Assignment 2 Problem 5

In this question, we generalize *quickSelect1* to work on two input arrays. Let the resulting algorithm be called *quickSelect2Arrays(A,B,k)*. Arrays A and B are of size n and m, respectively, and $k \in \{0, 1, ..., n + m - 1\}$. Algorithm *quickSelect2Arrays(A,B,k)* should return the item that would be in $C[k]$ if C was the array resulting from merging arrays A and B and C was sorted in non-decreasing order.

Your algorithm *quickSelect2Arrays(A,B,k)* must be in-place, i.e. only $O(1)$ additional space is allowed. Briefly and informally (one or two sentences) argue that the time complexity of your algorithm is the same as of *quickSelect1*, i.e. $O(v)$ in the average case where v is the total number of elements in A and B, i.e. $v = n + m$.

Solution:
The pseudocode for quickSelect2Arrays(A,B,k):

```
quickSelect2Arrays(A,B,k)
A,B: two arrays with length n,m respectively
k: the element of this postion that we want in the merging arrays.
p ← choose-pivot(A,B)
i ← partition(A,B,p)
if i = k then
    if i < n then
        return A[i]
    else
        return B[i-n]
else if i > k then
    if i ≤ n then
        return quick-select1(A[0,1,...,i-1],k)
    else then
        return quickSelect2Arrays(A,B[0,1,...,i-n-1],k)
else if i < k then
    if i ≥ n-1 then
        return quick-select1(B[i-n+1,...,n-1],k-(i+1))
    else then
        return quickSelect2Arrays(A[i+1,...,n-1],B,k-(i+1))
```

```
choose-pivot(A,B)
A,B: two arrays with length n,m respectively
if B empty then
    return A.size()-1
else then
```

```
        return A.size()+B.size()-1


partition(A,B,p)
A,B: array of size n,m respectively
p: integer such that 0≤p≤m+n-1
if p < n then
        swap(A[p], B[m-1])
else then
        swap(B[p-n], B[m-1])
i←-1, j←m+n-1, v←B[m-1]
loop
        do i←i+1 while (i<n and A[i]<v) or (i≥n and B[i-n]< v)
        do j←j-1 while ((j≥i) and ((j<n and A[j]>v) or (j≥n and B[j-n]>v)))
        if i≥j then break
        else if i<n and j<n swap(A[i],A[j])
        else if i<n and j≥n swap(A[i],A[j-n])
        else if i≥n and j<n swap(B[i-n],A[j])
        else if i≥n and j≥n swap(B[i-n],B[j-n])
end loop
if i<n
        swap(B[m-1], A[i])
else
        swap(B[m-1], B[i-n])
return i
```

The previous algorithm is just seems A and B are a array that been merged together but without creating a new array to ensure it is in-place. We achieve it by let the index number i to be i-A.size() if i exceed the size of A. Since we are assuming they are a single array, and do everything same with quick-select1 except the index(with a constant time conversion). Therefore, the time complexity of the algorithm is the same as of quickSelect1(ie. $O(n)$)).