In2010 – Innleveringsoppgave1
Adasam
23. September 2022

Oppgave 1
Deloppgave a)
Se fil SetBinarySearchTree.py

Oppgave 2
Deloppgave a)

Algoritmene til klassen Teque

```
class Teque()
    Constructor()
        Teque <- []
    Procedure push_back(x)
        Teque <-  teque + [x]
    Procedure push_front(x)
        Teque <- [x] + teque
    Procedure push_middle(x)
        size <- 0
        for i in teque
            size <- size +1
        middle <- ⌊(size + 1)/2⌋
        teque <- teque[0:middle] + [x] + teque[middle:(size+1)]
    Procedure get(i)
        Print teque[i]
```

Deloppgave b)
Se fil Teque.py

Deloppgave c)
Push_back har køretids kompleksitet O(1), da den kun bruker aritmetiske operasjoner og tilordninger.

Push_front har køretids kompleksitet O(1), da den kun bruker aritmetiske operasjoner og tilordninger.

Push_middle hard køretids kompleksitet O(n), da den kjører en for loop igjennom helle n og ellers bruker bare aritmetiske operasjoner og tilordninger.

Get har køretidskompleksitet O(1), den den kun aksesserer på index.

Deloppgave d)
Hvis vi vet at N er begrenset vil kjøretidskompleksiteten til push_back, push_front, og get forbli uendret, da de ikke er avhengige av størrelsen på N.
Get opperasjonen som er linær vil, hvis N er begrenset til en konstant, også få konstant tid

Oppgave 3

Deloppgave a)

```
Algoritme: finne veien ut
Procedure findWayOut (tree, x)
    Print x
    New = tree[x]
    FindWayOut(tree, new)
```

Kommentar: Jeg hadde først planlagt slik som pseudokoden over, men fikk en feil da jeg ikke håndterte roten. Dermed endte min algoritme slik:

```
Algoritme: finne veien ut
Procedure findWayOut (tree, x)
    Print x
    For key in tree.keys()
        If key equals x
            New = tree[x]
            findWayOut(tree, new)
```

Deloppgav b)

Se fil Katt.py

Oppgave 4

Deloppgave a)

```
Algoritme: Gjøre om et sortert array til et balansert søketre
Input: Et sortert array og en tom liste som skal bli det balanserte søketreet
Output: Et balansert søketre
Procedure MakeBalanceSearchTree( inList, inBalanceList)
    If len(inList) is less than or equal to 3
        Middele <- ⌊(len(inList) - 1)/2⌋
        Add inList[middle] to inBalanceList
        firstHalf <- inList[:middle]
        makeBalanceSearchTree( firstHalf, inBalanceList)
        secondHalf <- inList[middle+1:]
        makeBalanceSearchTree( secondHalf, inBalanceList)
    else
        If len(inList) equals 1
            Add inList[0] to inBalanceList
        Elif  inList[0] is greater than inList[1]
            Add inList[0] to inBalanceList
            Add inList[1] to inBalanceList
        else
            Add inList[1] to inBalanceList
            Add inList[0] to inBalanceList
    Return inBalanceList
```

Se fil BalanceArray.py

Deloppgave b)

Algoritme: Gjøre om et heap til å printe et balansert søketre

Input: Et heap
Output: Printe et balansert søketre

```
Procedure MakeBalanceSearchTree( inHeap)
    newHeapQueRight <- []
    newHeapQueLeft <- []
    Length <- len(inHeap)
    Middle <- ⌊length + 1)/2⌋
    If length is less than 1
        Return
    i <- 0
    While i is less than length
        i = i + 1
        If i is equal to middle
            Print heapq.heappop(inHeapQue
        else
            If i is less than middle
                Element <- heapq.heappop(inHeapQue)
                Heapq.heappush(newHeapQueLeft, element)
            else
                Element <- heapq.heappop(inHeapQue)
                Heapq.heappush(newHeapQueRight, element)
    makeBalancesearchTree(newHeapQueLeft)
    makeBalancesearchTree(newHeapQueRight)
```

Se fil: BalanceHeap.py