Bst – binary search tree

ALGORITHM: INNSETTING I ET BINÆRT SØKETRE

Input: En node v og et element x
Output: En oppdatert node v der en node som inneholder x er en etterkommer av v

```
Procedure Insert(v, x)
    if v = null then
        v ← new Node(x)
    else if x < v.element then
        v.left ← Insert(v.left, x)
    else if x > v.element then
        v.right ← Insert(v.right, x)
    return v
```

ALGORITHM: OPPSLAG I ET BINÆRT SØKETRE

Input: En node v og et element x
Output: Dersom x forekommer i en node u som en etterkommer av v, returner u, ellers **null**.

```
Procedure Search(v, x)
    if v = null then
        return null
    if v.element = x then
        return v
    if x < v.element then
        return Search(v.left, x)
    if x > v.element then
        return Search(v.right, x)
```

ALGORITHM: SLETT EN NODE I ET BINÆRT SØKETRE

Input: En node v og et element x
Output: Dersom x forekommer i en node u som en etterkommer av v, fjern u.

```
Procedure Remove(v, x)
    if v = null then
        return null
    if x < v.element then
        v.left ← Remove(v.left, x)
        return v
    if x > v.element then
        v.right ← Remove(v.right, x)
        return v
    if v.left = null then
        return v.right
    if v.right = null then
        return v.left
    u ← FindMin(v.right)
    v.element ← u.element
    v.right ← Remove(v.right, u.element)
    return v
```

ALGORITHM: FINN MINSTE NODE

Input: En node v
Output: Returner noden som inneholder den minste etterkommeren av v

```
Procedure FindMin(v)
    Etterlatt som øvelse!
```

heaps

ALGORITHM: INNSETTING I HEAP

Input: Et array A som representerer en heap med n elementer, og et element x
Output: Et array som representerer en heap, som inneholder x

```
Procedure Insert(A, x)
    A[n] ← x
    i ← n
    while 0 < i and A[i] < A[ParentOf(i)] do
        A[i], A[ParentOf(i)] ← A[ParentOf(i)], A[i]
        i ← ParentOf(i)
```

ALGORITHM: FJERNING AV MINSTE ELEMENT FRA HEAP

Input: Et ikke-tomt array A som representerer en heap med n elementer
Output: Det minste elementet fjernes og returneres

```
Procedure RemoveMin(A)
    x ← A[0]
    A[0] ← A.pop() // delete and return last element
    i ← 0
    while RightOf(i) < n − 1 do
        j ← if A[LeftOf(i)] ≤ A[RightOf(i)] then LeftOf(i) else RightOf(i)
        if A[j] > A[i] then
            break
        A[i], A[j] ← A[j], A[i]
        i ← j
    if LeftOf(i) < n − 1 and A[LeftOf(i)] ≤ A[i] then
        A[i], A[LeftOf(i)] ← A[LeftOf(i)], A[i]
    return x
```

```
1 Procedure ParentOf(i)
2     return ⌊(i−1)/2⌋
```

```
1 Procedure LeftOf(i)
2     return 2 · i + 1
```

```
1 Procedure RightOf(i)
2     return 2 · i + 2
```

Klassikerene

ALGORITHM: RETT FREM SØK

Input: Et array A og et element x
Output: Hvis x er i arrayet A, returner **true** ellers **false**

```
Procedure Search(A, x)
    for i ← 0 to |A| − 1 do
        if A[i] = x then
            return true
    return false
```

Binærsøk - O(N)

```
1 Procedure Constant(n)
2     return n · 3                                  // O(1)
```

```
1 Procedure Log(n)
2     i ← n
3     while i > 0 do
4         Constant(i)
5         i ← ⌊i/2⌋                                 // O(log(n))
```

```
1 Procedure Linear(n)
2     for i ← 0 to n − 1 do
3         Constant(i)                               // O(n)
```

```
1 Procedure Linearithmic(n)
2     for i ← 0 to n − 1 do
3         Log(n)                                    // O(n · log(n))
```

```
1 Procedure Quadratic(n)
2     for i ← 0 to n − 1 do
3         for j ← 0 to n − 1 do
4             Constant(i)                           // O(n^2)
```

```
1 Procedure Polynomial(n)
2     for i_1 ← 0 to n − 1 do
3         for i_2 ← 0 to n − 1 do
4             ⋮
5             for i_k ← 0 to n − 1 do
6                 Constant(i)                       // O(n^k)
```

```
1 Procedure Exponential(n)
2     if n = 0 then
3         return 1
4     a ← Exponential(n − 1)
5     b ← Exponential(n − 1)
6     return a + b                                  // O(2^n)
```

ALGORITHM: BUBBLE SORT

Input: Et array A med n elementer
Output: Et *sortert* array med de samme n elementene

```
Procedure BubbleSort(A)
    for i ← 0 to n − 2 do
        for j ← 0 to n − i − 2 do
            if A[j] > A[j + 1] then
                A[j], A[j + 1] ← A[j + 1], A[j]
```

Bubble sort – O(n^2)

ALGORITHM: SELECTION SORT

Input: Et array A med n elementer
Output: Et *sortert* array med de samme n elementene

```
Procedure SelectionSort(A)
    for i ← 0 to n − 1 do
        k ← i
        for j ← i + 1 to n − 1 do
            if A[j] < A[k] then
                k ← j
        if i ≠ k then
            A[i], A[k] ← A[k], A[i]
```

selection sort – O(n^2)

ALGORITHM: INSERTION SORT

Input: Et array A med n elementer
Output: Et *sortert* array med de samme n elementene

```
Procedure InsertionSort(A)
    for i ← 1 to n − 1 do
        j ← i
        while j > 0 and A[j-1] > A[j] do
            A[j-1], A[j] ← A[j], A[j-1]
            j ← j − 1
```

insertion sort – O(n^2)

ALGORITHM: SORTERT FLETTING AV TO ARRAYER

Input: To sorterte arrayer A_1 og A_2 og et array A, der $|A_1| + |A_2| = |A| = n$
Output: Et sortert array A med elementene fra A_1 og A_2

```
Procedure Merge(A_1, A_2, A)
    i ← 0
    j ← 0
    while i < |A_1| and j < |A_2| do
        if A_1[i] ≤ A_2[j] then
            A[i + j] ← A_1[i]
            i ← i + 1
        else
            A[i + j] ← A_2[j]
            j ← j + 1
    while i < |A_1| do
        A[i + j] ← A_1[i]
        i ← i + 1
    while j < |A_2| do
        A[i + j] ← A_2[j]
        j ← j + 1
    return A
```

ALGORITHM: MERGE SORT

Input: Et array A med n elementer
Output: Et *sortert* array med de samme n elementene

```
Procedure MergeSort(A)
    if n ≤ 1 then
        return A
    i ← ⌊n/2⌋
    A_1 ← MergeSort(A[0..i-1])
    A_2 ← MergeSort(A[i..n-1])
    return Merge(A_1, A_2, A)
```

merge sort – O(n log n)