Which of the following is the best description of selection sort?

A. Once a value is placed in the sorted part, it will never move again
B. All values in the sorted part are always less than or equal to all values in the unsorted part
C. Both A and B are true
D. Neither A or B is true

Answer: C
Both A and B are true

We saw that the time complexity of selection sort was $O(n^2)$, what is the asymptotic space complexity for auxiliary storage? That is how much additional memory do we need beyond the original list?

A. O(1)
B. O(log n)
C. O(n)
D. O(n log n)
E. $O(n^2)$

Answer: A
Space complexity is the amount of memory needed by an algorithm. For selection sort, beyond the original list, we only need to store the current minimum value and index. Thus the additional storage is O(1).

What is the best-case time complexity of insertion sort

```
def insertion_sort(a_list):
    for i in range(1,len(a_list)):
        value = a_list[i]
        index = i
        while index > 0 and a_list[index-1] > value:
            a_list[index] = a_list[index-1]
            index -= 1
        a_list[index] = value
```

A. O(1)
B. O(log n)
C. O(n)
D. O(n log n)

Answer: C

In the best case the list is sorted (or all values are within a constant distance of being sorted). In that case we only have a fixed number of operations in the body of the outer loop, i.e., the inner while loop doesn't execute because a_list[index-1] <= value, or only executes a fixed number of times. In that case f(n)=n(c) or O(n)!

Which of the following is the best description of merge sort?

A. Before the final merge, all values in “left” are less than or equal to all values in "right"
B. Before the final merge, all values in “left” are sorted and all values in “right” are sorted
C. Both A and B are true
D. Neither A or B are true

Answer: B
Since merge sort splits the list in half and sorts the left and right half independently, before the final merge each half is sorted, but there is no relationship between the values in each half.