SEARCHING – BASIC SORT

A – Theory part

A.1. *Linear search* must examine, on average, half of the N elements of an array while *binary search* needs to only check $\log_2 N$. Thus, *binary search* is a great deal more efficient than *linear search*. Why, then, should we still study and use *linear search*?

A.2. Consider the following array of integers, {1, 4, 6, 7, 9, 10, 14}. For *linear search*, which elements will be examined to determine that 9 is in the array, and which elements will be examined to determine that 11 is not in the array? Repeat the question for *binary search*.

A.3. Consider an array of 5 integers. What are the maximum number of **comparisons** and the maximum number of **assignments** 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. Which sorting algorithm(s) is best for each of the following situations? Justify your answer.

a. When all elements of the input array are identical, which algorithm(s) will take least time?

b. When the swapping is very costly, which algorithm(s) should be preferred so that the number of swaps are minimized in general?

A.6. 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

B – Coding part

B.1. Write the function, `removeAll`, to take three parameters: an array of integers, the number of elements in the array, and an integer (say, `removeItem`). The function should find and delete all the occurrences of `removeItem` in the array. If the value does not exist or the array is empty, output an appropriate message. Consider both cases: the array is unsorted, and the array is already sorted. (Note that after deleting the element, the number of elements in the array is reduced.)

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

B.3. 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.4. 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