Chapter-23

1.What is the time complexity for an insertion sort?

**A. The time complexity for an insertion sort is O(n^2).**

B. The time complexity for an insertion sort is O(n/2).

C. The time complexity for an insertion sort is O(n).

D. The time complexity for an insertion sort is O(n2).

**2.** If a list is already sorted, how many comparisons will the insertionSort method perform?

**A. n - 1 times.**

B. n times

C. n2 times

D. 0 times

**3.** What is the time complexity for a bubble sort?

A. The time complexity for a bubble sort is O(n/2).

**B. The time complexity for a bubble sort is O(n^2).**

C. The time complexity for a bubble sort is O(n).

D. The time complexity for a bubble sort is O(n-1).

4. If a list is already sorted, how many comparisons will the bubbleSort method perform?

**A. n - 1 times.**

B. n times.

C. n2 times.

D. n /2 times.

5. What is the time complexity for a merge sort?

A. The time complexity for a merge sort is O(n-1).

A. The time complexity for a merge sort is O(n/2).

A. The time complexity for a merge sort is O(n2).

**A. The time complexity for a merge sort is O(nlogn).**

6. What is the time complexity for a quick sort?

**A. The time complexity for a quick sort is O(n^2) in the worst case and O(nlogn) in the average case.**

B. The time complexity for a quick sort is O(n2) in the worst case and O(nlogn) in the average case.

C. Both A and B

D. None of the above.

7. Why is quick sort more space efficient than merge sort?

1. Quick sort needs to create temporary arrays, while merge sort does not need temporary arrrys.
2. **Quick sort does not need to create temporary arrays, while merge sort needs temporary arrrys.**
3. Quick sort needs to create temporary arrays, while merge sort needs temporary arrrys.
4. None of the above

8. Which of the following statements are wrong?

1. Heap<Object> heap1 = **new** Heap<>();
2. Heap<Number> heap2 = **new** Heap<>();
3. Heap<BigInteger> heap3 = **new** Heap<>();
4. Heap<Calendar> heap4 = **new** Heap<>();
5. Heap<> heap5 = **new** Heap<>();
6. Lines 2 and 3 are wrong
7. Lines 1 and 3 are wrong
8. Lines 2 and 5 are wrong
9. **Lines 1 and 2 are wrong**

9. What is the return value from invoking the remove method if the heap is empty?

A. The return value will not be null.

**B. The return value will be null.**

10. What is the time complexity of inserting a new element into a heap and what is the time complexity of deleting an element from a heap?

1. O(logn) for only insertion.
2. O(logn) for only deletion.
3. O(logn) for none of insertion and deletion.
4. **O(logn) for both insertion and deletion.**

11. Can you sort a list of strings using a bucket sort?

1. Yes, Bucket sort is suitable for sorting strings.
2. **No, Bucket sort is not suitable for sorting strings.**

12. Suppose list is an instance of MyList, can you get an iterator for list using list.iterator()?

**A. Yes. Because MyList is Iterable.**

B. No. Because MyList is not Iterable.

13. Can you create a list using new MyList()?

**A. No. Because MyList is an interface.**

A. Yes

14. What are the limitations of the array data type?

1. **An array is a fixed-size data structure. Once an array is created, its size cannot be changed.**
2. An array is not a fixed-size data structure.
3. Both A and B are true.
4. None of the above.

15. If a linked list does not contain any nodes, what are the values in head and tail?

A. head and tail are not null.

B. tail is null.

C. head is null.

**D. head and tail are null.**

16. If a linked list has only one node, is head == tail true? List all cases in which head == tail is true.

A**. Yes. When the list is empty, head == tail is also true.**

A. No. When the list is empty, head == tail is also false

17. What is the time complexity of the addFirst(e) and removeFirst() methods in MyLinkedList?

1. **O(1)**
2. O(2)
3. O(0)
4. O(n)

18. What would be the time complexity for the size() method if the size data field is not used in MyLinkedList?

A. O(n/2)

B. O(n2)

C. O(n-1)

**D. O(n)**

19. What is a priority queue?

1. In a priority queue, elements are not assigned with priorities. When accessing elements, the element with the highest priority is removed first.
2. In a priority queue, elements are assigned with priorities. When accessing elements, the element with the highest priority is not removed first.
3. **In a priority queue, elements are assigned with priorities. When accessing elements, the element with the highest priority is removed first.**
4. None of the above

**20.** What method is defined in the java.lang.Iterable<E> interface?

1. The iterator() method is defined in the java.lang.List interface.
2. The iterator() method is defined in the java.lang.Set interface.
3. The iterator() method is defined in the java.lang.HashSet interface.
4. **The iterator() method is defined in the java.lang.Iterable interface.**

**21.** What is the benefit of being a subtype of Iterable<E>?

1. Being a subtype of Iterable, the elements of the container cannot be traversed using a for-each loop.
2. **Being a subtype of Iterable, the elements of the container can be traversed using a for-each loop.**
3. None of the above.

**22.** Every internal node in a Huffman tree has two children. Is it true?

1. **Yes.**
2. No

**23.** If the Heap class in line 50 in Listing 25.9 is replaced by java.util.PriorityQueue, will the program still work?

1. **Yes, except that you also have to change heap.getSize() to heap.size().**
2. No