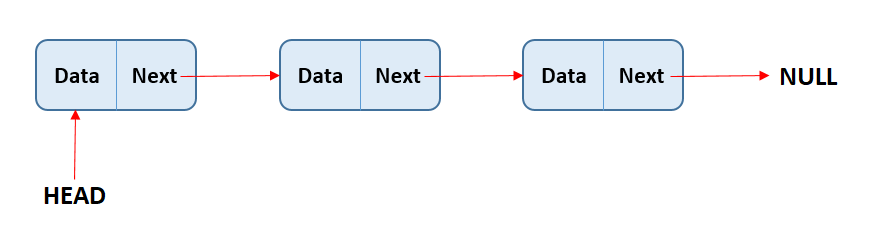
## Linked List

<https://www.geeksforgeeks.org/data-structures/linked-list/>

<https://www.youtube.com/watch?v=dMy2hq9OUMc&list=PL_c9BZzLwBRLpDEpYRFXKBN-2ZCsAx0ps&index=10>

<https://www.youtube.com/watch?v=HDSszyCM7Tk&list=PL_c9BZzLwBRLpDEpYRFXKBN-2ZCsAx0ps&index=11>

<https://www.youtube.com/watch?v=sVxBVvlnJsM>



**Python**

class Node: # Class Node

def \_\_init\_\_(self,data): # Constructor

self.data = data # Setter data i objekt til parameter data

self.next = None # Setter next i objekt som skal peke på neste objekt

# til nullverdi (None)

class SingleLinkedList:

def \_\_init\_\_(self):

self.head = None # Head (første Node) i Linked list

self.tail = None # Tail (siste Node) i Linked list

def addNode(self, data): # Metode for å legge til node i Linked List

newNode = Node(data) # Lage en ny Node

if(self.head == None): # Sjekke om listen er tom

self.head = newNode # Hvis listen er tom vil noden være første

self.tail = newNode # og siste node i Linked list

else:

self.tail.next = newNode # Ny Node blir satt til siste node (Tail) sin Next

self.tail = newNode # Ny Node blir satt til siste node (Tail)

def display(self): # Vise alle noder på skjerm

gjeldende = self.head # Variabel gjeldende blir satt til første node (Head)

if(self.head == None): # Hvis første node (Head) er tom

print("List is empty") # Skriv ut at listen er tom

return # Avslutt Metode

print("Noder i linked list: ") # Skriv overskrift for nodeutskrift

while(gjeldende != None): # Så lenge gjeldende node har en verdi (ikke None)

print(gjeldende.data) # Skriv ut data til gjeldende node

gjeldende = gjeldende.next # Bytt gjeldende til gjeldende neste

minLinkedList = SingleLinkedList() # Lage Linked List basert på class SingleLinkedList

minLinkedList.addNode(1); # Legge til node med metode addNode. Sender med data

minLinkedList.addNode(2); # Legge til node med metode addNode. Sender med data

minLinkedList.addNode(3); # Legge til node med metode addNode. Sender med data

minLinkedList.addNode(4); # Legge til node med metode addNode. Sender med data

minLinkedList.display(); # Kaller metode display som skriver ut alle node data

C#

using System;

namespace LinkedListEksempel

{

public class Node // Class Node

{

public Node next; // Parameter next type Node

public object data; // Parameter data type object

}

public class LinkedList // Class LinkedList

{

private Node head; // Dummy head for å ha en start på Linked List

public void Display() // Metode Display

{

Node gjeldende = head; // Setter Node gjeldende til head

while (gjeldende != null) // Så lenge gjeldende ikke er null

{

Console.WriteLine(gjeldende.data); // Send data i gjeldende til console

gjeldende = gjeldende.next; // Gjeldende er lik noden gjeldende.next peker på

}

}

public void AddNodeFirst(object data) // Metode for å legge til Node foran de andre nodene

{

Node nyNode = new Node(); // Lager ny node nyNode

nyNode.data = data; // Legger inn data fra metode kall i nyNode.data

nyNode.next = head; // Legger inn peker i nyNode.next til det som er head Noden

head = nyNode; // Plasserer ny node først ved å si at nyNode er head Node

}

public void AddNodeLast(object data) // Metode for å legge til Node etter de andre nodene

{

if (head == null) // Hvis hode er null

{

head = new Node // Constructor: Lag nytt head Node (for å ha en start)

{

data = data, // Legger inn data fra metode kall i head.data

next = null // Legger inn null i head.next

};

}

else

{

Node nyNode = new Node() // Lager ny node nyNode

{

data = data // Legger inn data fra metode kall i nyNode.data

};

Node gjeldende = head; // Setter Node gjeldende til head

while (gjeldende.next != null) // Så lenge gjeldende.next ikke er null

{

gjeldende = gjeldende.next; // Gjeldende er lik noden gjeldende.next peker på

}

gjeldende.next = nyNode; // På enden av listen sett nect i siste noden til nyNode

}

}

}

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Legge til Node foran de andre nodene i Linked List:");

LinkedList minListe1 = new LinkedList(); // Lager ny LinkedList minListe1

minListe1.AddNodeFirst(1); // Legger til node med metode addNodeFirst. Sender med data

minListe1.AddNodeFirst(2); // Legger til node med metode addNodeFirst. Sender med data

minListe1.AddNodeFirst(3); // Legger til node med metode addNodeFirst. Sender med data

minListe1.AddNodeFirst(4); // Legger til node med metode addNodeFirst. Sender med data

minListe1.Display(); // Kaller metode for å sende LinkedList til console

Console.WriteLine();

Console.WriteLine("Legge til Node etter de andre nodene i Linked List");

LinkedList minListe2 = new LinkedList(); // Lager ny LinkedList minListe1

minListe2.AddNodeLast("En"); // Legger til node med metode addNodeLast. Sender med data

minListe2.AddNodeLast("To"); // Legger til node med metode addNodeLast. Sender med data

minListe2.AddNodeLast("Tre"); // Legger til node med metode addNodeLast. Sender med data

minListe2.AddNodeLast("Fire"); // Legger til node med metode addNodeLast. Sender med data

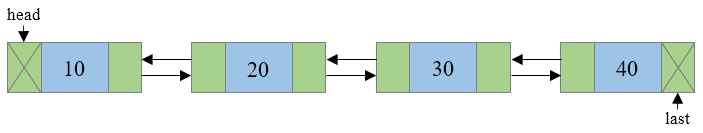
minListe2.Display(); // Kaller metode for å sende LinkedList til console

}

}

}

## Double linked list



**Python**

class Node: # Class Node

def \_\_init\_\_(self,data): # Constructor

self.data = data # Setter data i objekt til parameter data

self.next = None # Setter next i objekt som skal peke på neste objekt til nullverdi (None)

self.prev = None # Setter prev i objekt som skal peke på forrige objekt til nullverdi (None)

class DoubleLinkedList:

def \_\_init\_\_(self):

self.head = None # Head (første Node) i Linked list

def addNode(self, data): # Metode for å legge til node først i Linked List

nyNode = Node(data) # Lage en ny Node

nyNode.next = self.head; # Setter peker fra nyNode.next til self.head

if(self.head != None): # Sjekke om listen er tom

self.head.prev = nyNode # Hvis listen er tom vil noden være første

self.head = nyNode # og siste node i Linked list

def appendNode(self, data): # Metode for å legge til node først i Linked List

nyNode = Node(data) # Lage en ny Node

nyNode.next = None # Setter nyNode.next til none

if self.head is None: # Sjekke om listen er tom

nyNode.prev = None # Hvis listen er tom sett nyNode.prev til None

self.head = nyNode # Hvis listen er tom sett self.head(første node) til nyNode

return # Hvis listen er tom Avslutt

last = self.head # Sett last til self.head(første)

while (last.next is not None): # Så lenge last.next ikke er None

last = last.next # last settes til last.next

last.next = nyNode # Når last.nest = null sett last next til nuNode

nyNode.prev = last # Sett nyNode.prev til last

return # Avlsutt

def display(self, node): # Vise alle noder på skjerm

if(self.head == None): # Hvis første node (Head) er tom

print("List is empty") # Skriv ut at listen er tom

return # Avslutt Metode

print("Noder i linked list: ") # Skriv overskrift for nodeutskrift

while(node != None): # Så lenge gjeldende node har en verdi (ikke None)

print(node.data) # Skriv ut data til gjeldende node

last = node # Siste node settes til node

node=node.next # Bytt gjeldende node til gjeldende neste node

minLinkedList = DoubleLinkedList() # Lage Linked List basert på class SingleLinkedList

minLinkedList.addNode(1); # Legge til node med metode addNode. Sender med data

minLinkedList.addNode(2); # Legge til node med metode addNode. Sender med data

minLinkedList.addNode(3); # Legge til node med metode addNode. Sender med data

minLinkedList.addNode(4); # Legge til node med metode addNode. Sender med data

minLinkedList.appendNode(10); # Legge til node med metode addNode. Sender med data

minLinkedList.appendNode(20); # Legge til node med metode addNode. Sender med data

minLinkedList.appendNode(30); # Legge til node med metode addNode. Sender med data

minLinkedList.appendNode(40); # Legge til node med metode addNode. Sender med data

minLinkedList.display(minLinkedList.head); # Kaller metode display som skriver ut alle node data

**C#**

using System;

namespace DoubleLinkedListEksempel

{

public class Node // Class Node

{

public Node next; // Parameter next type Node

public Node prev; // Parameter prev type Node

public object data; // Parameter data type object

}

public class LinkedList // Class LinkedList

{

private Node head; // Dummy head for å ha en start på Linked List

public void Display() // Metode Display

{

Node gjeldende = head; // Setter Node gjeldende til head

while (gjeldende != null) // Så lenge gjeldende ikke er null

{

Console.WriteLine(gjeldende.data); // Send data i gjeldende til console

gjeldende = gjeldende.next; // Gjeldende er lik noden gjeldende.next peker på

}

}

public void AddNodeFirst(object data) // Metode for å legge til Node foran de andre nodene

{

Node nyNode = new Node(); // Lager ny node nyNode

nyNode.data = data; // Legger inn data fra metode kall i nyNode.data

nyNode.next = head; // Legger inn peker i nyNode.next til det som er head Noden

nyNode.prev = null; // legger inn null i nyNode.prev siden nyNids skal bli head

if (head != null) // Hvis head ikke er null

head.prev = nyNode; // legger inn peker i head.prev til nyNode som skal bli ny head

head = nyNode; // Plasserer ny node først ved å si at nyNode er head Node

}

public void AddNodeLast(object data) // Metode for å legge til Node etter de andre nodene

{

Node nyNode = new Node() // Lager ny node nyNode

{

data = data // Legger inn data fra metode kall i nyNode.data

};

Node gjeldende = head; // Klargjør for søk ved å kopiere head til en ny node gjeldende

nyNode.next = null; // Legger inn peker i nyNode.next til null

if (head == null) // Hvis hode er null fordi listen er tom

{

nyNode.prev = null; // legger inn null i nyNode.prev

head = nyNode; // head = nyNode

return; // Avslutt

}

while (gjeldende.next != null) // Så lenge gjeldende ikke er null (går til slutten av listen)

gjeldende = gjeldende.next; // gjeldende node settes til gjeldende.next

gjeldende.next = nyNode; // Når slutten på listen er nått settes siste nodes next til nyNode

nyNode.prev = gjeldende; // Og nyNode.prev settes til den gamle sistenoden

}

}

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Legge til Node foran de andre nodene i Linked List:");

LinkedList minListe1 = new LinkedList(); // Lager ny LinkedList minListe1

minListe1.AddNodeFirst(1); // Legger til node med metode addNodeFirst. Sender med data

minListe1.AddNodeFirst(2); // Legger til node med metode addNodeFirst. Sender med data

minListe1.AddNodeFirst(3); // Legger til node med metode addNodeFirst. Sender med data

minListe1.AddNodeFirst(4); // Legger til node med metode addNodeFirst. Sender med data

minListe1.Display(); // Kaller metode for å sende LinkedList til console

Console.WriteLine();

Console.WriteLine("Legge til Node etter de andre nodene i Linked List");

LinkedList minListe2 = new LinkedList(); // Lager ny LinkedList minListe2

minListe2.AddNodeLast("En"); // Legger til node med metode addNodeLast. Sender med data

minListe2.AddNodeLast("To"); // Legger til node med metode addNodeLast. Sender med data

minListe2.AddNodeLast("Tre"); // Legger til node med metode addNodeLast. Sender med data

minListe2.AddNodeLast("Fire"); // Legger til node med metode addNodeLast. Sender med data

minListe2.Display(); // Kaller metode for å sende LinkedList til console

}

}

}

## Hash Tables og Dictionaries

Data samles i par. Nøkkel må være unik. Data adresseres via nøkkelfelt.

Et bilde som inneholder tekst, klokke

Automatisk generert beskrivelse

Eksempel:

**Python**

person = {"Navn":"Per Spellmann"}

print(person["Navn"]) # person er her dictionary, mens Navn er key

**C#**

Hashtable person = new()

{

{"Navn", "Per Spellmann"},

};

Console.WriteLine(person["Navn"]);

### Python Dictionary

**Eksempel 1:**

minDict = {"Navn":"","Adresse":"","Postnummer":"", "Poststed":"","Alder":0}

minDict["Navn"] = "Per Spellmann"

minDict["Adresse"] = "Lia 12"

minDict["Postnummer"] = "2000"

minDict["Poststed"] = "Lillestrøm"

minDict["Alder"] = 42

minDict["Kompetanse"] = "Python", "C#", "Database"

print("Navn : ",minDict["Navn"])

print("Adresse : ",minDict["Adresse"])

print("Postnummer : ",minDict["Postnummer"])

print("Poststed : ",minDict["Poststed"])

print("Alder : ",minDict["Alder"])

teller = 1

for komp in minDict["Kompetanse"]:

print(f"Kompeatanse {teller} : ",komp)

teller +=1

**Eksempel 2:**

elever = {

"Elev1": {

"Navn":"Per Spellmann",

"Adresse":"Lia 12",

"Postnummer":"2000",

"Poststed":"Lillestrøm",

"Alder":42,

"Kompetanse": ["Python", "C#", "Database"]}}

elever["Elev2"] = {}

elever["Elev2"]["Navn"] = "Kari Spellmann"

elever["Elev2"]["Adresse"] = "Lia 21"

elever["Elev2"]["Postnummer"] = "2013"

elever["Elev2"]["Poststed"] = "Skjetten"

elever["Elev2"]["Alder"] = "35"

elever["Elev2"]["Kompetanse"] = ["SQl","XML","JSON"]

for elev in elever:

print("\nNavn : ",elever[elev]["Navn"])

print("Adresse : ",elever[elev]["Adresse"])

print("Postnummer : ",elever[elev]["Postnummer"])

print("Poststed : ",elever[elev]["Poststed"])

print("Alder : ",elever[elev]["Alder"])

teller = 1

for komp in elever[elev]["Kompetanse"]:

print(f"Kompeatanse {teller} : ",komp)

teller +=1

### C#

**Hashtable**

using System;

using System.Collections;

class Eksempel

{

public static void Main()

{

Hashtable hashTableEksempel = new()

{

{ 1, "Hallo Verden!" },

{ 2, 42 },

{ 3, 3.14 },

{ 4, null },

};

foreach (var verdi in hashTableEksempel.Keys)

{

Console.WriteLine("Key: {0} Verdi: {1}", verdi, hashTableEksempel[verdi]);

}

Hashtable minHashTable = new();

minHashTable.Add("Navn", "Per Spellmenn");

minHashTable.Add("Adresse", "Lia 12");

minHashTable.Add("Postnummer", "2000");

minHashTable.Add("Poststed", "Lillestrøm");

minHashTable.Add("Kompetanse", "SQL, XML, JSON");

Console.WriteLine();

foreach (DictionaryEntry elev in minHashTable)

{

Console.WriteLine("{0} : {1} ", elev.Key, elev.Value);

}

}

}

**Dictionary**

using System;

using System.Collections.Generic;

class DictionaryEksempel

{

public static void Main()

{

Dictionary<string, string> Elever = new()

{

{ "Navn", "Per Spellmann" },

{ "Adresse", "Lia 12" },

{ "Postnr", "2000" },

{ "Poststed", "Lillestrøm" },

{ "Kompetanse", "SQL, XML ,JSON"}

};

foreach (KeyValuePair<string, string> elev in Elever)

{

Console.WriteLine("Key = {0}, Value = {1}", elev.Key, elev.Value);

}

}

}

## Binære trær

# https://pythonbasics.org/constructor/

# https://www.youtube.com/watch?v=KzZhAqu3s4M&t=3s (se hele videoen)

# https://www.digitalocean.com/community/tutorials/understanding-class-and-instance-variables-in-python-3

class Node: # Class Node

def \_\_init\_\_(self, data): # Constructor for Node

self.leftnode = None # Setter instance variabel leftnode til ingen verdi

self.rightnode = None # Setter instance variabel rightnode til ingen verdi

self.value = data # Setter instance variabel value til verdien mottatt i parameteren data

class Tree: # Class Tree

def \_\_init\_\_(self): # Constructor for Tree

self.root = None # Lage et tomt tree ( starten på treet )

def printTree(self): # Class Method printTree. Brukes til å sjekke om strukturen er tom. parameter inn er et objekt

if(self.root != None): # Hvis treet (self root) ikke er tomt

self.\_printTree(self.root) # Kall Method \_printTree og sender med objekt.

def \_printTree(self, node): # Class Method \_printTree Skriver ut alle noder i tre. Parameter inn er er self og et objekt

if(node != None): # Hvis parameter node ikke er None

self.\_printTree(node.leftnode) # Kall Method \_printTree og sender med objekt node.leftnode.

print(str(node.value) + ' ') # Skriv ut string variant av node.value og en space.

self.\_printTree(node.rightnode) # Kall Method \_printTree og sender med objekt node.rightnode.

def add(self, data): # Class Method add. Brukes til å sjekke om strukturen er tom, og eventuelt lage tre. parameter inn er self og data

if(self.root == None): # Hvis treet (self root) er tomt

self.root = Node(data) # Lag node med root.value lik data fra parameter

else: # Hvis treet (self root) ikke er tomt

self.\_add(data, self.root) # Kall Method self.\_add med parameter data og peker til root i tre

def \_add(self, data, node): # Class Method \_add. Btukes til å legge til noder i tre under root. Parameter inn er self, data til node og et objekt|

if(data < node.value): # Hvis parameter data er mindre enn value i noden.

if(node.leftnode != None): # Hvis node.leftnode ikke er None

self.\_add(data, node.leftnode) # Kall Method \_add med parameter data og node.leftnode

else: # Hvis node.leftnode er None

node.leftnode = Node(data) # Lag ny Node og legg inn som node.leftnode

else: # Hvis parameter data er større enn value i noden.

if(node.rightnode != None): # Hvis node.rightnode ikke er None

self.\_add(data, node.rightnode) # Kall Method \_add med parameter data og node.rightnode

else: # Hvis node.rightnode er None

node.rightnode = Node(data) # Lag ny Node og legg inn som node.rightnode

if \_\_name\_\_ == '\_\_main\_\_': # Hvis modul er main modul https://docs.python.org/3/library/\_\_main\_\_.html

tree = Tree() # Lager et Tree men navn tree

tall=1 # Lokal variabel tall settes til 1

while tall !=0: # Så lenge variabel tall ikke er lik 0

tall = int(input("Tast inn et tall, avslutt med 0: ")) # Be bruker taste inn et tall og avslutte med 0. Legg input i varabel tall

if tall != 0: # Hvis tall ikke er 0

tree.add(tall) # Kall metode add for å legge ny node i treet

tree.printTree() # Kall metode tree.printTree for å skrive ut tre

Pyton binarytree Modul

# https://binarytree.readthedocs.io/en/latest/

# pip install binarytree

from binarytree import build

if \_\_name\_\_ == '\_\_main\_\_': # Hvis modul er main modul

tall=1 # Lokal variabel tall settes til 1

liste=[] # Lokal variabel list type list opprettes tom

while tall !=0: # Så lenge variabel tall ikke er lik 0

tall = int(input("Tast inn et tall, avslutt med 0: ")) # Be bruker taste inn et tall og avslutte med 0. Legg input i varabel tall

if tall != 0: # Hvis tall ikke er 0

liste.append(tall) # Legger verdi tall til i liste

tree = build(liste) # Bygger tree med metode build fra usortert liste

print(tree) # Skriver ut tree

sortert = sorted(liste) # Sorterer liste

sortedTree = build(sortert) # Bygger tree med metode build fra sortert liste

print(sortedTree) # Skriver ut sortedTree

## Stack og Queue

<https://www.youtube.com/watch?v=IxQHWt0GpsU>

<https://www.youtube.com/watch?v=IITnvmnfi_Y>

# Stack fungerer etter prinsippet «Sist inn, først ut

minStack = ["1IMA", "1IMB", "1IMC"]

minStack.append("2ITA")

minStack.append("2ITB")

minStack.append("1STA")

minStack.append("1STB")

print(minStack)

minStack.pop()

minStack.pop()

print(minStack)

for verdi in minStack:

print(verdi)

# Stack med deque

from collections import deque

minQueue = deque(["1IMA", "1IMB", "1IMC"])

print(minQueue,"\n")

minQueue.append("2ITA")

minQueue.append("2ITB")

print(minQueue,"\n")

minQueue.pop()

minQueue.pop()

for verdi in minQueue:

print(verdi)

# Queue fungerer etter prinsippet «Først inn, først ut

minStack = ["1IMA", "1IMB", "1IMC"]

minStack.append("2ITA")

minStack.append("2ITB")

minStack.append("1STA")

minStack.append("1STB")

print(minStack)

minStack.pop(0)

minStack.pop(0)

print(minStack)

for verdi in minStack:

print(verdi)

# Queue med deque

from collections import deque

minQueue = deque(["1IMA", "1IMB", "1IMC"])

print(minQueue,"\n")

minQueue.append("2ITA")

minQueue.append("2ITB")

print(minQueue,"\n")

minQueue.popleft()

minQueue.popleft()

for verdi in minQueue:

print(verdi)

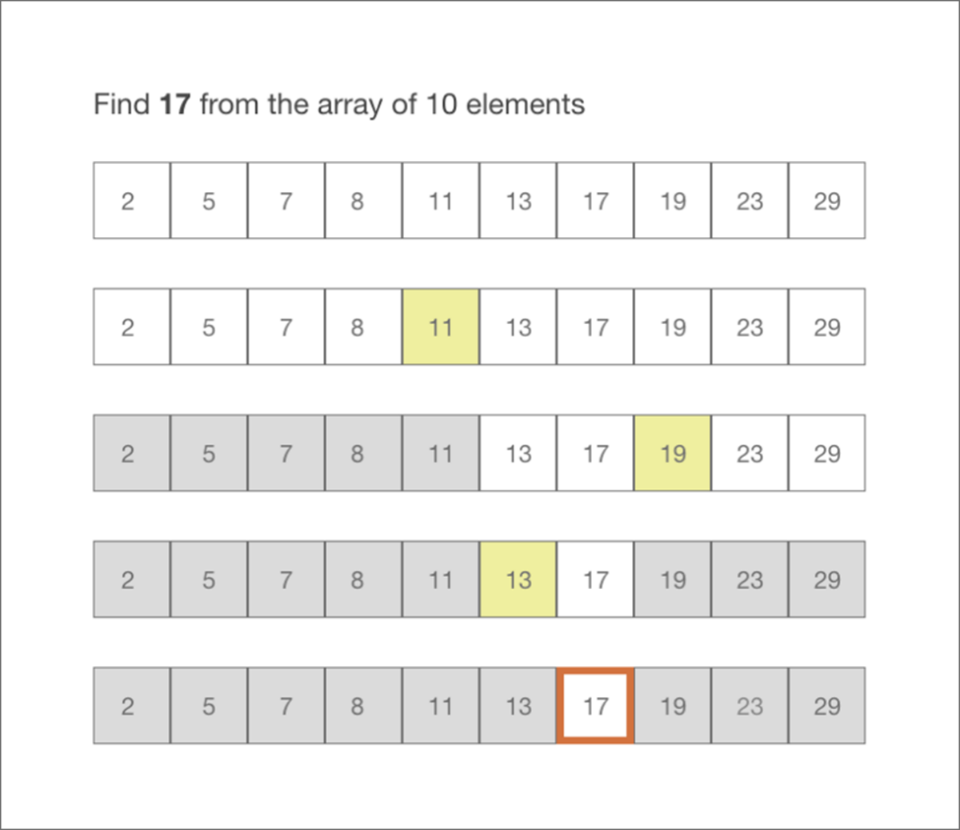
# Algoritmer

## Binary Search

**if \_\_name\_\_ == '\_\_main\_\_':**

<https://www.youtube.com/watch?v=sugvnHA7ElY>

**Binary search videeo**  
<https://www.youtube.com/watch?v=DE-ye0t0oxE>



### Iterative

def binarySearch(tallListe, maalVerdi): # Metide BinarySearch variabler er liste og verdi vi skal finne

venstre = 0 # Variabel venstre stees til index 0

hoyre = len(tallListe) - 1 # Variabel hoyre settes til siste index

while venstre <= hoyre: # Så lenge venstre er mindre eller lik hoyre

midt = (venstre + hoyre) // 2 # Finner midterste verdi i liste

if maalVerdi == tallListe[midt]: # Hvis verdi vi skal finne er midterste verdi

return midt # Returner index midterste

elif maalVerdi < tallListe[midt]: # Hvis ikke er verdi vi skal finne mindre enn midt

hoyre = midt - 1 # Variabel hoyre settes til index midten - 1

else: # Verdi vi skal finne er større enn midt

venstre = midt + 1 # Variabel hoyre settes til index midten + 1

return -1 # Verdi finnes ikke returner -1

if \_\_name\_\_ == '\_\_main\_\_': # Hvis modul er main modul https://docs.python.org/3/library/\_\_main\_\_.html

sortertListe = [2, 5, 6, 8, 9, 10] # Variabel sortertListe settes til en liste

verdi = 11 # Variabel verdi vi skal søke etter får et innhold

# kaller metode med liste og verdi det skal søkes etter som parametere

indexIsortertListe = binarySearch(sortertListe, verdi)

# Hvis metode returnerer noe annet enn -1 så er det indeks plass i sortertListe

# Da skrives verdien sin plass i sortertListe ut

# Hvis metode returnerer -1 skrives det ur at verdien ikke finnes

if indexIsortertListe != -1:

print(f"Verdien finne i sortert list på index {indexIsortertListe}")

else:

print(f"{verdi} finnes ikke i sortert liste")

**C#**

using System;

class BinarySearchEksempel

{

static int BinarySearch(int[] sortertListe, int tall)

{

int venstre = 0;

int hoyre = sortertListe.Length - 1;

while (venstre <= hoyre)

{

int midt = venstre + (hoyre - venstre) / 2;

if (sortertListe[midt] == tall)

return midt;

if (sortertListe[midt] < tall)

venstre = midt + 1;

else

hoyre = midt - 1;

}

return -1;

}

public static void Main()

{

int[] sortertListe = { 2, 3, 4, 10, 40 };

int verdi = 10;

int plassIsortertListe = BinarySearch(sortertListe, verdi);

if (plassIsortertListe == -1)

Console.WriteLine(verdi+" finnes ikke i sortert liste");

else

Console.WriteLine(verdi + " finnes i sortert liste på plass " + plassIsortertListe);

}

}

### Recursive

**Python**

def binarySearch(tallmengde, venstre, hoyre, tall):

if venstre > hoyre:

return -1

mid = (venstre + hoyre) // 2

if tall == tallmengde[mid]:

return mid

elif tall < tallmengde[mid]:

return binarySearch(tallmengde, venstre, mid - 1, tall)

else:

return binarySearch(tallmengde, mid + 1, hoyre, tall)

if \_\_name\_\_ == '\_\_main\_\_':

tallmengde = [2, 5, 6, 8, 9, 10]

tall = 5

(venstre, hoyre) = (0, len(tallmengde) - 1)

resultat = binarySearch(tallmengde, venstre, hoyre, tall)

if resultat != -1:

print('Tall finnes i tallmengde', resultat)

else:

print('Tall finnes ikke i tallmengde')

**C#**

using System;

class BinarySearchEksempel

{

static int BinarySearch(int[] sortertListe, int venstre, int hoyre, int tall)

{

if (hoyre >= venstre)

{

int midten = venstre + (hoyre - venstre) / 2;

if (sortertListe[midten] == tall)

return midten;

if (sortertListe[midten] > tall)

return BinarySearch(sortertListe, venstre, midten - 1, tall);

return BinarySearch(sortertListe, midten + 1, hoyre, tall);

}

return -1;

}

public static void Main()

{

int[] sortertListe = { 2, 3, 4, 10, 40 };

int verdi = 10;

int plassIsortertListe = BinarySearch(sortertListe, 0, sortertListe.Length - 1, verdi);

if (plassIsortertListe == -1)

Console.WriteLine(verdi + " finnes ikke i sortert liste");

else

Console.WriteLine(verdi + " finnes i sortert liste på plass " + plassIsortertListe);

}

}

## Binære trær lag for lag

<https://www.youtube.com/watch?v=9RHO6jU--GU>



Binære trær  
<https://www.youtube.com/watch?v=H5JubkIy_p8>

<https://www.youtube.com/watch?v=86g8jAQug04>

### Iterative

class Node:

def \_\_init\_\_(self, key=None, venstre=None, hoyre=None):

self.key = key

self.venstre = venstre

self.hoyre = hoyre

def skrivUtLag(root, lag):

if root is None:

return False

if lag == 1:

print(root.key, end=' ')

return True

venstre = skrivUtLag(root.venstre, lag - 1)

hoyre = skrivUtLag(root.hoyre, lag - 1)

return venstre or hoyre

def levelOrderTraversal(root):

lag = 1

while skrivUtLag(root, lag):

lag = lag + 1

if \_\_name\_\_ == '\_\_main\_\_':

root = Node(15)

root.venstre = Node(10)

root.hoyre = Node(20)

root.venstre.venstre = Node(8)

root.venstre.hoyre = Node(12)

root.hoyre.venstre = Node(16)

root.hoyre.hoyre = Node(25)

levelOrderTraversal(root)

### Recursive

class Node:

def \_\_init\_\_(self, key=None, venstre=None, hoyre=None):

self.key = key

self.venstre = venstre

self.hoyre = hoyre

def sortere(root, lag, dict):

if root is None:

return

dict.setdefault(lag, []).append(root.key)

sortere(root.venstre, lag + 1, dict)

sortere(root.hoyre, lag + 1, dict)

def levelOrderTraversal(root):

dict = {}

sortere(root, 1, dict)

for i in range(1, len(dict) + 1):

print(f"Lag {i}:", dict[i])

if \_\_name\_\_ == '\_\_main\_\_':

root = Node(15)

root.venstre = Node(10)

root.hoyre = Node(20)

root.venstre.venstre = Node(8)

root.venstre.hoyre = Node(12)

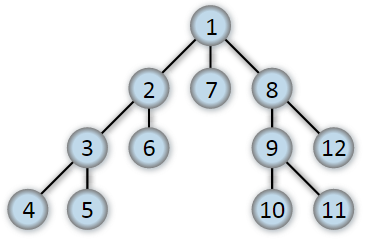
root.hoyre.venstre = Node(16)

root.hoyre.hoyre = Node(25)

root.hoyre.hoyre.hoyre = Node(30)

levelOrderTraversal(root)

## Binære trær dybde først



### Iterative

class Node:

def \_\_init\_\_(self, key=None, venstre=None, hoyre=None):

self.key = key

self.venstre = venstre

self.hoyre = hoyre

def skrivUtLag(root, lag):

if root is None:

return False

if lag == 1:

print(root.key, end=' ')

return True

venstre = skrivUtLag(root.venstre, lag - 1)

hoyre = skrivUtLag(root.hoyre, lag - 1)

return venstre or hoyre

def levelOrderTraversal(root):

lag = 1

while skrivUtLag(root, lag):

lag = lag + 1

if \_\_name\_\_ == '\_\_main\_\_':

root = Node(15)

root.venstre = Node(10)

root.hoyre = Node(20)

root.venstre.venstre = Node(8)

root.venstre.hoyre = Node(12)

root.hoyre.venstre = Node(16)

root.hoyre.hoyre = Node(25)

levelOrderTraversal(root)

### Recursive

class Node:

def \_\_init\_\_(self, key=None, venstre=None, hoyre=None):

self.key = key

self.venstre = venstre

self.hoyre = hoyre

def sortere(root, lag, dict):

if root is None:

return

dict.setdefault(lag, []).append(root.key)

sortere(root.venstre, lag + 1, dict)

sortere(root.hoyre, lag + 1, dict)

def levelOrderTraversal(root):

dict = {}

sortere(root, 1, dict)

for i in range(1, len(dict) + 1):

print(f"Lag {i}:", dict[i])

if \_\_name\_\_ == '\_\_main\_\_':

root = Node(15)

root.venstre = Node(10)

root.hoyre = Node(20)

root.venstre.venstre = Node(8)

root.venstre.hoyre = Node(12)

root.hoyre.venstre = Node(16)

root.hoyre.hoyre = Node(25)

root.hoyre.hoyre.hoyre = Node(30)

levelOrderTraversal(root)

## Inorder, Preorder, Postorder

<https://en.wikipedia.org/wiki/Tree_traversal>



**Depth First Traversals:**Inorder (Left, Root, Right) : 4 2 5 1 3   
Preorder (Root, Left, Right) : 1 2 4 5 3   
Postorder (Left, Right, Root) : 4 5 2 3 1

### inorder iterative

from collections import deque

class Node:

def \_\_init\_\_(self, data=None, venstre=None, hoyre=None):

self.data = data

self.venstre = venstre

self.hoyre = hoyre

def inorder(root):

tallmengde = deque()

gjeldende = root

while tallmengde or gjeldende:

if gjeldende:

tallmengde.append(gjeldende)

gjeldende = gjeldende.venstre

else:

gjeldende = tallmengde.pop()

print(gjeldende.data, end=' ')

gjeldende = gjeldende.hoyre

if \_\_name\_\_ == '\_\_main\_\_':

root = Node(1)

root.venstre = Node(2)

root.hoyre = Node(3)

root.venstre.venstre = Node(4)

root.hoyre.venstre = Node(5)

root.hoyre.hoyre = Node(6)

root.hoyre.venstre.venstre = Node(7)

root.hoyre.venstre.hoyre = Node(8)

inorder(root)

### inorder recursive

class Node:

def \_\_init\_\_(self, data=None, venstre=None, hoyre=None):

self.data = data

self.venstre = venstre

self.hoyre = hoyre

def inorder(root):

if root is None:

return

inorder(root.venstre)

print(root.data, end=' ')

inorder(root.hoyre)

if \_\_name\_\_ == '\_\_main\_\_':

root = Node(1)

root.venstre = Node(2)

root.hoyre = Node(3)

root.venstre.venstre = Node(4)

root.hoyre.venstre = Node(5)

root.hoyre.hoyre = Node(6)

root.hoyre.venstre.venstre = Node(7)

root.hoyre.venstre.hoyre = Node(8)

inorder(root)

### preorder iterative

from collections import deque

class Node:

def \_\_init\_\_(self, data=None, venstre=None, hoyre=None):

self.data = data

self.venstre = venstre

self.hoyre = hoyre

def preorder(root):

if root is None:

return

tallmengde = deque()

tallmengde.append(root)

while tallmengde:

tall = tallmengde.pop()

print(tall.data, end=' ')

if tall.hoyre:

tallmengde.append(tall.hoyre)

if tall.venstre:

tallmengde.append(tall.venstre)

if \_\_name\_\_ == '\_\_main\_\_':

root = Node(1)

root.venstre = Node(2)

root.hoyre = Node(3)

root.venstre.venstre = Node(4)

root.hoyre.venstre = Node(5)

root.hoyre.hoyre = Node(6)

root.hoyre.venstre.venstre = Node(7)

root.hoyre.venstre.hoyre = Node(8)

preorder(root)

### preorder recursive

class Node:

def \_\_init\_\_(self, data=None, venstre=None, hoyre=None):

self.data = data

self.venstre = venstre

self.hoyre = hoyre

def preorder(root):

if root is None:

return

print(root.data, end=' ')

preorder(root.venstre)

preorder(root.hoyre)

if \_\_name\_\_ == '\_\_main\_\_':

root = Node(1)

root.venstre = Node(2)

root.hoyre = Node(3)

root.venstre.venstre = Node(4)

root.hoyre.venstre = Node(5)

root.hoyre.hoyre = Node(6)

root.hoyre.venstre.venstre = Node(7)

root.hoyre.venstre.hoyre = Node(8)

preorder(root)

### postorder iterative

from collections import deque

class Node:

def \_\_init\_\_(self, data=None, venstre=None, hoyre=None):

self.data = data

self.venstre = venstre

self.hoyre = hoyre

def postorder(root):

tallmengde = deque()

tallmengde.append(root)

ut = deque()

while tallmengde:

tall = tallmengde.pop()

ut.append(tall.data)

if tall.venstre:

tallmengde.append(tall.venstre)

if tall.hoyre:

tallmengde.append(tall.hoyre)

while ut:

print(ut.pop(), end=' ')

if \_\_name\_\_ == '\_\_main\_\_':

root = Node(1)

root.venstre = Node(2)

root.hoyre = Node(3)

root.venstre.venstre = Node(4)

root.hoyre.venstre = Node(5)

root.hoyre.hoyre = Node(6)

root.hoyre.venstre.venstre = Node(7)

root.hoyre.venstre.hoyre = Node(8)

postorder(root)

### postorder recursive

## class Node:

def \_\_init\_\_(self, data=None, venstre=None, hoyre=None):

self.data = data

self.venstre = venstre

self.hoyre = hoyre

def postorder(root):

if root is None:

return

postorder(root.venstre)

postorder(root.hoyre)

print(root.data, end=' ')

if \_\_name\_\_ == '\_\_main\_\_':

root = Node(1)

root.venstre = Node(2)

root.hoyre = Node(3)

root.venstre.venstre = Node(4)

root.hoyre.venstre = Node(5)

root.hoyre.hoyre = Node(6)

root.hoyre.venstre.venstre = Node(7)

root.hoyre.venstre.hoyre = Node(8)

postorder(root)

## Insertion sort

<https://www.youtube.com/watch?v=Nkw6Jg_Gi4w>

Et bilde som inneholder tekst, elektronikk, tastatur, kalkulator

Automatisk generert beskrivelse

### Insertion sort iterative

def insertionSort(tallListe):

for i in range(1, len(tallListe)):

verdi = tallListe[i]

j = i

while j > 0 and tallListe[j - 1] > verdi:

tallListe[j] = tallListe[j - 1]

j = j - 1

tallListe[j] = verdi

if \_\_name\_\_ == '\_\_main\_\_':

tallListe = [3, 8, 5, 4, 1, 9, -2]

insertionSort(tallListe)

print(tallListe)

### Insertion sort recursive

def insertionSort(tallListe, i, n):

verdi = tallListe[i]

j = i

while j > 0 and tallListe[j - 1] > verdi:

tallListe[j] = tallListe[j - 1]

j = j - 1

tallListe[j] = verdi

if i + 1 <= n:

insertionSort(tallListe, i + 1, n)

if \_\_name\_\_ == '\_\_main\_\_':

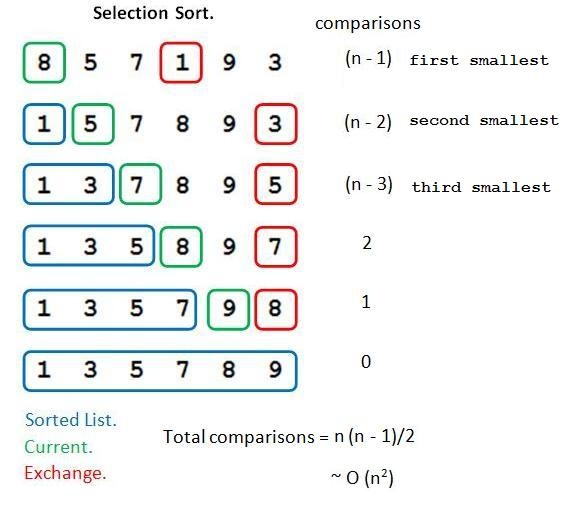
tallListe = [3, 8, 5, 4, 1, 9, -2]

insertionSort(tallListe, 1, len(tallListe) - 1)

print(tallListe)

## Selection sort

<https://www.youtube.com/watch?v=5KjapFQNxUo>



### Selection sort iterative

def byttPlass(tallListe, i, j):

midlertidig = tallListe[i]

tallListe[i] = tallListe[j]

tallListe[j] = midlertidig

def selectionSort(tallListe):

for tellerMinste in range(len(tallListe) - 1):

minst = tellerMinste

for verdi in range(tellerMinste + 1, len(tallListe)):

if tallListe[verdi] < tallListe[minst]:

minst = verdi

byttPlass(tallListe, minst, tellerMinste)

if \_\_name\_\_ == '\_\_main\_\_':

tallListe = [3, 5, 8, 4, 1, 9, -2]

selectionSort(tallListe)

print(tallListe)

### Selection sort recursive

def byttePlass(tallListe, i, j):

midlertidig = tallListe[i]

tallListe[i] = tallListe[j]

tallListe[j] = midlertidig

def selectionSort(tallListe, telleminste, listeLengde):

minst = telleminste

for verdi in range(telleminste + 1, listeLengde):

if tallListe[verdi] < tallListe[minst]:

minst = verdi

byttePlass(tallListe, minst, telleminste)

if telleminste + 1 < listeLengde:

selectionSort(tallListe, telleminste + 1, listeLengde)

if \_\_name\_\_ == '\_\_main\_\_':

tallListe = [3, 5, 8, 4, 1, 9, -2]

selectionSort(tallListe, 0, len(tallListe))

print(tallListe)

## Merge sort

<https://www.youtube.com/watch?v=cVZMah9kEjI>

**Python**

def merge(tallListe, hjelpeliste, forstePlassIListen, midten, sistePlassIListen):

k = forstePlassIListen

i = forstePlassIListen

j = midten + 1

while i <= midten and j <= sistePlassIListen:

if tallListe[i] <= tallListe[j]:

hjelpeliste[k] = tallListe[i]

k = k + 1

i = i + 1

else:

hjelpeliste[k] = tallListe[j]

k = k + 1

j = j + 1

while i <= midten:

hjelpeliste[k] = tallListe[i]

k = k + 1

i = i + 1

for i in range(forstePlassIListen, sistePlassIListen + 1):

tallListe[i] = hjelpeliste[i]

def mergesort(tallListe, hjelpeliste, forstePlassIListen, sistePlassIListen):

if sistePlassIListen == forstePlassIListen:

return

# Bitwise shift >>1

# https://www.youtube.com/watch?v=fDKUq38H2jk

# https://youtu.be/PyfKCvHALj8?t=553

midten = (forstePlassIListen + ((sistePlassIListen - forstePlassIListen) >> 1)) # >>1 er Bitwise shift. Se videoene.

mergesort(tallListe, hjelpeliste, forstePlassIListen, midten)

mergesort(tallListe, hjelpeliste, midten + 1, sistePlassIListen)

merge(tallListe, hjelpeliste, forstePlassIListen, midten, sistePlassIListen)

def erSortert(tallListe):

forrige = tallListe[0]

for i in range(1, len(tallListe)):

if forrige > tallListe[i]:

print("MergeSort fungerer ikke med tallene!!")

return False

forrige = tallListe[i]

return True

if \_\_name\_\_ == '\_\_main\_\_':

tallListe = [12, 3, 18, 24, 0, 5, -2]

hjelpeliste = tallListe.copy()

mergesort(tallListe, hjelpeliste, 0, len(tallListe) - 1)

if erSortert(tallListe):

print(tallListe)

**C#**

using System;

namespace MergeSortDemo

{

class Program

{

static public void MergeSort(int[] tallListe, int forstePlassIListen, int midten, int sistePlassIListen)

{

int[] midlertidig = new int[25];

int i, vendtreVerdi, antallVerdier, midlertidigPlass;

vendtreVerdi = (midten - 1);

midlertidigPlass = forstePlassIListen;

antallVerdier = (sistePlassIListen - forstePlassIListen + 1);

while ((forstePlassIListen <= vendtreVerdi) && (midten <= sistePlassIListen))

{

if (tallListe[forstePlassIListen] <= tallListe[midten])

midlertidig[midlertidigPlass++] = tallListe[forstePlassIListen++];

else

midlertidig[midlertidigPlass++] = tallListe[midten++];

}

while (forstePlassIListen <= vendtreVerdi)

midlertidig[midlertidigPlass++] = tallListe[forstePlassIListen++];

while (midten <= sistePlassIListen)

midlertidig[midlertidigPlass++] = tallListe[midten++];

for (i = 0; i < antallVerdier; i++)

{

tallListe[sistePlassIListen] = midlertidig[sistePlassIListen];

sistePlassIListen--;

}

}

static public void Sortere(int[] tallListe, int forstePlassIListen, int sistePlassIListen)

{

int midten;

if (sistePlassIListen > forstePlassIListen)

{

midten = (sistePlassIListen + forstePlassIListen) / 2;

Sortere(tallListe, forstePlassIListen, midten);

Sortere(tallListe, (midten + 1), sistePlassIListen);

MergeSort(tallListe, forstePlassIListen, (midten + 1), sistePlassIListen);

}

}

static void Main(string[] args)

{

int[] tallListe = { 99, 0, 13, 300, 89, -9, 4, 2, 200 };

int lengde = tallListe.Length;

Console.WriteLine("Listen usortert:");

foreach (int verdi in tallListe)

{

Console.Write(verdi + " ");

}

Console.WriteLine();

Console.WriteLine("Listen sortert");

Sortere(tallListe, 0, lengde - 1);

foreach (int verdi in tallListe)

{

Console.Write(verdi + " ");

}

}

}

}

## Quicksort

<https://www.youtube.com/watch?v=SLauY6PpjW4&t=137s>

<https://www.youtube.com/watch?v=9KBwdDEwal8>

**Python**

def bytt(tallListe, sistePlassIListen, indexPlassIListen):

midlertidigVerdi = tallListe[sistePlassIListen]

tallListe[sistePlassIListen] = tallListe[indexPlassIListen]

tallListe[indexPlassIListen] = midlertidigVerdi

def partition(tallListe, forstePlassIListen, sistePlassIListen):

pivot = tallListe[sistePlassIListen]

pIndex = forstePlassIListen

for verdi in range(forstePlassIListen, sistePlassIListen):

if tallListe[verdi] <= pivot:

bytt(tallListe, verdi, pIndex)

pIndex = pIndex + 1

bytt(tallListe, sistePlassIListen, pIndex)

return pIndex

def quicksort(tallListe, forstePlassIListen, sistePlassIListen):

if forstePlassIListen >= sistePlassIListen:

return

pivot = partition(tallListe, forstePlassIListen, sistePlassIListen)

quicksort(tallListe, forstePlassIListen, pivot - 1) # Pivot er feltet som deler tall listen

quicksort(tallListe, pivot + 1, sistePlassIListen)

if \_\_name\_\_ == '\_\_main\_\_':

tallListe = [9, -3, 5, 2, 6, 8, -6, 1, 3]

quicksort(tallListe, 0, len(tallListe) - 1)

print(tallListe)

**C#**

using System;

namespace Quick\_Sort

{

class Program

{

private static void Quick\_Sort(int[] tallListe, int forstePlassIListen, int sistePlassIListen)

{

if (forstePlassIListen < sistePlassIListen)

{

int pivot = Dele(tallListe, forstePlassIListen, sistePlassIListen);

if (pivot > 1)

{

Quick\_Sort(tallListe, forstePlassIListen, pivot - 1);

}

if (pivot + 1 < sistePlassIListen)

{

Quick\_Sort(tallListe, pivot + 1, sistePlassIListen);

}

}

}

private static int Dele(int[] tallListe, int forstePlassIListen, int sistePlassIListen)

{

int pivot = tallListe[forstePlassIListen];

while (true)

{

while (tallListe[forstePlassIListen] < pivot)

{

forstePlassIListen++;

}

while (tallListe[sistePlassIListen] > pivot)

{

sistePlassIListen--;

}

if (forstePlassIListen < sistePlassIListen)

{

if (tallListe[forstePlassIListen] == tallListe[sistePlassIListen]) return sistePlassIListen;

int midlertidig = tallListe[forstePlassIListen];

tallListe[forstePlassIListen] = tallListe[sistePlassIListen];

tallListe[sistePlassIListen] = midlertidig;

}

else

{

return sistePlassIListen;

}

}

}

static void Main(string[] args)

{

int[] tallListe = new int[] { 45, 6, 19, -21, 299, -34, 1, 0, 62, 5 };

Console.WriteLine("Tall listen usortert : ");

foreach (var item in tallListe)

{

Console.Write(" " + item);

}

Console.WriteLine();

Quick\_Sort(tallListe, 0, tallListe.Length - 1);

Console.WriteLine();

Console.WriteLine("Tall listen sortert : ");

foreach (var item in tallListe)

{

Console.Write(" " + item);

}

Console.WriteLine();

}

}

}

## Et bilde som inneholder overvåke, himmel, svart, skjerm Automatisk generert beskrivelseHeapsort

<https://www.youtube.com/watch?v=2DmK_H7IdTo>

**Python**

def heapify(tallListe, midten, lengde):

venstre = 2\*midten + 1

hoyre = 2\*midten + 2

largest = midten

if venstre < lengde and tallListe[venstre] > tallListe[midten]:

largest = venstre

if hoyre < lengde and tallListe[hoyre] > tallListe[largest]:

largest = hoyre

if largest != midten:

midlertidig = tallListe[midten]

tallListe[midten] = tallListe[largest]

tallListe[largest] = midlertidig

heapify(tallListe, largest, lengde)

def fjerne(tallListe, size):

if size <= 0:

return -1

top = tallListe[0]

tallListe[0] = tallListe[size - 1]

heapify(tallListe, 0, size - 1)

return top

def heapsort(tallListe):

lengde = len(tallListe)

midten = (lengde - 2) // 2

while midten >= 0:

heapify(tallListe, midten, lengde)

midten = midten - 1

while lengde:

tallListe[lengde - 1] = fjerne(tallListe, lengde)

lengde = lengde - 1

if \_\_name\_\_ == '\_\_main\_\_':

tallListe = [6, 4, 7, 1, 9, -2]

heapsort(tallListe)

print(tallListe)

**C#**

using System;

namespace HeapSort

{

public class example

{

static void HeapSort(int[] tallListe, int lengde)

{

for (int midten = lengde / 2 - 1; midten >= 0; midten--)

Heapify(tallListe, lengde, midten);

for (int plass = lengde - 1; plass >= 0; plass--)

{

int midlertidig = tallListe[0];

tallListe[0] = tallListe[plass];

tallListe[plass] = midlertidig;

Heapify(tallListe, plass, 0);

}

}

static void Heapify(int[] tallListe, int plass, int verdi)

{

int storst = verdi;

int left = 2 \* verdi + 1;

int right = 2 \* verdi + 2;

if (left < plass && tallListe[left] > tallListe[storst])

storst = left;

if (right < plass && tallListe[right] > tallListe[storst])

storst = right;

if (storst != verdi)

{

int midlertidig = tallListe[verdi];

tallListe[verdi] = tallListe[storst];

tallListe[storst] = midlertidig;

Heapify(tallListe, plass, storst);

}

}

public static void Main()

{

int[] tallListe = { 6, 4, 7, 1, 9, -2 };

int lengde = tallListe.Length;

Console.Write("Usortert liste: ");

int teller;

for (teller = 0; teller < lengde; teller++)

{

Console.Write(tallListe[teller] + " ");

}

HeapSort(tallListe, lengde);

Console.Write("\nSortert liste: ");

for (teller = 0; teller < lengde; teller++)

{

Console.Write(tallListe[teller] + " ");

}

}

}

}

# Oppgave

**Bruk alle datastrukturene og algoritmene i praktiske eksempler du skal beskrive hva koden gjør..**

**Du trenger ikke bruke både iterative og rekursive versjoner, og du velger ett av språkene**