# C 2.1

findMiddle(S)

if header.Next() == null then

return null

p <- header.Next()

q <- trailer.Previous()

while p ≠ q /\ p.Next() ≠ q do

q <- p.Previous()

p <- q.Next()

return p.value()

# C 2.2

S1<-Empty Stack

S2<-Empty Stack

**enqueue(val)**

if S1.size() = N - 1 then O(1)

throw FullQueueException

S1.push(val) O(1)

**Total running time O(1)**

**dequeue()**

if S2.isEmpty() then O(1)

while !S1.isEmpty() do O(n)

S2.push(S1.pop()) O(n)

if !S2.isEmpty() then O(1)

return S2.pop() O(1)

else

throw EmptyQueueException

**Total running time O(n)**

# C 2.3

S1<-Empty Queue

S2<-Empty Queue

Push(val)

if S1.size() = N - 1 || S2.size() = N - 1 then O(1)

throw FullStackException O(1)

if S1.isEmpty() O(1)

S2.enqueue(val) O(1)

else

S1.enqueue(val) O(1)

**Total running time O(1)**

Pop()

if S1.isEmpty() && S2.isEmpty then

return EmptyStackException O(1)

if S1.isEmpty() then O(1)

while S2.size() > 1 O(n)

S1.enqueue(S2.dequeue()) O(n)

return S2.dequeue() O(1)

else

while S1.size() > 1 O(n)

S2.enqueue(S1.dequeue) O(n)

return S1.dequeue() O(1)

**Total running time O(n)**

# A.

Algorithm removeDublicate(S)

for(int i=0; i< seq.size(); i++) O(n)

for(int j=i+1;j<seq.size();j++) O(n2)

if(seq.elemAtRank(i)==seq.elemAtRank(j)) O(1)

seq.removeAtRank(j) O(1)

**Total running time O(n2)**

# B.

Main

Sequence s = {1, 2, 3}

Sequence rs <- Sequence

rs.insertFirst(new Sequence() {})

Subsets(s, 0, rs)

Algorithm Sequence Subsets(Sequence s, index i, Sequence curr)

if(s.size() == i)

return rs.insertFirst(curr)

Subsets(s, i + 1, rs)

Subsets(s, i + 1, rs.insertLast(s.atRank(i)) )

# R 2.1

insertFirst(e)

Node n <- new Node(header, e, header.next)

header.next.previous <- n

header.next <- n

insertLast(e)

Node n <- new Node(trailer.previous, e, trailer)

trailer.previous.next <- n

Trailer.previous <- n

insertBefore(p,e)

if(p == 1)

insertFirst(e)

else if(p == size())

insertLast(e)

else

Node n <- header.next

for(int i = 0; i < p; i++)

n <- n.next

Node newNode <- Node(n.previous, e, n.next)

n.next.previous <- newNode

n.next <- newNode