Question 1:
A. Returns number of occurrences of an object (target) in a collection
B. If the array ends with a run of the maximally occurring string that string won’t register as the max since it’s not different than what came before it, hence a check after the loop is necessary.
C. sortMax is $O(n \log n) + O(n)$ which is $O(n \log n)$ since sorting is $O(n \log n)$
setMax is, in worst case (all different) $O(n^2)$ since the treest will have $n$ elements and each one generates an $O(n)$ call of Collections.frequency. On average, say half the elements are different, we still get $O(n^2)$. In general the complexity is $O(mn)$ where $m$ number of distinct elements and $n$ is total number of elements. As long as $m > \log n$ netMax is worse.
D. A big array in which all elements are equal will be $O(n)$ for setMax since only one element in treest. But sortMax is still $O(n \log n)$

Question 2:
A. $n+n = O(n)$
B. $1 + 2 + 3 + \ldots + n-1 = (n-1)n/2$ which is $O(n^2)$
C. You can double $k \log(n)$ times before reaching $n$ when $k$ starts at 1, e.g., 1,2,4,8,16,32,64,\ldots,512,1024 so $O(\log n)$
D. $O(\sqrt{n})$ since $p$ is incremented until it passes the square root of $n$, and this increment can’t happen more than $\sqrt{n}$ times

Question 3:
A. 6-list is 21
100-list is 100*101/2 = 50*101 = 5050
B. Removing all N leaves an N-1 list which has $1 + 2 + \ldots + N-1$ elements = $(N-1)N/2$ which is $O(N^2)$
or, starting with $O(N^2)$ in a N-list, we remove N elements and $N^2 - N$ is still $O(N^2)$
C. We have $1 + 2 + \ldots + N/2$ elements left which is $N^2/(N+2)/2 = O(N^2)$ ignoring coefficients and trailing terms
D. makeListZ is N-list, notice that front is returned and it’s the front of a recursive call, and that the N-nodes are added *to the end of this list*
makeListY is reverse list. The value returned, first,

Question 4:
A. Two loops, each is $O(n)$ for an n-node list. Creating the array is also $O(n)$, but we still have $O(n)$ since $n+n+n=3n=O(n)$
B. Intuitively adding to an ArrayList and then calling toArray would be better since the ArrayList traversal would be faster than traversing a linked-list. In practice this isn’t the case and I don’t actually know how to make it faster.
C. 
```java
    public Node arrayToList(String[] array){
        Node front = null;
        for(int k=0; k < array.length; k++){
            front = new Node(array[k],front);
        }
        return front;
    }
```
D. 
```java
    public Node addInOrder2(Node list, String s) {
        if (list == null) return new Node(s,null);
        if (s.compareTo(list.info) <= 0) return new Node(s,list);
        Node front = list;
        while (list.next != null){
            if (s.compareTo(list.next.info) > 0){
                break;
            }
            list = list.next;
        }
        // new node comes after list
        list.next = new Node(s,list.next);
        return front;
    }
```
public Node addInOrder3(Node list, String s) {
    if (list == null) return new Node(s, null);
    if (s.compareTo(list.info) <= 0) return new Node(s, list);
    Node front = list;
    Node prev = list;
    list = list.next;
    while (list != null && s.compareTo(list.info) > 0){
        prev = list;
        list = list.next;
    }
    // new node comes after prev, before list
    prev.next = new Node(s, list);
    return front;
}