1. Sorting Algorithms: Step-by-Step Analysis
a) Selection Sort

Selection sort works by repeatedly finding the smallest remaining value in the unsorted portion of the list and swapping it into its correct position. Each pass places one element permanently.

Given the list:
4, 6, 2, 10, 9

After 1st iteration: The smallest value (2) is selected and swapped to the front
→ 2, 6, 4, 10, 9

After 2nd iteration: The next smallest value (4) is placed in position
→ 2, 4, 6, 10, 9

After 3rd iteration: The next smallest value (6) is already correctly placed
→ 2, 4, 6, 10, 9

After 4th iteration: The next smallest value (9) is swapped into position
→ 2, 4, 6, 9, 10

At this point, the list is fully sorted.

b) Insertion Sort

Insertion sort builds a sorted section of the list one item at a time by inserting each new element into its proper position relative to the already-sorted portion.

Given the list:
4, 6, 2, 10, 9

After 1st iteration: 6 is compared to 4 and remains in place
→ 4, 6, 2, 10, 9

After 2nd iteration: 2 is inserted before both 4 and 6
→ 2, 4, 6, 10, 9

After 3rd iteration: 10 is already in the correct position
→ 2, 4, 6, 10, 9

After 4th iteration: 9 is inserted before 10
→ 2, 4, 6, 9, 10

2. Binary Search Requirement

In order to use a binary search, the list must be sorted beforehand. This is necessary because binary search relies on repeatedly eliminating half of the remaining list based on comparisons, which is only possible if the data is in order.

3. Recursive Method Analysis

The method prints a statement before the recursive call and another after the recursive call completes, meaning the output reflects both descending calls and ascending returns.

```
public void ct(int n) {
   System.out.println("Starting " + n);
   if (n > 0) {
      ct(n/3);
      System.out.println("Middle " + n);
   }
}
```

a) Output when ct(13) is called
Starting 13
Starting 4
Starting 1
Starting 0
Middle 1
Middle 4
Middle 13

b) Output when ct(3) is called
Starting 3
Starting 1
Starting 0
Middle 1
Middle 3

c) Output when ct(0) is called
Starting 0

This occurs because the base case (n > 0) is false, so no recursive call is made.

7. Stability of Sorting Algorithms

A sorting algorithm is considered stable if items with equal key values maintain their original relative order after sorting.

Selection Sort: Not stable
Selection sort swaps elements without considering whether equal values change order.

Example:
A(5), B(3), C(5)
→ After selection sort: B(3), C(5), A(5)
The two items with key value 5 have changed order.

Merge Sort: Stable
Merge sort preserves the order of equal elements during the merge process.

True / False with Justification
a) Sorting always puts items from low to high

False. Sorting can also be done from high to low depending on the algorithm or criteria used.

b) One measure of the efficiency of a sorting algorithm is its speed

True. Time complexity is a key factor when evaluating algorithm efficiency.

c) Selection sort is more efficient than insertion sort

False. Insertion sort is generally more efficient for small or nearly sorted lists.

d) A method can call itself

True. This is known as recursion.

e) Merge sort is more efficient than selection sort for large arrays

True. Merge sort has a better time complexity for large datasets.

f) A binary search is used to sort a list

False. Binary search is used to locate an item, not to sort data.

g) A binary search starts by examining the last item

False. Binary search begins by examining the middle item.

h) The more statements required, the more efficient the algorithm

False. Efficiency depends on how performance scales, not on the number of statements.

i) An interface can be used as a data type

True. Objects can be referenced using an interface type.

j) A recursive solution with no base case results in infinite recursion

True. Without a base case, the method never stops calling itself.

k) Insertion sort divides a list into halves and merges them

False. This description applies to merge sort, not insertion sort.