Chanh Dao Le – 986178

Assignment 2

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.

```
Algorithm insertBefore(p, e)
        Input: Position p, new element e
        Output: New Position q
        q <- new Position
        q.element <- e
        q.prev <- p.prev
        q.next <- p
        p.prev.next <- q
        p.prev <- q
        return q
Algorithm insertFirst(e)
        Input: New element e
        Output: New Position q
        q <- new Position
        q.element <- e
        q.next <- header.next
        q.prev <- header
        header.next.prev <- q
        header.next <- q
        return q
Algorithm insertLast(e)
        Input: New element e
        Output: New Position q
        q <- new Position
        q.element <- e
        q.next <- trailer
        q.prev <- trailer.prev
        trailer.prev.next <- q
        trailer.prev <- q
        return q
```

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: A doubly linked list L with an odd number of real nodes between header and trailer
      Output: The middle node
      p <- L.header
      q <- L.trailer
      while p ⌜= q do
            p = first.next
            q = last.prev
```

return p

The running time of this algorithm 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?

```
Algorithm enqueue(e)
        Input: Element e
        stack1.push(e)
Algorithm dequeue()
        Output: Element at the front of the queue
        if stack1.isEmpty() and stack2.isEmpty() then
                throw EmptyQueueException
        else
                if stack2.isEmpty() then
                        while ｢stack1.isEmpty() do
                                stack2.push(stack1.pop())
                p <- stack2.pop()
                return p
```

The running time for enqueue algorithm is O(1)

The running time for dequeue algorithm is 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?

```
Algorithm push(e)
        Input: Element e
        queue1.enqueue(e)
Algorithm pop()
        Output: Element at the top of the stack
        if queue1.isEmpty() and queue2.isEmpty() then
                throw EmptyStackException
        else
                for i <- 1 to queue1.size() - 1 do
                        queue2.enqueue(queue1.dequeue())
                p <- queue1.dequeue()
                tmp <- queue1
                queue1 <- queue2
                queue2 <- tmp
                return p
```

The running time of push method is O(1)

The running time of pop method is O(n)

C-2-4 Describe a recursive algorithm for enumerating all permutations of the numbers {1, 2,..., n}. What is the running time of your method?

```
Algorithm swap(s, i, j)
        Input: Sequence s, indexes i and j to swap data in s
        Output: Sequence with data swapped
        temp <- s[i]
        s[i] <- s[j]
        s[j] <- temp
Algorithm permute(s, i)
        Input: Sequence s, and index i of a position in s whose value is 0 initially
        Output: A sequence of permutations of s
        permutations <- new Sequence
```

```
if i < s.size() – 1 then
        for j from i to s.size() do
                if j ⌜= I then
                        swap(s, i, j)
                result <- permute(s, i + 1)
                for all permutation in result do
                        permutations.insertLast(permutation)
                if j ⌜= I then
                        swap(s, i, j)
else
        permutations.insertLast(s)
return permutations
```

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 elemAtRank(r)
        Input: Rank r
        Output: Element at rank r
        return V[(f + r) mod N]
Algorithm insertAtRank(r, o)
        Input: Rank r, and object o to be inserted at rank r
        If size() = N – 1 then
                throw FullVectorException
        start <- (f + r) mod N
        if start = f do                                 // insert at rank 0
                f <- (f – 1) mod N
                V[f] <- o
        else if start = l then                          // insert at the end
                V[l] <- o
                l <- (l + 1) mod N
        else then                                       // insert somewhere between f and l
                i <- l
                while i ⌜= start do
                        V[i] <- V[l - 1]
                        i <- i – 1
                        if i < 0 then
                                i <- i + N
                V[start] <- o
                l <- (l + 1) mod N
Algorithm removeAtRank(r)
        Input: Rank r
        Output: Removed element at rank r
        If isEmpty() then
                throw EmptyVectorException
        start <- (f + r) mod N
        o <- V[start]
        if start = f then                       // remove at rank 0
                f <- (f + 1) mod N
        else if start = l – 1 then              // remove at the end
                l <- (l – 1) mod N
```

```
else then                                   // remove somewhere between f and l
        i <- start
        end <- l – 1
        if end < 0 then
                end <- end + N
        while i ⌜= end do
                next <- (i +1) mod N
                V[i] <- V[next]
                i <- next
return o
```