Name: Wenhao Ge

1. (A)

```
find-missing (int[] A):
    left = 0
    right = A.length - 1
    while left <= right:
        mid = left + (right - left)/2;
        if ( A[mid] != mid+1 && A[mid-1] == mid ):
                return mid + 1;
        else if A[mid] == mid+1 :
                left = mid-1;
        else
            right = mid+1
    return -1
```

it's run time complexity : O(logn)

)

```
find-missing (int[] A):
    sum1 = (1+(n+1))n / 2
    sum2 = the sum of all values in array A
    return sum1 - sum2
```

un time complexity : O(n)

```
sort (int[] A):
    pointerA = 0
    pointerB = 0
    while pointerB < A.length:
        if ( A[pointerB] < 0):
            exchange values at pointerA, pointerB
            pointerA++
        pointerB++
```

run time complexity : O(n)
n means the length of the array

3.A

```
Merge(A, p, q, r)
n1 = q - p + 1
n2 = r - q
if n1 ≤ k
    helper(A, p, q, r)
else
    Let L[1...k+1] and R[1...k+1]
    be new array
    for i = 1 to k
        L[i] = A[q - k + i - 1]
    for j = 1 to k
        R[j] = A[q + j]
    i = 1
    j = 1
    for k = q - k to q + k
        if L[i] ≤ R[j]
            A[k] = L[i]
            i = i + 1
        else
            A[k] = R[j]
            j = j + 1
```

```
helper(A, p, q, r):
n1 = q - p + 1
n2 = r - q
Let L[1...n1+1] and R[1...n2+1]
be new arrays
for i = 1 to n1
    L[i] = A[p + i - 1]
for j = 1 to n2
    R[j] = A[q + j]
L[n1+1] = ∞
R[n2+1] = ∞
i = 1
j = 1
for k = p to r
    if L[i] ≤ R[j]
        A[k] = L[i]
        i = i + 1
    else
        A[k] = R[j]
        j = j + 1
```

(b) $T(n) = 2T(\frac{n}{2}) + O(k) = 2T(\frac{n}{2}) + O(1)$,

We can use a function $f(n) = n$, we can know that for $n > 2$, $f(n) < T(n) < 2 \cdot f(n)$, so $O(n) = n$