IMPERATIVE PROGRAMMING 2016 MOISE GABRIEL

QUESTION 4

(a)

```
// Swaps the values of m(i) and m(j)
// Cost = 4
def swap (i : int, j : int) : Unit= {
    var aux = m(i)
    m(i) = m(j)
    m(j) = aux
}
// exchange the values from m[i .. i+m) with m[j .. j+m)
// cost = 4 * m
def blockswap (i: int, j: int, m: int) : Unit= {
    for (q <- 0 until m) swap (i+q, j+q)
}
```

(b)

```
// reverse the order of the values in m[i..i+m)
// Cost = 2 * m
def reverse (i : int, in : int) : Unit= {
    var m = in - i
    for (q <- 0 to (n-1)/2) swap (i+q, i+m-1-q)
}
```

(c)

```
// rotate (i, n, k) shifts m[i..i+m) by k positions to the right
// cost = 4 * m
i) def rotateRev (i : int, m : int, k : int) : Unit= {
       reverse (i, i+m-k) // cost = 2 * (n-k)
       reverse (i+m-k, i+m) // cost = 2*k
       reverse (i, i+m) // cost = 2*m
   }
```

```
(ii) // Cost = O(k*n) because of the recursion ?
    def rotateBlRec (i: int, n: int, k: int) : Unit = {
        if (n-k==k) blockswap(i, i+n-k, k)
        else if (n-k, k) {blockswap(i, i+n-k, k); rotateBlRec (i+k, n-k, k)}
        else {blockswap (i, i+n-k, n-k); rotateBlRec( i+n-k, k, n-k)}
    }

(iii) // Cost = O(k*n)?
    def rotate Bl ( i1: int, n1: int, k1: int) : Unit = {
        var i=i1; var n=n1; var k = k1
        while (n-k != k)
        { if (n-k>k) {blockswap (i, i+n-k, k); i= i+k; n=n-k}
          else {blockswap (i, i+n-k, n-k); i= i+n-k; var aux= n; n=k; k=aux-k}
        }
        blockswap (i, i+n-k, k)
    }

(iv) // Cost = 2*k*n
    def rotateRep (i: int, n: int, k: int) : Unit = {
        for (q <- 1 to k)
        { var t= m(i+n-1) // cost = 1
          var j = i+n-1
          while (j>i) {m(j) = m(j-1) ; j-=1} // cost = 2*n-2
          m(i) = t // cost = 1
        }
    }
```

(d) it seems from the costs of the several rotate functions that rotateRev is the most efficient one.