**Qs on Big Oh.**

You may assume, in each case, that the most efficient algorithm is used.

Explain your answer concisely in each case.

New question) Look at the java api documentation for some of the Collection classes that we have used. Can you see big Big Oh values specified for some of the methods?

1) Suppose an algorithm requires a total of 3*n*3 + 2*n*2 – 3*n* + 4 visits. In big-Oh notation, the total number of visits is of what order?

a) *n*2 \* *n*2

b) *n*2

c) *n*6

d) *n*3

In Big-Oh notation, we focus on the highest order term, which is n³ in this case. The constants and lower order terms are ignored.

2) In big-Oh notation, when we consider the order of the number of visits an algorithm makes, what do we ignore?

I power of two terms

II the coefficients of the terms

III all lower order terms

a) I

b) II

c) I and II

d) II and III

we ignore the coefficients of the terms (II) and all lower order terms (III) when determining the Big-Oh notation

3) In big-Oh notation, suppose an algorithm requires an order of *n*3 element visits. How does doubling the number of elements affect the number of visits?

a) It doubles the number of visits.

b) It quadruples the number of visits.

c) It triples the number of visits.

d) It number of visits goes up by a factor of eight.

The number of visits becomes (2n)³ = 8n³, which is eight times the original.

4) What is the efficiency of adding an element exactly in the middle of a linked list? An ArrayList?

Linked List: O(n) (need to traverse to the middle). ArrayList: O(n) (need to shift elements to make space)

5) Using big-Oh notation, what is the cost of adding an element to an ArrayList as the second-to-last element?

O(n) we shift elements to make space for the new element.

6) In Big-Oh notation, selection sort is a(n) \_\_\_\_ algorithm.

a) *O*(*n*2)

b) *O*(1)

c) O (log *n)*

d) *O*(log *n*2)

Selection sort has a nested loop structure, resulting in quadratic time complexity

7) When the size of an array increases by a factor of 100, the time required by selection sort increases by a factor of \_\_\_\_.

a) 2,000

b) 5,000

c) 10,000

d) 12,000

If the size increases by 100, the time complexity scales with the square of that factor

8) In the textbook, we found that the number of element visits for merge sort totaled   
*n* + 5*n* log2 *n*. Which of the following is the appropriate big-Oh notation for merge sort?

a) 5*n* log2 *n*

b) *n* + log2 *n*

c) *n* + 5*n*

d) *n* log2 *n*

The dominant term is 5n log₂ n, which dictates the Big-Oh notation

9) Suppose we maintain a linked list of length *n* in sorted order. What would be the big-Oh notation for the add operation?

a) *O*(1)

b) *O*(*n*)

c) *O*(*n* log2 *n*)

d) *O*(*n*2)

traverse the entire list

10) Suppose we maintain two linked lists of length *n* in sorted order. What would be the big-Oh notation for the creating a third list, which included only elements common to both lists?

a) *O*(1)

b) *O*(*n*) traverse both lists simultaneously, resulting in linear time complexity

c) *O*(*n* log2 *n*)

d) *O*(*n*2)

11) Suppose we maintain two linked lists of length *n* in random element order. What would be the big-Oh notation for the creating a third list that includes only elements common to both lists, without sorting the first two lists?

a) *O*(1)

b) *O*(*n*)

c) *O*(*n* log2 *n*)

d) *O*(*n*2) for each element in one list, you may need to traverse the other list, leading to quadratic time complexity

12) Suppose we maintain a linked list of length *n* in random element order. What would be the big-Oh notation for an algorithm that prints each list element and the number of times it occurs in the list (without sorting the list)?

a) *O*(1)

b) *O*(*n*) traverse the list once to print each element

c) *O*(*n* log2 *n*)

d) *O*(*n*2)

13) Suppose we maintain a linked list of length *n* in random element order. What would be the big-Oh notation for printing out those elements which occur exactly once in the list (without sorting the list)?

a) *O*(1)

b) *O*(*n*) traverse the list once to count occurrences, then traverse again to print the elements that occur once

c) *O*(*n*log2*n*)

d) *O*(*n*2)

14) Suppose we maintain a linked list of length *n* in sorted order. What would be the big-Oh notation for printing out those elements that occur exactly once in the list?

a) *O*(1)

b) *O*(*n*) traverse the list in orderto print the elements that occur once, taking advantage of the sorted

c) *O*(*n* log2 *n*)

d) *O*(*n*2)