COS/SE Computersystemer

Lektion #10
Mere algoritmer.

Morten Hansen
moh@sdu.dk

Potens

Eks.: $5^3 = 125$

Generelt: $x^y = ?$

eller: $base^{exp} = ?$

Vi har brug for en algoritme!

Iterativ potens agoritme

Ide:

x^y udregnes ved at gange x med sig selv y gange.

```
def power( x, y ):

  result = 1

  while( y > 0 ):
    result = result * x
    y = y -1

  return result
```

Recursiv potens agoritme

Ide:

$x^y = x * x^{y-1}$. power(x, y) = x * power(x, y-1)

```
def power( x, y ):

  if( y<= 0 ):
    return 1
  return x * power( x, y-1 )
```

Iterative

Iterative Structures

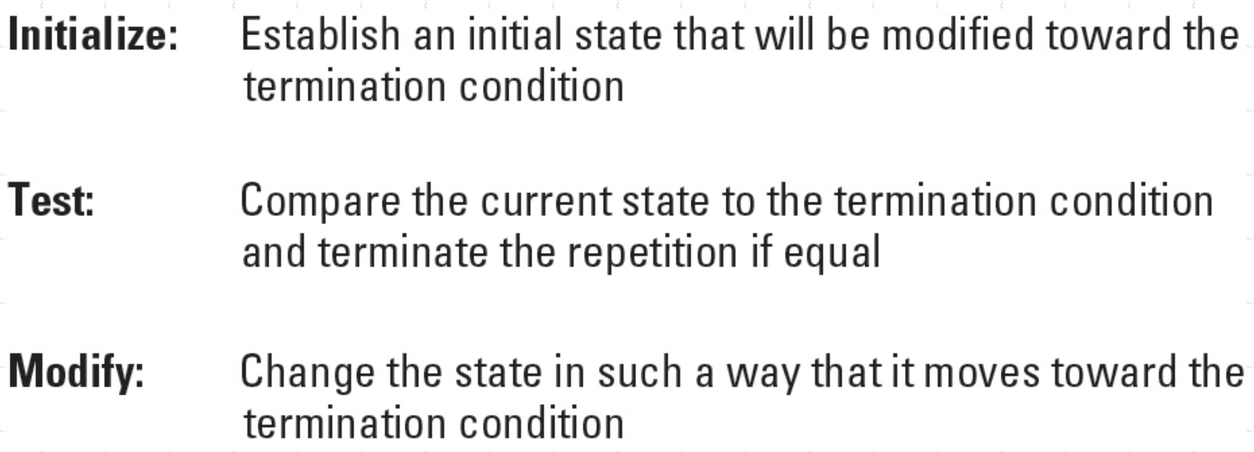
- A collection of instructions repeated in a looping manner
- Examples include:
  - Sequential Search Algorithm
  - Insertion Sort Algorithm

Figure 5.6 The sequential search algorithm in pseudocode

```
def Search (List, TargetValue):
  if (List is empty):
    Declare search a failure
  else:
    Select the first entry in List to be TestEntry
    while (TargetValue > TestEntry and entries remain):
      Select the next entry in List as TestEntry
      if (TargetValue == TestEntry):
        Declare search a success
      else:
        Declare search a failure
```

Morten Hansen
moh@sdu.dk

Components of repetitive control

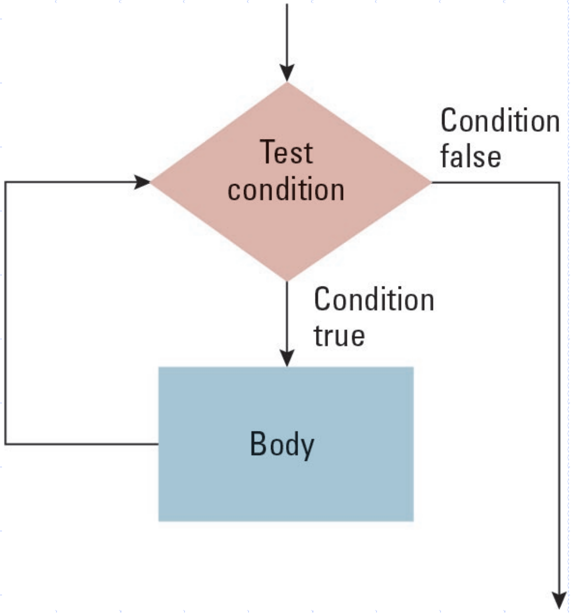


Initialize: Establish an initial state that will be modified toward the termination condition

Test: Compare the current state to the termination condition and terminate the repetition if equal

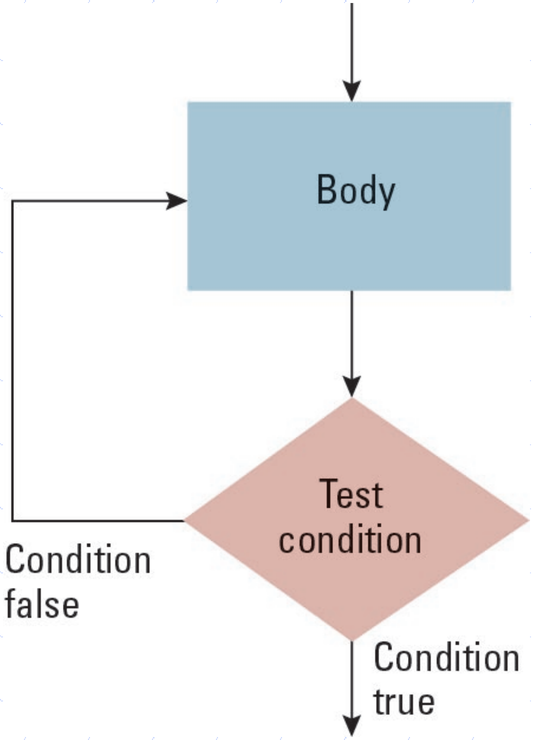
Modify: Change the state in such a way that it moves toward the termination condition

"while" løkke



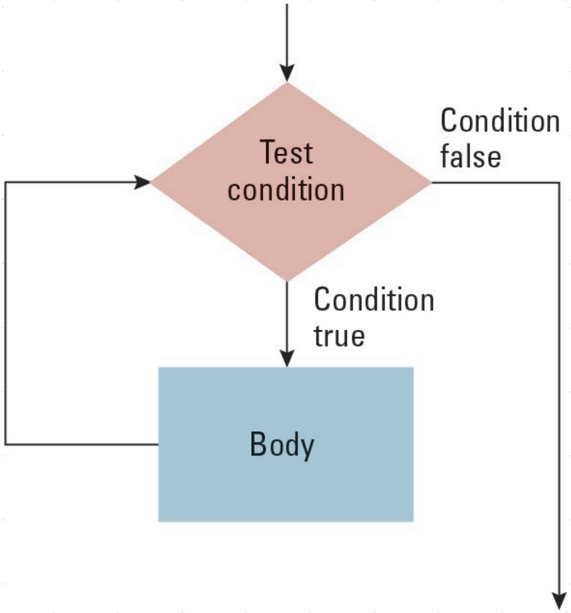
```
while ( condition ):
  ..do your stuff
```

"repeat until" løkke



```
clear condition
while not ( condition ):
  ..do your stuff
```

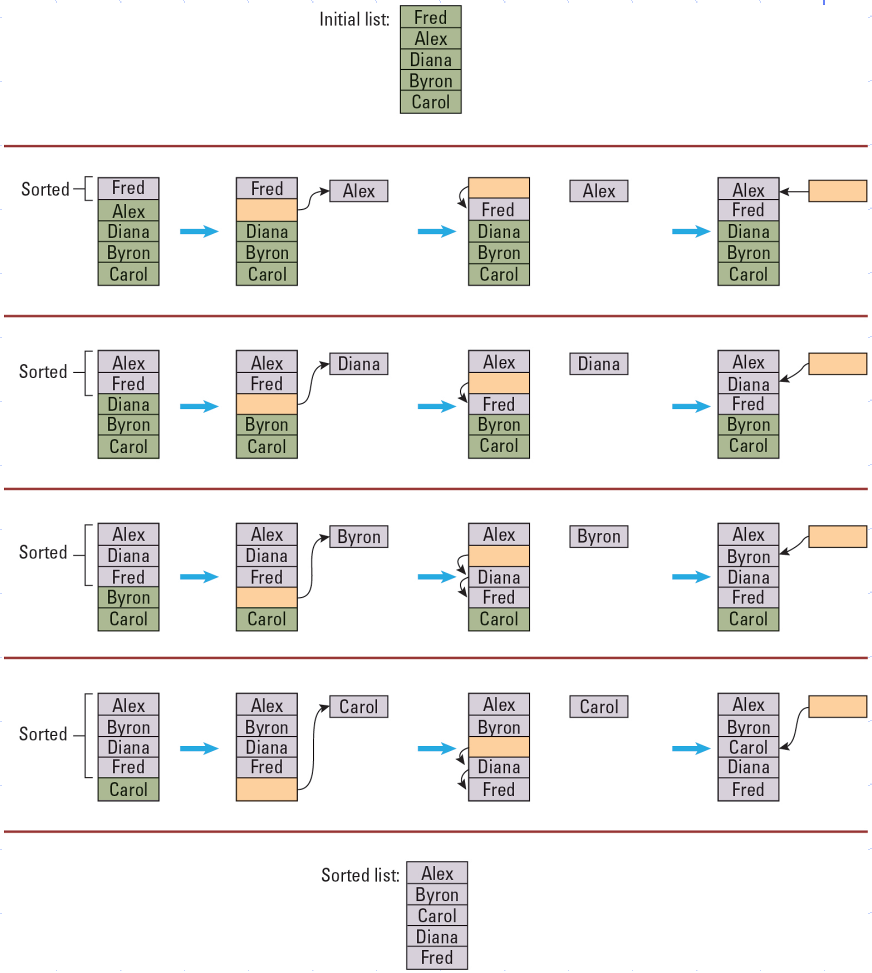
"for" løkke
looper et kendt antal gange.



```
n = 12
i = 0
while( i < n ):
  ..do your stuff
  i++
```

Figure 5.10 Sorting the list Fred, Alex, Diana, Byron, and Carol alphabetically – insertion sort

```
def Sort(List):
    N = 2
    while (N <= length of List):
        Pivot = Nth entry in List
        Remove Nth entry leaving a hole in List
        while (there is an Entry above the
                  hole and Entry > Pivot):
            Move Entry down into the hole leaving
            a hole in the list above the Entry
        Move Pivot into the hole
        N = N + 1
```



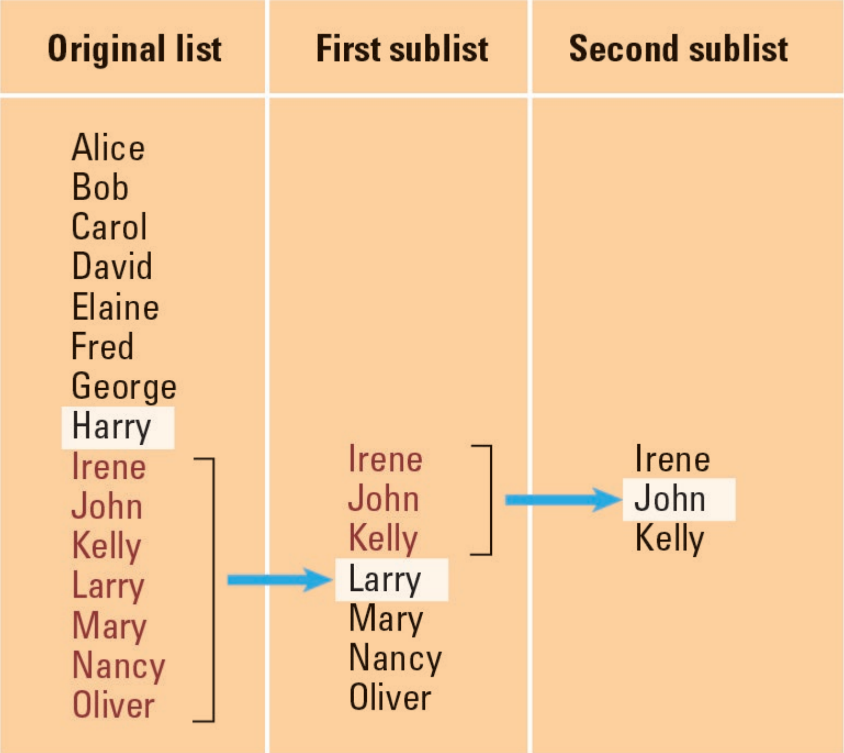
Morten Hansen
moh@sdu.dk

Recursive

Recursive Structures

- Repeating the set of instructions as a subtask of itself.
- Multiple activations of the procedure are formed, all but one of which are waiting for other activations to complete.
- Requires initialization, modification, and a test for termination (base case)
- Provides the illusion of multiple copies of the function, created dynamically in a telescoping manner
- Only one copy is actually running at a given time, the others are waiting
- Example: The Binary Search Algorithm

Figure 5.12 Applying our strategy to search a list for the entry John



Original list

Alice
Bob
Carol
David
Elaine
Fred
George
Harry
Irene
John
Kelly
Larry
Mary
Nancy
Oliver

First sublist

Irene
John
Kelly
Larry
Mary
Nancy
Oliver

Second sublist

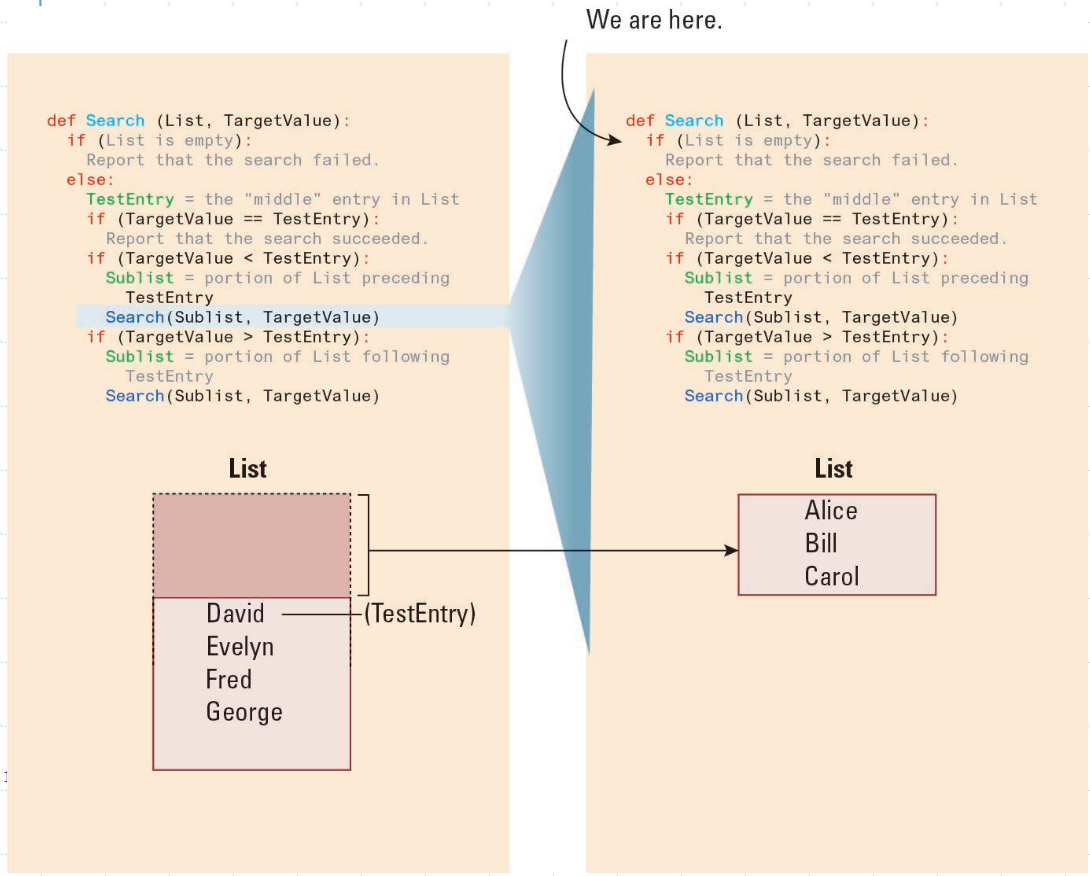
Irene
John
Kelly

```
if (List is empty):
    Report that the search failed
else:
    TestEntry = middle entry in the List
    if (TargetValue == TestEntry):
        Report that the search succeeded
    if (TargetValue < TestEntry):
        Search the portion of List preceding TestEntry for
        TargetValue, and report the result of that search
    if (TargetValue > TestEntry):
        Search the portion of List following TestEntry for
        TargetValue, and report the result of that search
```

Morten Hansen
moh@sdu.dk

Recursively Searching

Søger efter Bill

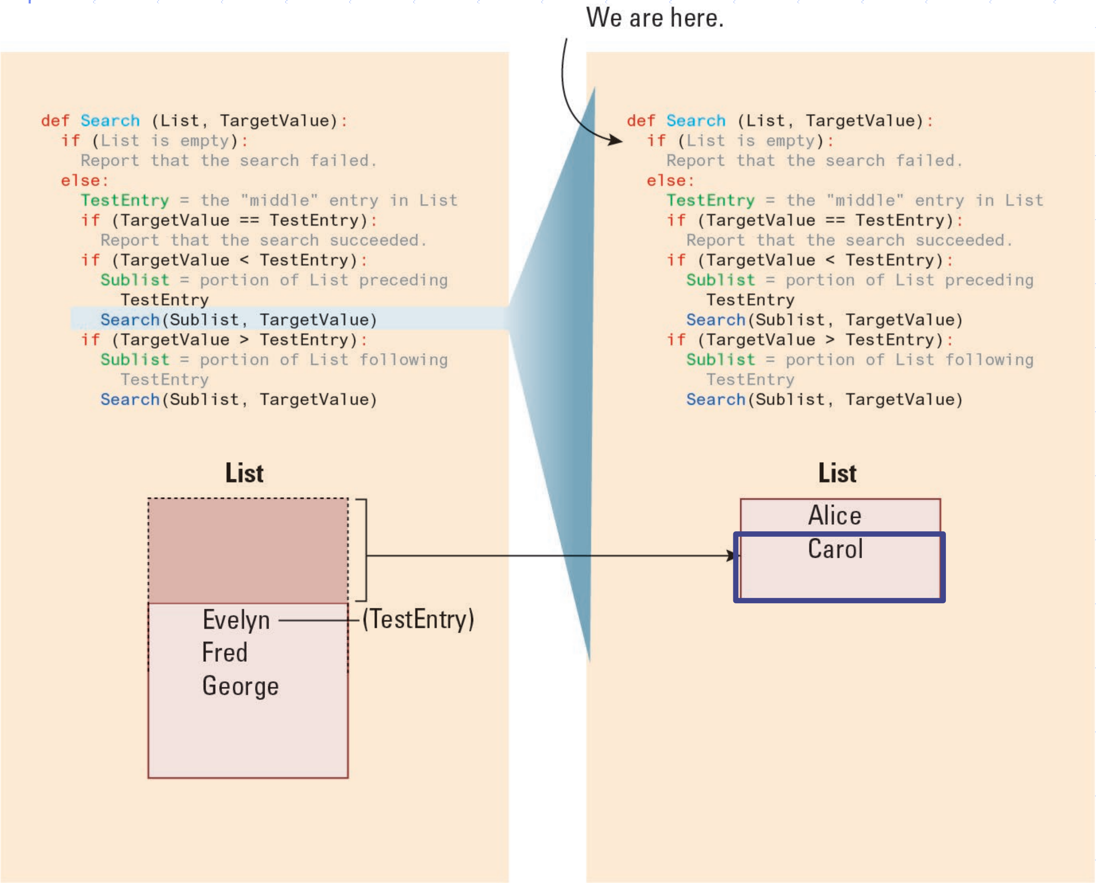


We are here.

```
def Search (List, TargetValue):
  if (List is empty):
    Report that the search failed.
  else:
    TestEntry = the "middle" entry in List
    if (TargetValue == TestEntry):
      Report that the search succeeded.
    if (TargetValue < TestEntry):
      Sublist = portion of List preceding
        TestEntry
      Search(Sublist, TargetValue)
    if (TargetValue > TestEntry):
      Sublist = portion of List following
        TestEntry
      Search(Sublist, TargetValue)
```

```
def Search (List, TargetValue):
  if (List is empty):
    Report that the search failed.
  else:
    TestEntry = the "middle" entry in List
    if (TargetValue == TestEntry):
      Report that the search succeeded.
    if (TargetValue < TestEntry):
      Sublist = portion of List preceding
        TestEntry
      Search(Sublist, TargetValue)
    if (TargetValue > TestEntry):
      Sublist = portion of List following
        TestEntry
      Search(Sublist, TargetValue)
```

Recursively Searching



We are here.

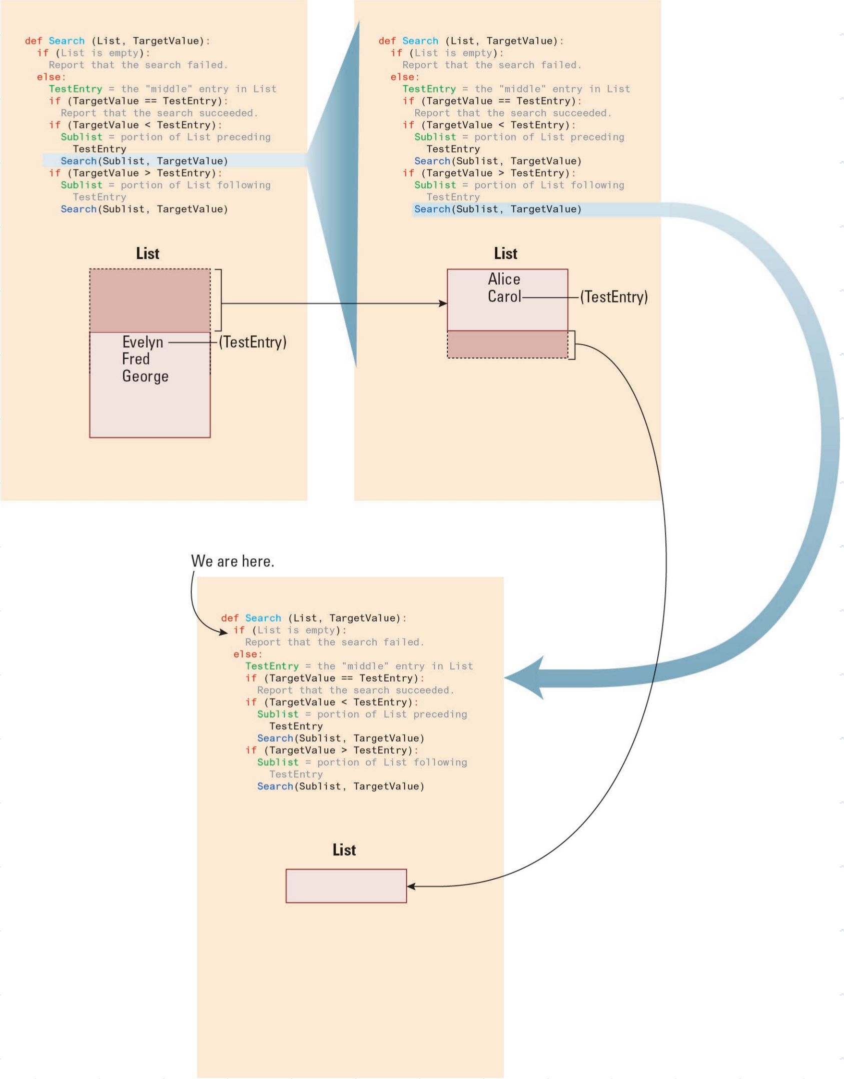
```
def Search (List, TargetValue):
  if (List is empty):
    Report that the search failed.
  else:
    TestEntry = the "middle" entry in List
    if (TargetValue == TestEntry):
      Report that the search succeeded.
    if (TargetValue < TestEntry):
      Sublist = portion of List preceding
        TestEntry
      Search(Sublist, TargetValue)
    if (TargetValue > TestEntry):
      Sublist = portion of List following
        TestEntry
      Search(Sublist, TargetValue)
```

```
def Search (List, TargetValue):
  if (List is empty):
    Report that the search failed.
  else:
    TestEntry = the "middle" entry in List
    if (TargetValue == TestEntry):
      Report that the search succeeded.
    if (TargetValue < TestEntry):
      Sublist = portion of List preceding
        TestEntry
      Search(Sublist, TargetValue)
    if (TargetValue > TestEntry):
      Sublist = portion of List following
        TestEntry
      Search(Sublist, TargetValue)
```

Søger efter Charlotte

Recursively Searching

Søger efter David

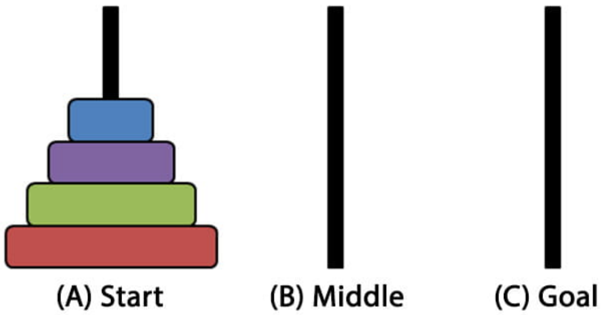


```
def Search (List, TargetValue):
  if (List is empty):
    Report that the search failed.
  else:
    TestEntry = the "middle" entry in List
    if (TargetValue == TestEntry):
      Report that the search succeeded.
    if (TargetValue < TestEntry):
      Sublist = portion of List preceding
        TestEntry
      Search(Sublist, TargetValue)
    if (TargetValue > TestEntry):
      Sublist = portion of List following
        TestEntry
      Search(Sublist, TargetValue)
```

```
def Search (List, TargetValue):
  if (List is empty):
    Report that the search failed.
  else:
    TestEntry = the "middle" entry in List
    if (TargetValue == TestEntry):
      Report that the search succeeded.
    if (TargetValue < TestEntry):
      Sublist = portion of List preceding
        TestEntry
      Search(Sublist, TargetValue)
    if (TargetValue > TestEntry):
      Sublist = portion of List following
        TestEntry
      Search(Sublist, TargetValue)
```

```
def Search (List, TargetValue):
  if (List is empty):
    Report that the search failed.
  else:
    TestEntry = the "middle" entry in List
    if (TargetValue == TestEntry):
      Report that the search succeeded.
    if (TargetValue < TestEntry):
      Sublist = portion of List preceding
        TestEntry
      Search(Sublist, TargetValue)
    if (TargetValue > TestEntry):
      Sublist = portion of List following
        TestEntry
      Search(Sublist, TargetValue)
```

The Tower of Hanoi



```
def moveTower( n, start, goal ):

   if( n == 1 ):
      moveSlice( start, goal )
   else:
      middle = findMiddle( start, goal )
      moveTower( n-1, start, middle )
      moveSlice( start, goal )
      moveTower( n-1, middle, goal )
```

Miniprojekt

Opgaven

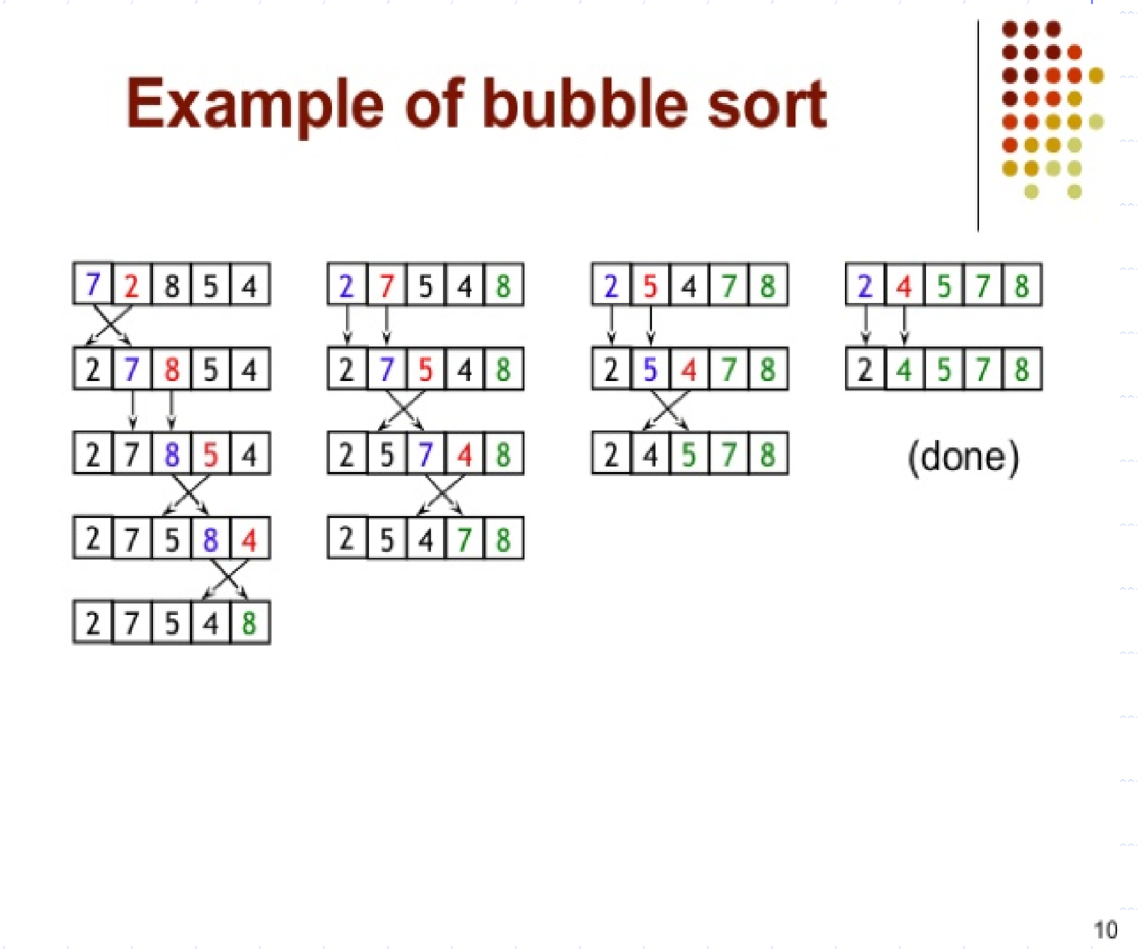
- Løses som udgangspunkt i grupper af 3, fra samme klasse.
- Der skal løses 4 opgaver, som ses på de næste slides.
  - 2 Sortering algoritmer
  - 2 Søgnings algoritmer
  - Test hvornår hvilke algoritmerne der er bedst på udvalgte inputs.

Sortering algoritmer

- **Bubble sort**
- **Merge sort**

Bubble sort

- In this example I will be outlining an example of how bubble sort works for a five-element list/array.
  - Start with the first two elements of the unsorted array.
  - Compare elements numbe 1 and 2. Swap the order i the second is less than th third
  - Compare elements numbe 2 and 3. Swap the order i the third is less than the second.
  - Compare elements numbe 3 and 4. Swap the order i the fourth is less than the third.
  - Compare elements number 4 and 5. Swap the order if the fifth is less than the fourth.



Example of bubble sort

7 2 8 5 4
2 7 8 5 4
2 7 8 5 4
2 7 5 8 4
2 7 5 4 8

2 7 5 4 8
2 7 5 4 8
2 5 7 4 8
2 5 4 7 8

2 5 4 7 8
2 5 4 7 8
2 4 5 7 8

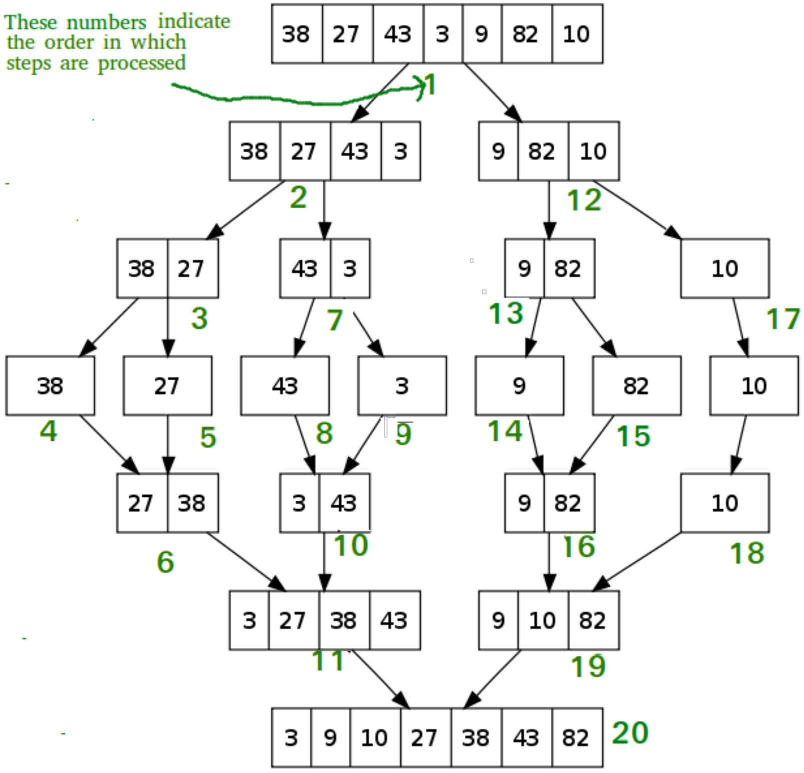
2 4 5 7 8
2 4 5 7 8
(done)

10

https://medium.com/@Esswhy91/what-the-heck-is-bubble-sort-25ee0b374b51

Merge sort

- Er en "Divide and Conquer" algoritme.
  - Deler problemet I mindre stykker
  - Sorterer reskosivte ved brug af merge
  - Ligger to dele sammen af gangen
- Består af to funktioner:
  - mergeSort
    - Deler listen op I minder dele
  - merge
    - Ligger to dele sammen af gangen



https://www.geeksforgeeks.org/merge-sort/

Test cases

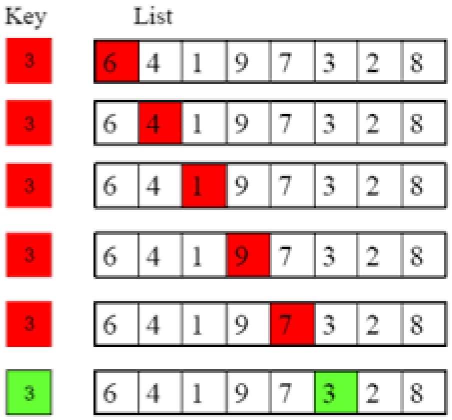
- Undersøg hvordan sorteringsalgoritmerne fungerer på input:
  - Liste af 10 elementer.
  - Liste af 1000 elementer, gerne flere.
- Undersøg hvor hurtig sorteringen er i:
  - Best case – sorterede lister.
  - Worst case – omvendt sorterede lister.
  - Randomiseret input.

Søgnings algoritmer

- **Linear search**
- **Binary search**

Linear search

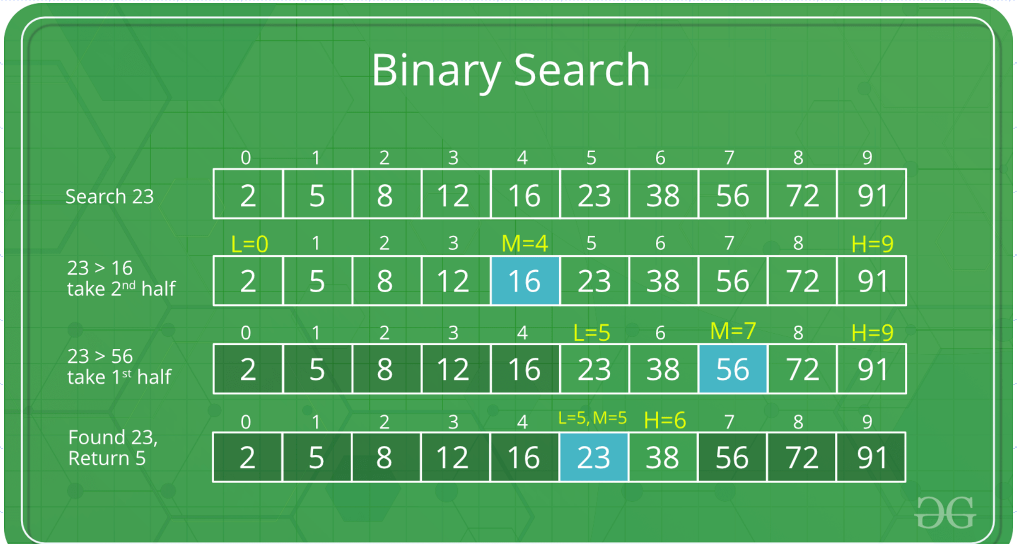
- Søger sekventielt igennem en liste for at finde en key.
- Skal ikke have sorterede input, men det bør I bruge I jeres løsninger



http://mathcenter.oxford.emory.edu/site/cs170/searchAndSort/

Binary search

- Skal have sorterede input.
- Deler problemet I mindre stykker, og kalder sig selv rekusivt:
  - Compare x with the middle element.
  - If x matches with middle element, we return the mid index.
  - Else If x is greater than the mid element, then x can only lie in right half subarray after the mid element. So we recur for right half.
  - Else (x is smaller) recur for the left half.



Binary Search

	0	1	2	3	4	5	6	7	8	9
Search 23	2	5	8	12	16	23	38	56	72	91
	L=0	1	2	3	M=4	5	6	7	8	H=9
23 > 16 take 2nd half	2	5	8	12	16	23	38	56	72	91
	0	1	2	3	4	L=5	6	M=7	8	H=9
23 > 56 take 1st half	2	5	8	12	16	23	38	56	72	91
	0	1	2	3	4	L=5, M=5	H=6	7	8	9
Found 23, Return 5	2	5	8	12	16	23	38	56	72	91

Test cases

- Undersøg hvordan sorteringsalgoritmerne fungerer på input:
  - Liste af længden 10, hvor alle elementer er forskellige.
  - Liste af længden 1000, hvor alle elementer er forskellige.
- Undersøg hvilken af søgninger er hurtigst i tilfælde af:
  - Leder efter første element i listen.
  - Søger efter sidste element i listen.
  - Søger efter det midterst element.
  - Søger efter et tilfældigt element i listen.

Spørgsmål?

Morten Hansen
moh@sdu.dk