Question 1:

a. True, since the comparisons and the swaps are done regardless of the condition. This mean any arrays of the same size would have the same number of comparisons and swaps.

b. True. A reverse list: A[0] > A[1] > A[2] > A[4]… So it doesn’t matter which v we are dealing with, this condition will hold true for a major part of the array.

In a reverse list, every next element will have to be compared against all the elements before it, because it is always going to be placed in the first position.

c. True. -> We’ll get back to explaining this later

Question 2:

Bubble sort exhibits the worst behaviour when the swap always happens. In other words, the next element A[j + 1] is always less than the current element A[j]. This means we have a descending array.

Question 3:

[30, 50, 70, 10, 40, 60]

Insertion Sort

First iteration: key = 50 (i = 1)

[30, 50, 70, 10, 40, 60]

Second iteration: key = 70 (i = 2)

[30, 50, 70, 10, 40, 60]

Third iteration: key = 10 (i = 3)

[10, 30, 50, 70, 40, 60]

Fourth iteration: key = 40 (i = 4)

[10, 30, 40, 50, 70, 60]

Fifth iteration: key = 60 (i = 5)

[10, 30, 40, 50, 60, 70]

Selection Sort

First iteration: (i = 0)

Minimum is 10, swap first number with 10

[10, 50, 70, 30, 40, 60]

Second iteration: (i = 1)

Minimum is 30, swap second number with 30

[10, 30, 70, 50, 40, 60]

Third iteration: (i = 2)

Minimum is 40, swap third number with 40

[10, 30, 40, 50, 70, 60]

Fourth iteration: (i = 3)

Minimum is 50, swap fourth number with 50

[10, 30, 40, 50, 60, 70]

Fifth iteration: (i = 4):

Minimum is 60, swap fifth number with 60

[10, 30, 40, 50, 60, 70]

**Bubble Sort**

[10, 50, **70, 30**, 40, 60]

Swap 70 and 30:

[10, 50, 30, **70, 40**, 60]

Swap 70 and 40:

[10, 50, 30, 40, **70, 60**]

Swap 70 and 60:

[10, 50, 30, 40, 60, 70]

[10, **50, 30**, 40, 60, 70]

Swap 50 and 30

[10, 30, **50, 40**, 60, 70]

Swap 50 and 40

[10, 30, 40, 50, 60, 70]

Question 4:

Test the Insertion Sort algorithm:

Normal cases: Random array, repeated values

Boundary cases: Already sorted array, inversely sorted array

Exceptional cases: Empty array

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Description** | **Precondition** | **Postcondition** | **Input data** | **Expected output** | **Actual result** |
| Sort a random array | Given a random array with no duplicates | The array will be sorted in ascending order | [1, 6, 3, 2, 5, 8, 7, 4, 9, 0] | [0, 1, 2, 3, 4, 5, 6, 7, 8, 9] being printed out |  |
| Sort a random array with repeated values | Given a random array with repeated values | The array will be sorted in non-descending order. | [6, 4, 5, 3, 4, 1, 2, 7, 3, 2] | [1, 2, 2, 3, 3, 4, 4, 5, 6, 7] being printed out |  |
| Sort an already sorted array | Given an ascending array with no duplicates | The array will still be sorted in ascending order | [0, 1, 2, 3, 4, 5, 6, 7, 8, 9] | [0, 1, 2, 3, 4, 5, 6, 7, 8, 9] being printed out |  |
| Sort an inversely sorted array | Given a descending array with no duplicates | The array will be sorted in ascending order | [9, 8, 7, 6, 5, 4, 3, 2, 1, 0] | [0, 1, 2, 3, 4, 5, 6, 7, 8, 9being printed out |  |
| Sort an empty array | Given an empty array | The array will remain empty | [] | [] being printed out |  |