LAB 2: QUADRATIC SORTING ALGORITHMS

Write a program to sort 25 integers into ascending order using simple sort algorithms: Bubble Sort, Insertion Sort and Selection Sort. For each sorting techniques learned in class, modify the algorithms so that the programs are able to count and print the number of passes, the number of data comparisons and the number of data swapping that take place in the sorting process.

a. Run the programs on the following list of integer values. You can initialize the data in the array declarations.

```
int dataArrayA[25]={.......................};
int dataArrayB[25]={.......................};
```

100	50	88	30	60	45	25	12	10	5	98	15	65	55	45	70	20	90	66	22	120	48	35	85	2

dataArrayA

5	8	30	25	35	40	42	50	55	22	24	60	66	70	75	78	80	88	95	100	118	98	120	122	121

dataArrayB

b. **dataArrayA** is an example of a worst case data – totally unsorted list, while **dataArrayB** is an example of a best case data – almost sorted list. Analyze the output displayed from each sorting technique based on the number of passes, number of data comparison and number of data swapping. Make your conclusion on which sorting technique is the best for worst case data and which sorting technique is the best for best case data. Which technique has performance that does not depend on the initial arrangement of data?

Answer in next page

c. Fill in the following table to help you discuss the performance analysis.

Technique	Case	No of Comparisons	No of Swaps	No of Passes
Conventional Bubble Sort	Worse Case	300	155	24
	Best Case	300	18	24
Discuss the performance Analysis for Conventional Bubble Sort				
Improved Bubble Sort	Worse Case	300	155	24
	Best Case	164	18	8
Discuss the performance Analysis for Bubble Sort				
Selection Sort	Worse Case	300	24	24
	Best Case	300	24	24
Discuss the performance Analysis for Selection Sort				
Insertion Sort	Worse Case	155	155	24
	Best Case	18	18	24
Discuss the performance Analysis for Insertion Sort				
Discussion on the Overall Performance Analysis for all Quadratic Techniques				
Merge Sort	*Worse Case*	*118*	*none*	*88*
	Best Case	*118*	*none*	*65*
Quick Sort	*Worse Case*	*122*	*89*	*122*
	Best Case	*212*	*208*	*212*

Bubble Sort O(n2):
- Number of comparisons will stay the same in all cases O(n2)
- Number of swaps might be 0 in best case or O(n2) in worst case

Improved Bubble Sort O(n) …. O(n2):
- Number of comparisons will be reduced to O(n) in best case and continue to be O(n2) in worst case
- Number of swaps might be 0 in best case or O(n2) in worst case

Selection Sort O(n2):
- Number of comparisons will stay the same in all cases O(n2)
- The efficiency of Selection Sort does not depend on the initial arrangement of the data
- Number of swaps might be O(1) in best case or O(n) in worst case

Insertion Sort O(n) …. O(n2):
- Number of comparisons will be reduced to O(n) in best case and continue to be O(n2) in worst case
- Number of shifts or swaps might be O(1) in best case or O(n2) in worst case

Merge Sort O(n lg n):
- Will have the same number of comparisons in all cases O(n lg n).
- It is extremely fast algorithm, but on the other hand it also require more space to hold the original array in a temporary new array.
- It doesn’t need to swaps element it will return the array after dividing it and combine it again in a sorted order.

Quick Sort O(n lg n) …. O(n2):
- This algorithm depends on the balance of the array, so that if the array is balanced then it will become O(n lg n), but if it is not then you will go to the worse case O(n2)
- In practice quick sort is counted as fast algorithms even if it happened to be in the worst case it will still run on an acceptable speed.

Analysis:
Bubble Sort and Selection Sort are not very efficient algorithms to do their time complexity in all situations however, improved Bubble Sort could be really useful in many cases and very efficient than most other algorithms due to their very great time complexity O(n). Insertion sort can be really helpful if it runs in it is best case, but it is not great choice in case of worst case which leaves us with the last two options.

Merge Sort vs Quick Sort:
In the worst case, merge sort does about 39% fewer comparisons than quicksort does in the average case + Merge sort is more efficient than quicksort for some types of lists if the data to be sorted can only be efficiently accessed sequentially, however Quick Sort could perform better in the case of best case because of the time space complexity which Merge Sort lack and require a temporary copy of the original array to perform it is operation.