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?**

Answer:

|  |  |
| --- | --- |
| Algorithm findMiddle (L)  Input: List with an odd number of nodes  Output: Middle position of List  p:= L.first()  q:= L.last()  while p != q do  p = L.after(p)  q= L.before(q)  return p | 1  1  n  n  n  1  Running Time T(n) = O(n) |

**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?**

Answer:

S1 := Empty Stack

S2 := Empty Stack

|  |  |
| --- | --- |
| Algorithm enqueue (o)  If size () = n - 1 then  Throw FullQueueException  S1.push(o)  Algorithm dequeue ()  If S2.isEmpty() then  While !S1.isEmpty() do  S2.push(S1.pop())  If s2.isEmpty() then  Throw EmptyQueueException  Return S2.pop() | 1  1  1  Running Time T(n) = O(1)  1  n  n  1  1  1  Running Time T(n) = 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?**

Answer:

Q1:= Empty Queue

Q2:= Empty Queue

|  |  |
| --- | --- |
| Algorithm push(o)  if size() = n-1 then  Throw FullStackException  Q1.enqueue(o)  Algorithm pop()  If Q2.isEmpty() then  While !Q1.isEmpty() do  Q2.enqueue(Q1.dequeue())  If Q2.isEmpty() then  Throw EmptyQueueException  Return Q2.dequeue() | 1  1  1  Running Time T(n) = O(1)  1  n  n  1  1  1  Running Time T(n) = O(n) |

1. **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?**

Answer: **The algorithm is the same for both a List and a Sequence**

|  |  |
| --- | --- |
| Algorithm remove\_duplicates(S)  If S.size() = 0 then  Return S  U:= EmptySequeue  p:= S.first()  insert\_unique (U, p.element())  While !S.isLast(p) do  p= S.after(p)  insert\_unique (U, p.element())  Return U  Algorithm insert\_unique (S, e)  If S.isEmpty() then  S.addLast(e)  Else  p:= S.first()  is\_valid:= true  if p.element() = e then  is\_valid = false  while is\_valid and !S.isLast(p) do  p=S.after(p)  if p.element() = e then  is\_valid = false  if is\_valid then  S.addLast(e) | 1  1  1  1  n  n  n  n2  1  Running Time T(n) = O(n2)  1  1  1  1  1  1  1  n  n  n  n  1  1 |

1. **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?**

Answer:

|  |  |
| --- | --- |
| Algorithm enumer\_subsets(S)  If S.isEmpty() then  R:= EmptySequeue  R.addLast(S)  Return R  e:=S.remove(S.first())  PS:= enumer\_subsets(S)  R:=copy(PS)  Return combine(e,PS,R) |  |

**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.**

Answer:

|  |
| --- |
| Algorithm insertBefore(*p,e*)  n:=newNode(e)  q:=p.prev  q.next=n  n.next=p  p.prev=n  n.prev=q |
| Algorithm insertFirst(*e*)  n:= newNode (e)  q:=header.next  header.next=n  n.next=q  p.prev=n  n.prev=header |
| Algorithm insertLast(*e*)  n:= newNode (e)  q:=trailer.prev  trailer. prev =n  n.prev=q  q.next=n  n.next= trailer |

**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.**

|  |
| --- |
| Algorithm insertAtRankZero(o)  If V.size() = n-1 then  Throw FullVectorException  f:=(f-1+n) mod n  V[f]=o |
| Algorithm removeAtRankZero()  If V.isEmpty() then  Throw EmptyVectorException  f:=(f+1) mod n |
| Algorithm insertAtRankEnd(o)  If V.size() = n-1 then  Throw FullVectorException  V[r]=o  r:=(r+1) mod n |
| Algorithm removeAtRankEnd()  If V.isEmpty() then  Throw EmptyVectorException  r:=(r-1+n) mod n |
| Algorithm elemAtRank (r)  i:=(f+r) mod n  return V[i] |