1. A)

```
Algorithm FindThreeOccurences (A, x) :
    Input integer x, integer array A
    Output boolean
    count = 0
    foreach element : A
        if element == x:
            count ← count + 1
        if count > 3:
            return false
    if count == 3:
        return true
    else
        return false
```

B)

The time complexity of my algorithm is O(n), in which n is the size of the array A. The worst case is to have to check all the elements of the array to count the occurrences of x.

C)

The space complexity of my algorithm is O(1) since we're using a fixed number of spaces. The size of input array does not affect, we just need one space for count.

2.

```
Algorithm binarySortedSearch (A, target, low, high):
```

Input: sorted integer array A, integer target, integer low, integer high

Output: Boolean

```
if low > high:
    return false

mid = (low + high) /2

if target == A[mid]:
    return true
else if target < A[mid]:
    return binarySortedSearch (A, target, low, mid - 1)
else:
    return binarySortedSearch (A, target, mid + 1, high)
```

3. A)

$f(n) \geq C\ g(n)$ for $n \geq n0$

$f(n) = n^{22} \log n + n^7$, $g(n) = n^6 \log n$

$n^{22} \log n + n^7 \geq n^6 \log n$

$n^{22} \log n + n^7 - n^6 \log n \geq 0$

$(n^{16}-1)\ n^6 \log n + n^7 \geq 0$

if $x > 0$ and $y > 0 \rightarrow x + y > 0$

$x = (n^{16}-1)\ n^6 \log n$

$(n^{16}-1)\ n^6 \log n > 0$

$(n^{16}-1) > 0$ (for all $n > 1$)

$n^6 > 0$ (for all $n > 0$)

$\log n > 0$ (for all $n > 1$)

$y = n^7$

$n^7 > 0$ (for all $n > 0$)

B)

$C' g(n) \leq f(n) \leq C'' g(n)$ for $n \geq n0$

$f(n) = 10^7n^5 + 5n^4 + 9000000n^2 + n$, $g(n) = n^7$

$10^7n^7 \leq 10^7n^5 + 5n^4 + 9000000n^2 + n \leq$

C)

$f(n) \geq C\, g(n)$ for $n \geq n0$

$f(n) = n!$, $g(n) = n^n$

$n! \geq n^n$ (for all $n > 0$)

The inequality does not hold since n! grows slower than n^n as n gets very large.

D)

E)

F)

4. A)

Algorithm ReverseTwoConsecutive (arr) :

```
n = length (arr)
mid = n /2
for i from 0 to mid -1 by 2:
    swap arr [i] and arr [i +1]
for i from mid to n-1 by 2 :
    arr [i] = arr [i] + arr [i-1]
```

B)

Time complexity of my algorithm is O(n) since both loops only iterate through half of the array thus n/2 + n/2 = n.

C)

The space complexity of my algorithm is O(1) since there is a constant space usage.