Assignment 2

C-2.1 Describe, in pseudo-code, a link-hopping method for finding the middle node of a doubly linked list with header and trailer sentinels, and an odd number of real nodes between them. (Note: This method can only use link-hopping; it cannot use a counter.) What is the running time of this method?

Algorithm findMiddleNode(L)

Input : List L has odd number of nodes

Output : middle position of L

if(L.isEmpty())

return null

h <- L.head()

t <- L.tail()

while h != t do

h <- L.after(h)

t <- L.before(t)

return h

Assumed that L has n nodes, the running time is O(n/2)

C-2.2 Describe, in pseudo-code, how to implement the queue ADT using two stacks. What is the running time of the enqueue() and dequeue() methods in this case?

|  |  |
| --- | --- |
| s1 <- empty stack  s2 <- empty stack  enqueue(o)  if(s1.size() = n) then  throw QueueFullException  s1.push(o)    dequeue()  while !s1.isEmpty() do  s2.push(s1.pop())  o <- s2.pop()  while !s2.isEmpty() do  s1.push(s2.pop())    return o | O(1)  O(1)  O(1)  O(1)  O(1)  Total running time: O(1)  O(n)  O(n)  O(1)  O(n)  O(n)  O(1)  Total running time: O(n) |

C-2.3 Describe how to implement the stack ADT using two queues. What is the running time of the push() and pop() methods in this case?

|  |  |
| --- | --- |
| q1 <- empty queue  q2 <- empty queue  push(o)  if q1.size() = N then  throw StackFullException  q1.enqueue(o)    pop()  s <- q1.size()  for i <- 0 to s-1 do  q2.enqueue(q1.dequeue())  o <- q1.dequeue()  while !q2.isEmpty() do  q1.enqueue(q2.dequeue())    return o | O(1)  O(1)  O(1)  O(1)  O(1)  Total running time: O(1)  O(1)  O(n-1)  O(n-1)  O(1)  O(n)  O(1)    O(1)  Total running time: O(n) |

A. Design a pseudo code algorithm to take a Sequence and remove all duplicate elements from the Sequence. Is the algorithm the same for both a List or a Sequence? Explain. Analyze your algorithm twice, once assuming it is a Sequence and once assuming it is a List. Which ADT is a better choice for this problem, i.e., does one version have a better running time over the other?

|  |  |
| --- | --- |
| Algorithm removeDuplicateInSequence(S)  if S.isEmpty() then  throw SequenceEmptyException  p <- S.first()  while !S.isLast(p) do  q <- S.after(p)  if q.element() = p.element() then  S.remove(q)  else  p <- q | O(1)  O(1)  O(1)  O(n)  O(n)  O(n)  O(n)  O(n)  Total of running time: O(n) |
| Algorithm removeDuplicateInList(L)  if L.isEmpty() then  throw ListEmptyException    p <- L.first()  while !L.isLast(p) do  t <- p  while !L.isLast(t) do  q <- L.after(t)  if q.element() = t.element() then  L.remove(t)  t <- q  if !L.isLast(p) then  p <- L.after(p) | O(1)  O(1)  O(1)  O(n)  O(n)  O(n2)  O(n2)  O(n2)  O(n2)  O(n2)  O(n2)  O(n2)  Total of running time: O(n2) |

B. Describe a recursive algorithm for enumerating all subsets of the numbers {1,2,…,n}, i.e., the powerset of the elements of a Sequence; the result should be a Sequence containing Sequences. What is the running time of your method?

|  |  |
| --- | --- |
| Algorithm enumerateSubsets(L, n)  S <- {}  if n = 0 then  S.add({})  return S  if n = 1 then  S.add({})  S.add({1})  return S    if n > 1 then  L:= enumerateSubsets(L, n-1)  len <- S.size()  for i <- 0 to len do  temp <- {}  temp <- clone S[i]  temp.add(L[i])  S.add(temp)    return S | O(n)  O(n)  O(n)  O(n)  O(n)  O(n)  O(n)  O(n)  O(n)  O(n)  O(n)  O(n) + O(n(n+1)/2) = O(n2)  O(n2)  O(n2)  O(n2)  O(n2)  O(n)  Total of running time: O(n2) |

R-2.1 Describe, using pseudo-code, implementations of the methods insertBefore(p,e) , insertFirst(e), and insertLast(e) of the List ADT, assuming the list is implemented using a doubly-linked list.

|  |
| --- |
| insertBefore(p, e)  if p.isEmpty() then  throw InvalidOperationException    q <- p.prev  q.next <- e  e.prev <- q  e.next <- p  p.prev <- e |
| insertFirst(e)  insertBefore(first(), e) |
| insertLast(e)  insertBefore(last(), e) |

Optional if you have time:

C-2-5 Describe the structure and pseudo-code for an array-based implementation of the vector ADT that achieves O(1) time for insertions and removals at rank 0, as well as insertions and removals at the end of the vector. Your implementation should also provide for a constant-time elemAtRank method.