Project Report

DAA_Project.py: This file contains all the sorting algorithms along with their executions

Imports used - random to generate random numbers and time to calculate the execution time in seconds

Functions - This file has 14 functions which includes the main function

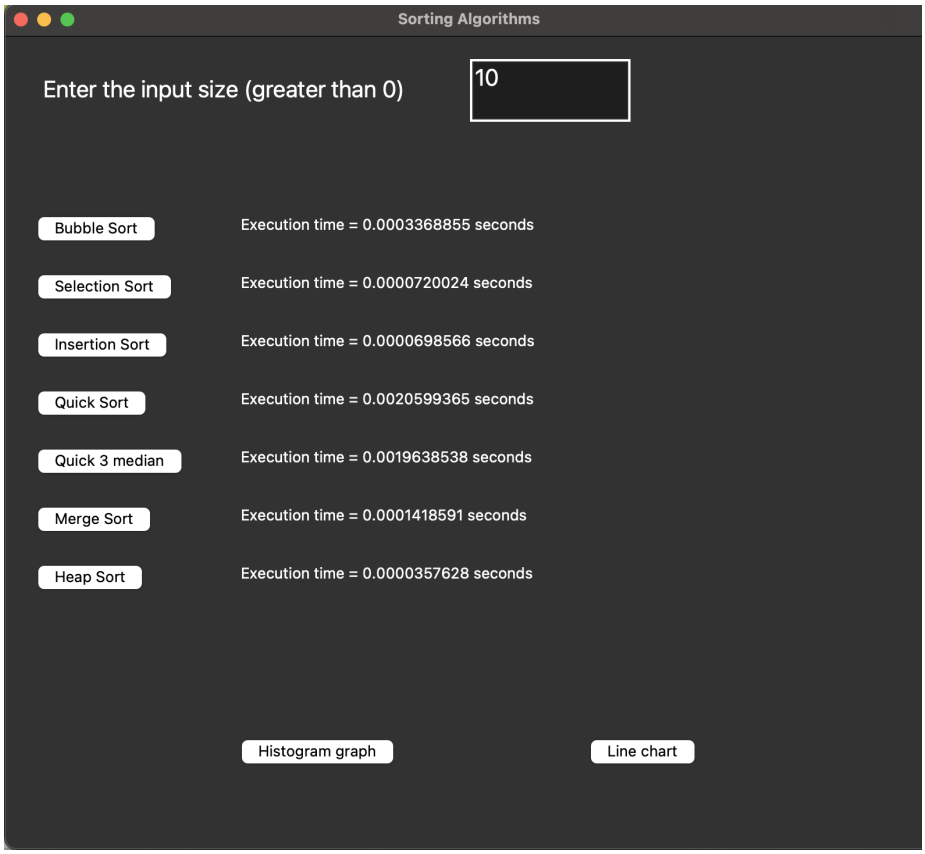
Function name	Description
main	This function reads the input from the user for input size, generates a random list based on the input size and executes all the sorting algorithms and calculates their execution times
bubble_sort	This function fetches random numbers list and creates a new list, performs sorting on the list using bubble sort algorithm and returns a sorted list
selection_sort	This function fetches random numbers list and creates a new list, performs sorting on the list using selection sort algorithm and returns a sorted list
insertion_sort	This function fetches random numbers list and creates a new list, performs sorting on the list using insertion sort algorithm and returns a sorted list
merge	This function is merging the array list in sorted manner
merge_sort	This function is dividing the array list recursively into two halves till the size becomes 1
quickSort1	This function is the recursive sorting function based on quick search which is used for both regular and 3 Median quick sort
partition	This function partitions the array list based on last element as pivot
quick_sort	This function is to call the regular quick sort recursive function and return the sorted list
quick_sort_3Median	This function is to call the 3 Median quick sort recursive function and return the sorted list
qs3MedianPartition	This function finds the median based on 3 Median process, creates partitions and returns the index of the pivot. The indexof pivot is used in function quickSort1 to call recursive quick sort.
heap_sort	This function makes used of buildHeap() and heapify() for sorting the array list using heap sort
buildHeap	This function is used to build a max heap from the input list of numbers
heapify	This function rearranges the heap in case the heap violates the property to build a max heap

DAA_Project_GUI.py: This file contains all the UI related functions and methods along with the calling of DAA_Project.py file to use the sorting function in this file to perform the execution and plotting the graphs

- The GUI has been created using the Tkinter python library. On execution a new window appears which has the textbox, labels and clickable buttons
- The input size takes value greater than zero which takes the dataset size or the list size to sort
- Different clickable buttons for every sort are provided which will show the execution time in seconds on clicking
- Histogram graph and line chart buttons showing the comparison between sorting algorithms and their execution times

Experimental results

Data size = 10



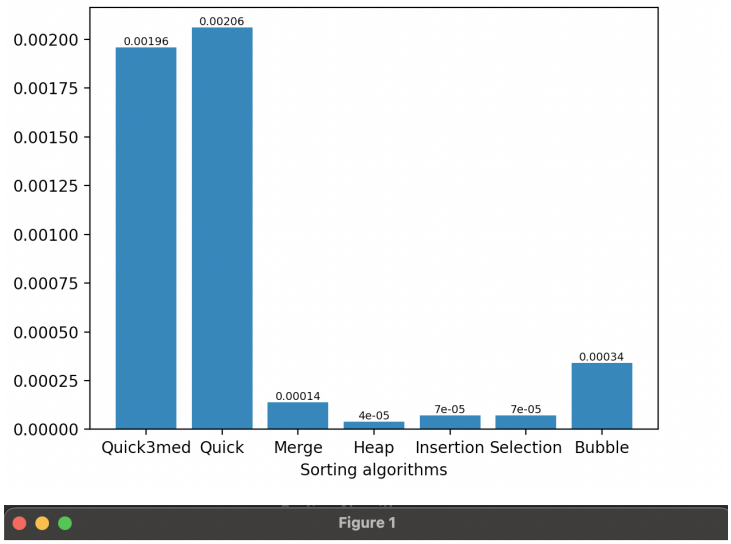
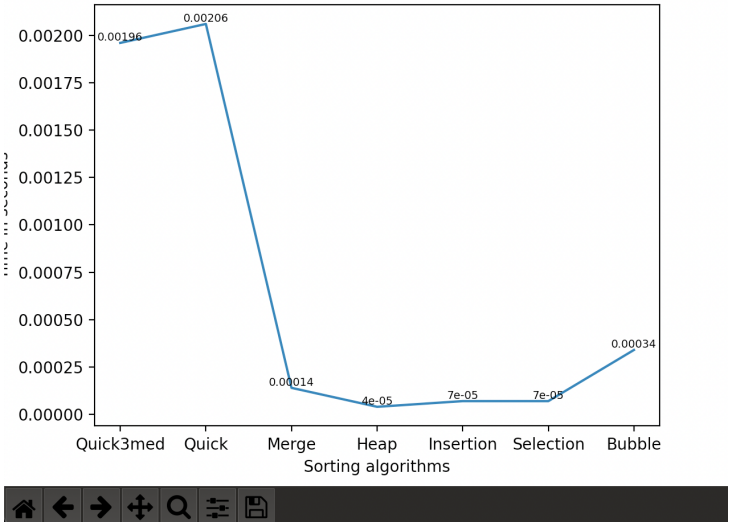


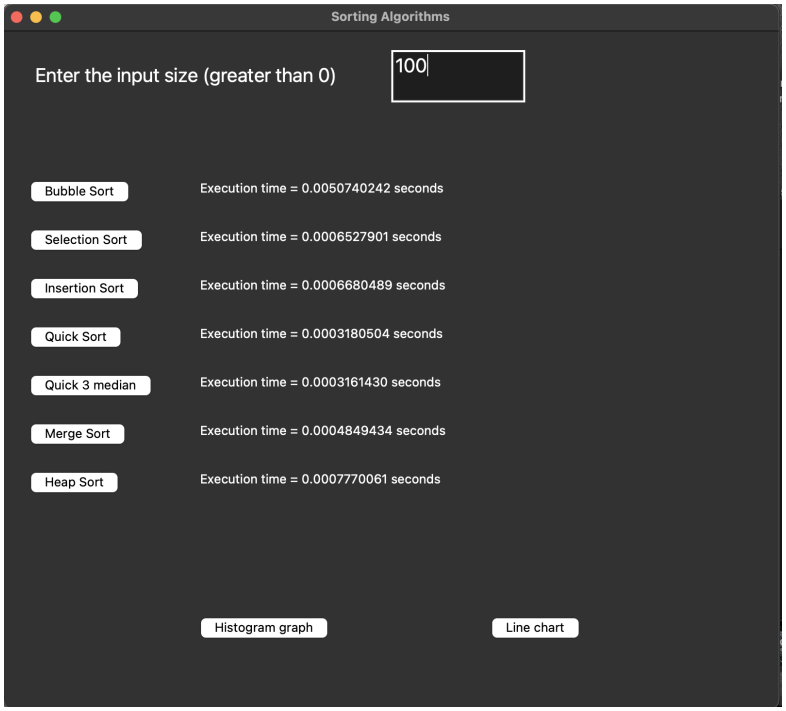
Figure 1

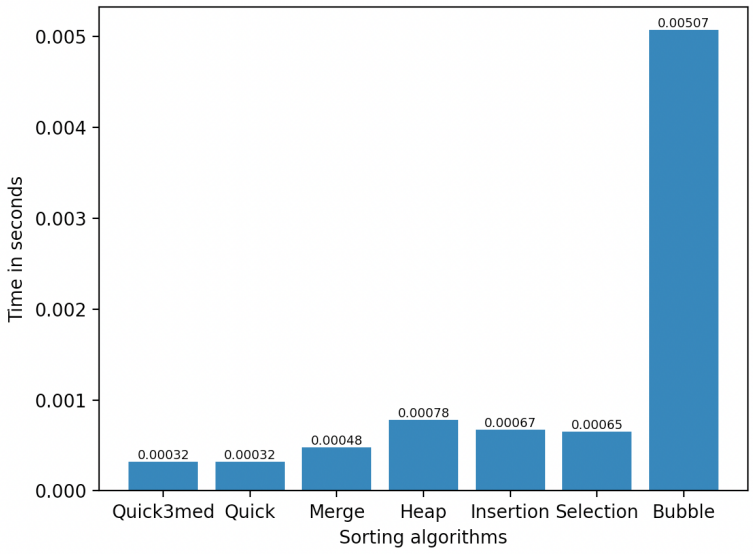



```
Bubble Sort: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9] , Running time: 0.0003368855 seconds

Selection Sort: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9] , Running time: 0.0000720024 seconds

Insertion Sort: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9] , Running time: 0.0000698566 seconds

Quick Sort: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9] , Running time: 0.0020599365 seconds

Quick Sort 3 median: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9] , Running time: 0.0019638538 seconds

Merge Sort: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9] , Running time: 0.0001418591 seconds

Heap Sort: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9] , Running time: 0.0000357628 seconds
```

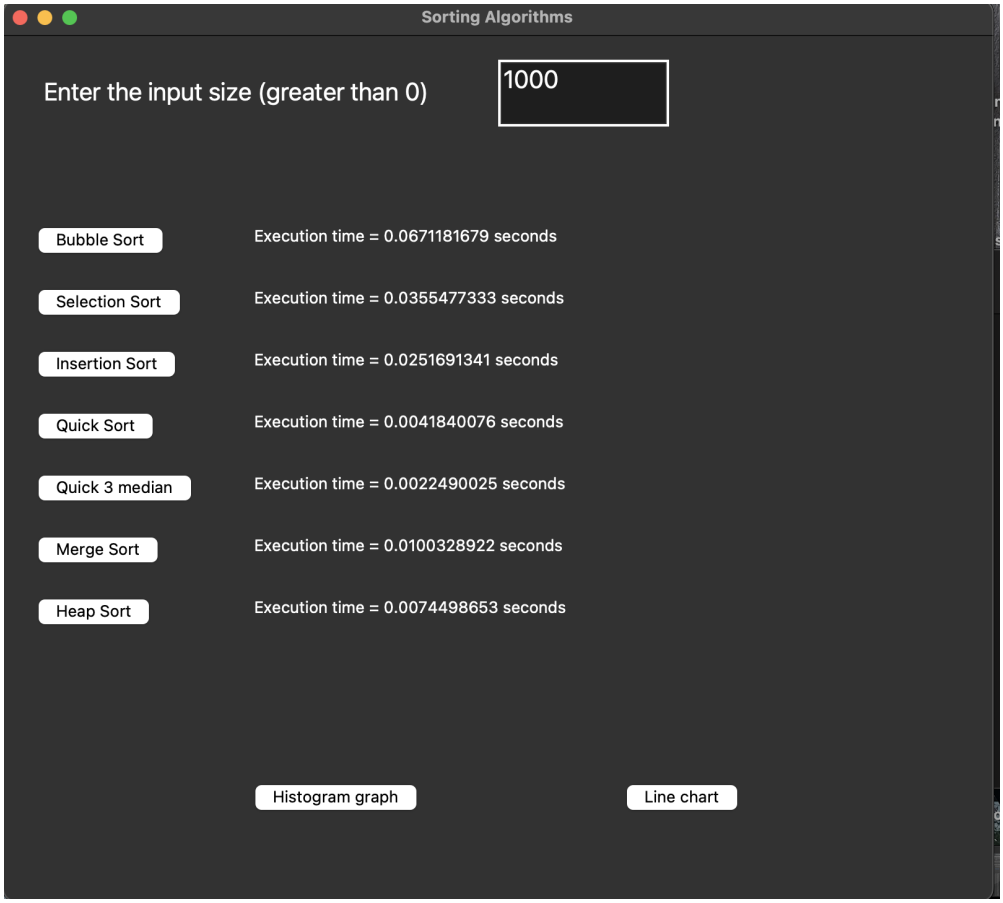
Version Control | Run | TODO | Problems | Debug | Python Packages | Python Console | Terminal | Event Log

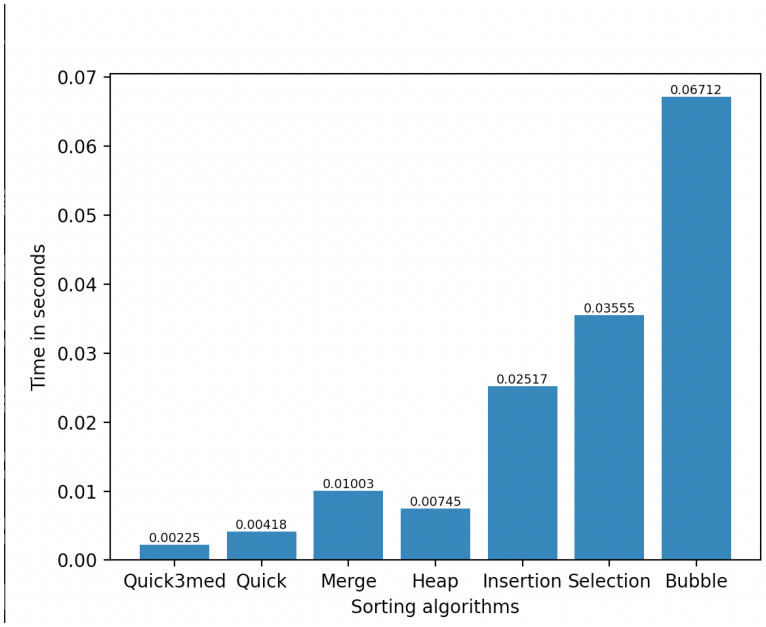
Data size = 100



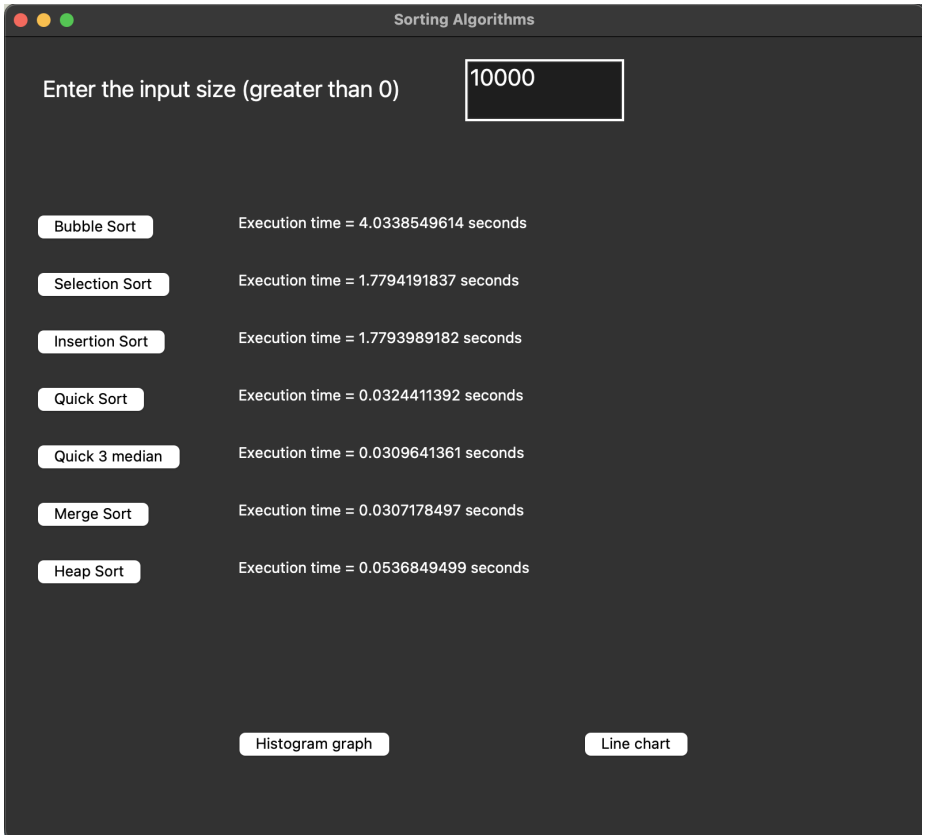


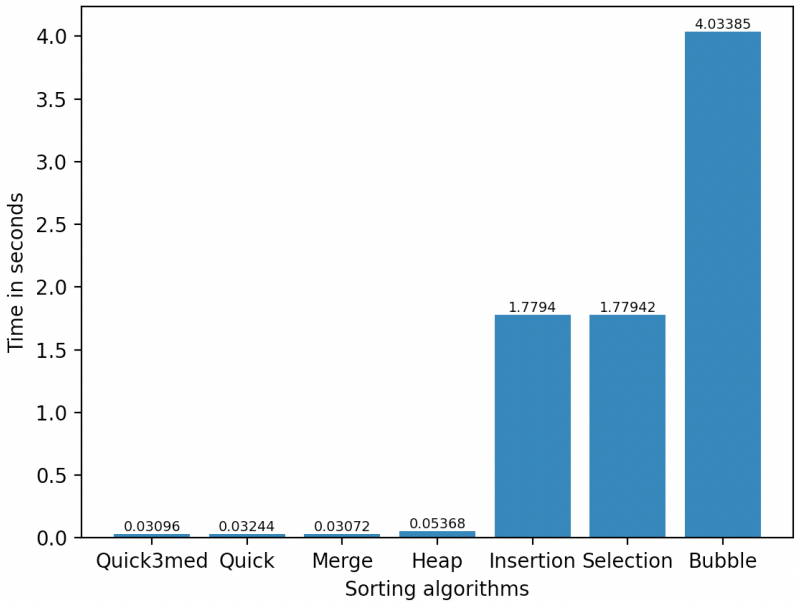
Data size = 1000



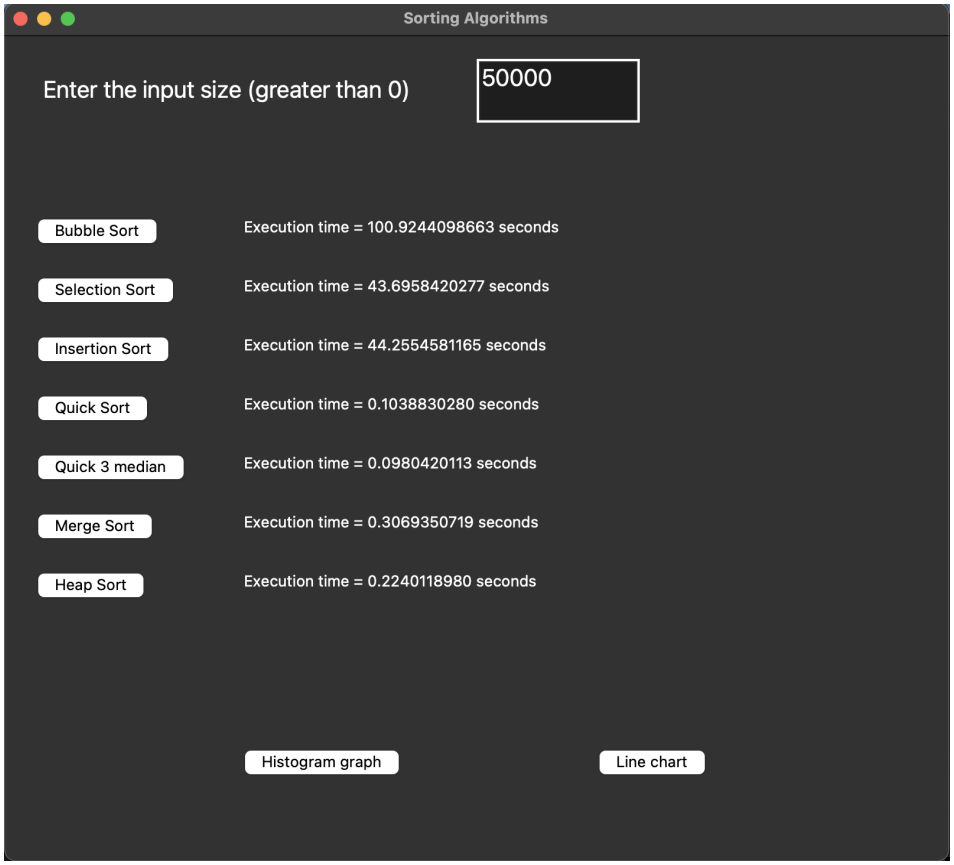


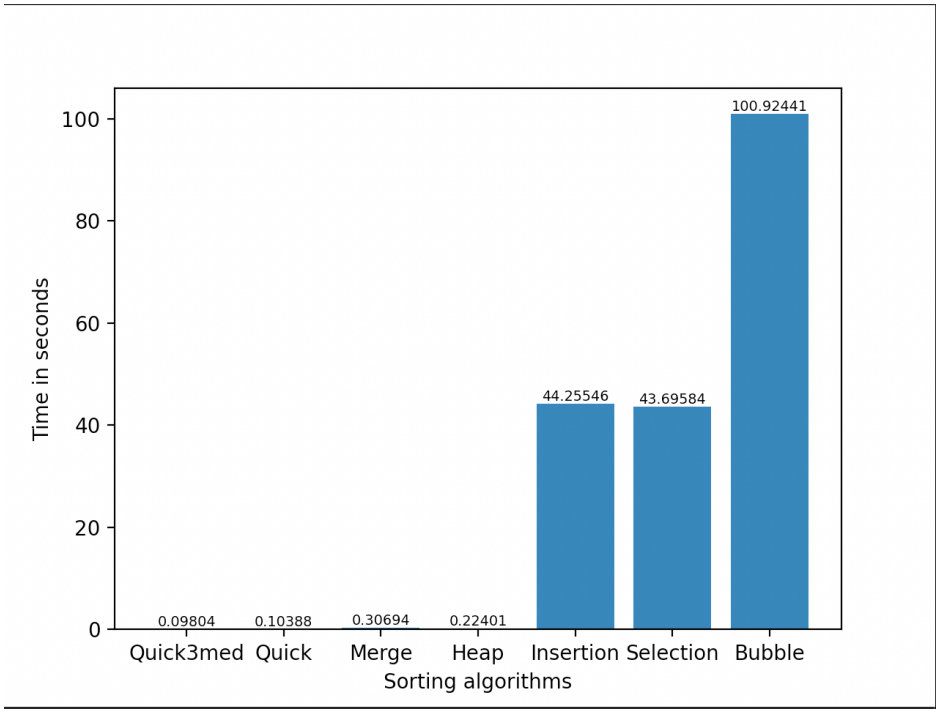
Data size = 10000





Data size = 50000





Improvements

Bubble sort: This algorithm ensures that the elements at the end are sorted one by one from the right so we don't need to traverse the inner loop until the end in all iterations

Selection sort: We can check whether the current element and the smallest element are same before swapping the array's elements

Insertion sort: This algorithm can be improved by using binary search rather than linear search to determine where the element should be inserted

Heap sort: This algorithm can be improved by using it as in-place sorting where the same array is used as heap

Quick sort: The improvement can be seen with 3 median process where we find the median as the pivot element

Merge sort: We can cut the running time by skipping the call to merge function if the array is already in order

Better - Conditions

Bubble sort: This algorithm is fast and good for small data sizes. This algorithm performs better when the dataset is almost sorted

Selection sort: This algorithm performs better when the array size is relatively small and the dataset is not partially sorted

Insertion sort: This algorithm performs better when the dataset is small and the input array is nearly sorted with some elements not in position in the entire dataset

Heap sort: This algorithm performs better when there is a need to manage the memory well as this algorithm needs minimal memory usage as it is an in-place algorithm

Quick sort: This algorithm performs better with the large size of data

Merge sort: This algorithm performs better with the large size of data

References: Pseudocode from the course slides has been referred to build the code for sorting algorithms