|  |  |  |
| --- | --- | --- |
| **APCS Exposure Java** | **Exercises 18.06-12** | **Date:** |
| **Name:** | | **Period:** |

1. In a worst case scenario, how many swaps would a Bubble Sort make if there were 10,000 elements in an array?

2. Look at the **selectionSort** methof inprogram ***Java1812.java***.

What does the variable **smallest** actually store?

3. What makes the *Selection Sort* more efficient that the *Bubble Sort*?

4. Does an *Insertion Sort* use a **swap** method?

5. List the steps of the Insertion Sort algorithm.

6. If finding the correct page in a 2000 page phone book can be done in *11* tries using the *Binary Search*, how many tries would it take for a 4000 page phone book?

7. For a *Binary Search* to work, what must be true of the list?

8. The *Binary Search* has variables named **lo** and **hi**. What does it mean if **hi** becomes less than **lo**?

9. Which 4 sorts are considered to be remarkably fast?

10. Which sort is ideal for combining to sorted lists into a single sorted list?

11. Refer to your answer to the previous question. Can this sort be used to sort random data?

12. Execution efficiency is concerned with what?

13. Is the ***TimeTest*** class a standard Java class?

14. Look at program ***Java1817.java*** and its chart in **Figure 18.39**.

Estimate the Process Time for a Data Quantity of 8,000,000.

15. Refer to the previous question.

Estimate the Process Time for a Data Quantity of 12,000,000.

16. Look at program ***Java1818.java*** and its chart in **Figure 18.41**.

Estimate the Process Time for a Data Quantity of 80,000.

17. Refer to the previous question.

Estimate the Process Time for a Data Quantity of 120,000.

18. Why is the *Selection Sort* is faster than to the *Bubble Sort.*

19. Look at program ***Java1819.java*** and its 2 charts in **Figure 18.43**.

Estimate the Selection Sort Process Time for a Data Quantity of 80,000.

20. Refer to the previous question.

Estimate the Process Time for a Data Quantity of 120,000.

21. Look again at the 2 charts in **Figure 18.43**.

The Selection Sort does execute faster than the Bubble Sort, but are their *behaviors* the same?

*.*

22. Look at program ***Java1820.java*** and its 3 charts in **Figure 18.45**.

Estimate the Insertion Sort Process Time for a Data Quantity of 80,000.

23. Refer to the previous question.

Estimate the Process Time for a Data Quantity of 120,000.

24. Look again at the 3 charts in **Figure 18.45**.

What do you notice about the *behaviors* of all 3 sorts?

25. Which sort (bubble, selection or insertion) would be the best to use if you already have a sorted list of data and you need to add a few new members to the list?

26. All 3 sorting algorithms shown thus far (bubble, selection or insertion) will \_\_\_\_\_\_\_\_\_\_\_\_\_ their *Processing Time* when the *Data Quantity* doubles.

27. Look at the 2 charts in **Figure 18.47**.

Is the Merge Sort a little faster than the Insertion Sort, or a lot faster?

28. Refer to the previous question.

Does the Merge Sort have the *Quadrupling* behavior shown by the other 3 sorts in this chapter?

29. Look at program ***Java1822.java*** and its chart in **Figure 18.49**.

Estimate the Process Time for a Data Quantity of 8,000,000.

30. Refer to the previous question.

Estimate the Process Time for a Data Quantity of 12,000,000.

31. Look at program ***Java1823.java*** and its 2 charts in **Figure 18.51**.

Estimate the Binary Seach Process Time for a Data Quantity of 8,000,000.

32. Refer to the previous question.

Estimate the Process Time for a Data Quantity of 16,000,000.

33. How many times will the **qwerty** method be called in this program?

**public class Ex1834**

**{**

**public static void main(String args[])**

**{**

**int t = 0;**

**for (int p = 0; p <= 4; p++)**

**for (int q = 1; q < 4; q++)**

**qwerty();**

**}**

**}**

34. How many times will the **qwerty** method be called in this program?

**public class Ex1835**

**{**

**public static void main(String args[])**

**{**

**int t = 0;**

**for (int p = 0; p < 10; p++)**

**for (int q = p; q < 10; q++)**

**qwerty();**

**}**

**}**