Basic sorting algorithms

A – Theory part

A.1. Identify the **best case** and **worst case** of *bubble sort*. Repeat the question for *selection sort*, *insertion sort*, and *interchange sort*.

A.2. Consider an array of 5 integers. What is the maximum number of **comparisons** if the array is sorted by *bubble sort*? Justify your answer. Repeat the question for *selection sort, insertion sort*, and *interchange sort*.

A.3. Consider an array of 5 integers. What is the maximum number of **data moves** if the array is sorted by *bubble sort*? Justify your answer. Repeat the question for *selection sort*, *insertion sort*, and *interchange sort*.

A.4. Consider the following array of integers, {26, 48, 12, 92, 28, 6, 33}. For *bubble sort*, show the resulting array for each of the first three iterations of the outer loop. Repeat the question for *selection sort*, *insertion sort*, and *interchange sort*.

A.5. *Selection sort compares each pair of keys at most once*. Is the given statement TRUE or FALSE? Justify your answer.

A.6. We could improve the efficiency of *bubble sort* by early termination using only an additional Boolean variable. Describe how to do that with your pseudo-code.

A.7. Consider a situation when all elements of the input array are identical. Which sorting algorithm will take least time?

A.8. Consider a situation where swapping is very costly. Which sorting algorithm(s) should be preferred so that the number of swaps are minimized in general? Justify your answer.

A.9. Assume that the following timings were taken of three algorithms as they processed lists of integers. Which of these algorithms is most likely to be *selection sort*? What characteristics of the sort (and its performance) led you to your choice?

	time to process 2000 integers	time to process 4000 integers	time to process 8000 integers
Algorithm 1	0.1431 sec	0.5722 sec	2.2989 sec
Algorithm 2	0.8011 sec	1.4300 sec	2.4512 sec
Algorithm 3	0.0132 sec	0.0304 sec	0.0634 sec

A.10. This C++ fragment implements *bubble sort* for **ascending order**. Is the code valid? If no, suggest how to fix the errors.

```
for (i = 1; i < n; i++)
    for (j = n – 1; j <= i; j--)
        if (a[j] > a[j - 1])
            a[j] = a[j - 1]);
            a[j-1] = a[j];
```

A.11. Which sorting algorithm is presented by this C++ code segment? Explain.

```
int s, t, numItems = 10;
for (s = (numItems - 1); s >= 0; s--){
    for (t = 1; t <= s; t++){
        if (arr[t - 1] > arr[t]) {
            int temp = arr[t - 1];
            arr[t - 1] = arr[t];
            arr[t] = temp;
        }
    }
}
```

A.12. Choose the best answer for each of the below single-choice questions about *selection sort*.

- *Through experiment, you determine that selection sort performs 5000 comparisons when sorting an array of some size k. If you doubled the size of the array to 2k, approximately how many comparisons would you expect it to perform?*

  a) 5000 b) 10000 c) 20000
  d) 40000 e) The value would depend on the array's content

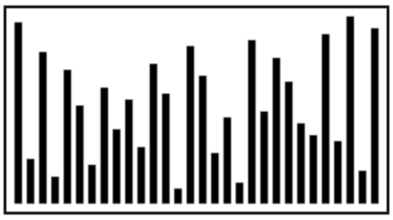
- *Through experiment, you determine that selection sort performs 5000 data moves when sorting an array of some size k. If you doubled the size of the array to 2k, approximately how many moves would you expect it to perform?*

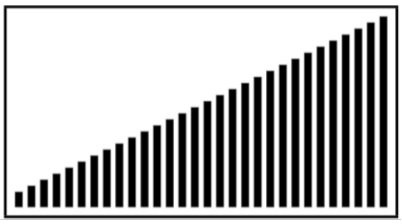
  a) 5000 b) 10000 c) 20000
  d) 40000 e) The value would depend on the array's content

- *A selection sort of 100 items has completed 42 iterations of the main loop. How many items are now guaranteed to be in their final spots (never to be moved again)?*

  a) 21 b) 41 c) 42 d) 43

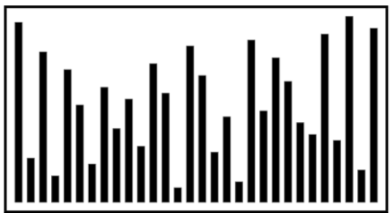
A.13. Suppose we want to sort a collection of sticks according to heights.

From:


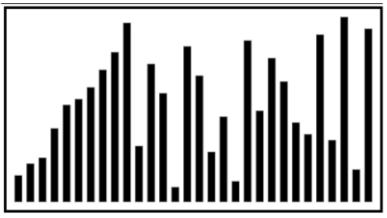
To:


Each of the following diagrams show the state of the sticks part way through sorting using a different sorting algorithm. For each diagram, say which algorithm is being used, and give a brief explanation for your choice.

From:



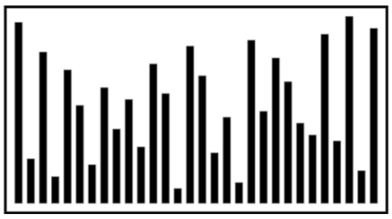
Part way:



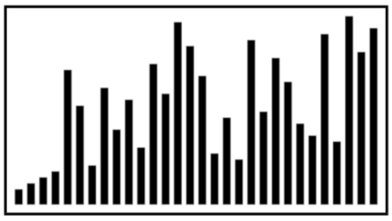
The algorithm above is ________________________ because ____

____

From:



Part way:



The algorithm above is ________________________ because ____

____

B – Coding part

B.1. Provide appropriate modifications to *selection sort* so that it can be used for **descending order**. Repeat the question for *selection sort, insertion sort*, and *interchange sort.*

B.2. Implement the improved version of *bubble sort*, in which a flag is used to check whether the array is sorted.

B.3. Implement the improved version of *insertion sort,* BinaryInsertionSort, which finds the insertion point by binary search.

B.4. Implement an O(n^2) sorting algorithm that is different from those in the lecture.

B.5. Print the elements of an array in the **decreasing frequency**. If two numbers have same frequency, print the one which came first. For example,

Input	Output
{2, 5, 2, 8, 5, 6, 8, 8}	{8, 8, 8, 2, 2, 5, 5, 6}
{2, 5, 2, 6, -1, 9999999, 5, 8, 8, 8}	{8, 8, 8, 2, 2, 5, 5, 6, -1, 9999999}

B.6. Count the number of inversions in an array of integers, indicating how far the array is from being sorted. If array is already sorted, the inversion count is 0. If array is sorted in reverse order, the inversion count is the maximum. Formally speaking, two elements `a[i]` and `a[j]` form an inversion if `a[i] > a[j] and i < j`.

For example,

Input	Output	Explanation
{8, 4, 2, 1}	6	(8,4), (4,2), (8,2), (8,1), (4,1), (2,1)
{3, 1, 2}	2	(3,1), (3,2)

B.7. Write a function to receive an array of strings and arrange the strings in **ascending order**. For example,

Input	[Farm, Zoo, Car, Apple, Bee, Golf, Bee, Dog, Golf, Zoo, Zoo, Bee, Bee, Apple]
Output	[Apple, Apple, Bee, Bee, Bee, Bee, Car, Dog, Farm, Golf, Golf, Zoo, Zoo, Zoo]

Then write another function that accepts the newly sorted array and outputs the set of distinct strings, each of which associates with its number of occurrences. For example,

Input	[Apple, Apple, Bee, Bee, Bee, Bee, Car, Dog, Farm, Golf, Golf, Zoo, Zoo, Zoo]
Output	Apple: 2, Bee:4, Car:1, Dog:1, Farm:1, Golf:2, Zoo:3