Lecturer: Dominic Carr

Computational Thinking With Algorithms

Student: Richard Deegan G00387896

Contents

[Preliminaries 2](#_Toc72009227)

[Introduction 2](#_Toc72009228)

[Time and Space Complexity 3](#_Toc72009229)

[Big O Notation 3](#_Toc72009230)

[Performance, Comparator functions and Time Complexity. 4](#_Toc72009231)

[Stable and Unstable algorithms 6](#_Toc72009232)

[Ideal Sorting Algorithm. 7](#_Toc72009233)

[In place sorting 7](#_Toc72009234)

[Comparison and non-comparison based. 7](#_Toc72009235)

[Sorting Algorithms 7](#_Toc72009236)

[Bubble Sort – Comparison Based Algorithm 7](#_Toc72009237)

[Introduction 7](#_Toc72009238)

[Space and time complexity 9](#_Toc72009239)

[How Bubble Sort works with original diagram 9](#_Toc72009240)

[Merge Sort 11](#_Toc72009241)

[Introduction 11](#_Toc72009242)

[Space and Time Complexity 12](#_Toc72009243)

[How Merge Sort Works With Original Diagram 12](#_Toc72009244)

[Counting Sort – non Comparison. 15](#_Toc72009245)

[Introduction 15](#_Toc72009246)

[Space and Time Complexity 15](#_Toc72009247)

[How Counting Sort Works With Original Diagram 15](#_Toc72009248)

[Selection Sort 17](#_Toc72009249)

[Introduction 17](#_Toc72009250)

[Space and Time Complexity 17](#_Toc72009251)

[How Selection Sort Works With Original Diagram 18](#_Toc72009252)

[Insertion Sort 19](#_Toc72009253)

[Introduction 19](#_Toc72009254)

[Space and Time Complexity 22](#_Toc72009255)

[How Insertion Sort Works With Original Diagram 22](#_Toc72009256)

[Implementation and Benchmarking 24](#_Toc72009257)

[Results of the benchmarking 24](#_Toc72009258)

[Summary of Results- Graph and Table 24](#_Toc72009259)

[References 24](#_Toc72009260)

# Preliminaries

For this project you will write an application which will be used to benchmark five different sorting algorithms. You will also write a report which introduces the algorithms you have chosen and discusses the results of the benchmarking process. The five sorting algorithms must be chosen according to the following criteria:

1. A simple comparison-based sort (Bubble Sort, Selection Sort or Insertion Sort)

2. An efficient comparison-based sort (Merge Sort, Quicksort or Heap Sort)

3. A non-comparison sort (Counting Sort, Bucket Sort or Radix Sort)

4. Any other sorting algorithm of your choice

5. Any other sorting algorithm of your choice.

For the purpose of this assignment brief the following Algorithms were chosen:

* Bubble Sort
* Merge Sort
* Counting Sort
* Selection Sort
* Insertion Sort

# Introduction

Algorithms are a set of processes or rules that must be followed in order to solve problems. A common analogy for a computer algorithm is that of a food recipe. They are similar in that they are both a set of instructions that must be followed to get a desired outcome. Many different algorithms can be created to achieve the same result, just like recipes, in this instance the flavour and taste may be different where in the case of algorithms, maybe the run time will be different. In general, there are seven key features that form the basis of well-designed algorithms:

1. Input: Input data is well defined when beginning the algorithms operations
2. Output: Output data is well defined when the algorithm has finished.
3. Finiteness: An algorithm must always end unless it will run forever.
4. Unambiguous: Each step/set of code must be well defined and logical in each case.
5. Correctness: The algorithm must arrive at a correct solution, within error rates.
6. Feasibility: It should be feasible to run the algorithm considering resources.
7. Efficiency: The algorithm must be as quick as possible given the size of the task.

# Time and Space Complexity

Considering the features above, two key concepts emerge when considering algorithms; that is space and time complexity. Space complexity refers to the amount of space or memory it takes an algorithm to run its function. It includes both auxiliary space and input space. Space complexity S(p) of any algorithm p is S(p) = A + Sp(I) Where A is treated as the fixed part and S(I) is treated as the variable part of the algorithm which depends on instance characteristic I. The example below illustrates the concept.

A picture containing graphical user interface

Description automatically generated

Here we have three variables R,P, Q and one constant. S(p) is equal to 1 + 3. Now space is dependent on the data types of the given constant types and variables, this will be multiplied accordingly. [1].

Time complexity is the amount of time it takes an algorithm to complete its task as a function of the input length. The time complexity of algorithms is usually expressed using notation such as Big O, Big Theta, or Big Omega. Essentially we are considering how well the algorithm deals with the scaling of numbers, as the input data size increases its important to know how well an algorithm will scale accordingly. For a programmer, its highly valuable to know the effect of space and time complexity on algorithms, this will enable the programmer to solve problems in the most efficient way possible.

# Big O Notation

Describing the run time of an algorithm is a complex task and all efforts are focused on finding a function that best approximates the runtime. Asymptotic notations are used to describe the run time and have roots from Number Theory []. Big O notation is used to describe asymptotic behaviours of functions. The growth rate of the functions is called “order”, and it is order of magnitudes in which concerns the programmer.

**Big-oh (O):** Let f(n) and g(n) be functions such that: - ∃k > 0 ∃n0 ∀n > n0 |f(n)| ≤ |g(n) · k| then f(n) ∈ O(g(n)) or with some abuse of notation f(n) = O(g(n)). [Krueger et al., 1999] essentially Big-Oh represents the upper limit of resources required for the algorithm to run, that is the maximum amount of time and space. It describes the algorithm in the worst-case scenario, in essence it describes how a function grows slower or at least as slow as another.

**Big Omega(Ω):** Omega is used to describe the complexity of an algorithm in the best case. Essentially it represents the lower limit of resources required for the algorithm to run. This is the best-case scenario and exhibits linear growth in execution time as n is increased. []

**Big-Theta (Θ):** Let f(n) and g(n) be functions such that :- ∃k1 > 0 ∃k2 > 0 ∃n0 ∀n > n0 g(n) · k1 ≤ f(n) ≤ g(n) · k2 then f(n) ∈ Θ(g(n)) or with abuse of notation f(n) = Θ(g(n)).[] Essentially the theta notation is utilised to specify the runtime of an algorithm is no greater or less than a certain “order”. It describes the complexities of the algorithm in the average case by representing both big-oh (O) and Big Omega(Ω) expressions.

The following is a list of common types of orders and their names (Danziger, 2010):

Graphical user interface, text, application

Description automatically generated

Common Sorting algorithms include the following, Selection Sort, Bubble Sort, Insertion Sort, Merge Sort, Quick Sort, Heap Sort, Counting Sort, Radix Sort, and Bucket Sort. Sorting algorithms.

# Performance, Comparator functions and Time Complexity.

There are two main key features when determining the performance an algorithm (Mejia, 2020).

1. Input Size: The size of the data (n) which is inputted into the algorithm must be considered, or the number of data items in a single instance.
2. Operation count: The number of operations f(n) required by the algorithm, or its running time.

Diagram, schematic

Description automatically generated

Considering the above diagram, if we use statements with basic operations such as comparisons, assignments, or reading a variable, we can therefore assume that they take a constant time. That is 0(1). If there is a fixed number of operations we can say that time will be constant, even if we have multiple statements.

Conditional statements are critical for coding algorithms and calculating the time complexity considering the worst case or big -oh, we must take the worst-case performance. For instance, an “if” block of code takes 0(n log n) time complexity, while the associated “else” part of the code takes a runtime of O(1), in this case 0(n log n) is used as it is the worst case.

Loop statements are determined by figuring out the runtime of the block inside the loop and multiplying it by the number of times the program repeats the loop. All loops that grow in proportional to the input size will have a linear time complexity, 0(n). If the array is looped only half the way then its still 0(n). Constant time loops are different in that the runtime is constant, that is if the code is not dependent on the input size it will always run x times and therefore possess a time complexity of 0(n).

For logarithmic time loops where an array is split in half for each iteration. The while loop will execute for the number of times we can split the array length in half. This is calculated by suing the log function. For example, of the array length is 8, then we can say log2(8) = 3.

Nested loops are highly inefficient as the algorithm must loop over each element within the nested loop as well as the original loop, the expected time complexity would be O(n^2).

The time complexity of functions depends on the runtime of the function. If the run time of say 3 functions are all constant at O(1) then the final runtime would be O(n^3).

For recursive functions, it’s possible to visualise each function invocation as a node. The below figure helps illustrates this in greater detail. The total number of calls in a complete node tree is 2^n -1. Fn(4) is not complete and the last level has only two nodes, that is Fn(1) and Fn(0), while a complete node tree would have eight nodes. For this instance, the run time is exponential O(2^n), however it won’t get any worse as the upper limit is 2^n.

Diagram

Description automatically generated

# Stable and Unstable Algorithms

A sorting algorithm is said to be stable if two objects with equal keys appear in the same order in sorted output as they appear in the input array to be sorted. Some sorting algorithms are stable by nature like Insertion sort, Merge Sort, Bubble Sort, etc. On the other hand, in an unstable algorithm, straw or spork may be interchanged e.g selection sort, quick sort, heap sort. [] In essence, sorting stability ensures records with the same key keep the relative order, before and after the sort. This may be important depending on the problem you are trying to solve. However, if the retention of that relative order is not necessary then an unstable algorithm may be used, in this case it will free up some memory space.

Diagram, table

Description automatically generated

In place sorting: this type of sorting algorithm uses a fixed additional amount of working space that is independent of the input size. Insertion sort and bubble sort would be examples of this. In other words, these algorithm uses constant extra space for producing the output. It sorts the list only by modifying the order of the elements within the list [14].

## Ideal Sorting Algorithm.

The ideal sorting algorithm would have the following (news.ycombiator, 2016):

* Stable: Equal keys aren’t reordered.
* Operates in place, requiring O(1) extra space.
* Worst-case O(n·lg(n)) key comparisons.
* Worst-case O(n) swaps.
* Adaptive: Speeds up to O(n) when data is nearly sorted or when there are few unique keys.

# Sorting Algorithms

## Bubble Sort – Comparison Based Algorithm

### Introduction

Thebubble sort makes multiple passes through a list. It compares adjacent items and exchanges those that are out of order. Each pass through the list places the next largest value in its proper place. In essence, each item “bubbles” up to the location where it belongs [7].

A picture containing text, electronics, keyboard

Description automatically generated

The above figure illustrates this in practice. The shaded numbers are compared; hence bubble sort is a comparison based algorithm.  If there are n items in the list, then there are n−1n−1 pairs of items that need to be compared on the first pass. At the start of the second pass, the largest number is now in the correct place. There are now n-1 items in the array to sort out, or in other words, there will be at least n – 2 pairs. The exchange operation is referred to as “swap”, in Python is possible to perform a simultaneous assignment with a statement such as “a, b=b, a” will result in two assignments being done at the same time. This can be seen in the below image.

Diagram

Description automatically generated

### Space and time complexity

A bubble sort is often considered the most inefficient sorting method since it must exchange items before the final location is known. These “wasted” exchange operations are very costly. However, because the bubble sort makes passes through the entire unsorted portion of the list, it has the capability to do something most sorting algorithms cannot. In particular, if during a pass there are no exchanges, then we know that the list must be sorted. A bubble sort can be modified to stop early if it finds that the list has become sorted. This means that for lists that require just a few passes, a bubble sort may have an advantage in that it will recognize the sorted list and stop [8].

The time complexity of Bubble Sort is O(n2).

The main advantage of Bubble Sort is the simplicity of the algorithm.

The space complexity for Bubble Sort is O(1), because only a single additional memory space is required i.e. for temp variable.

Also, the best-case time complexity will be O(n), it is when the list is already sorted.

Following are the Time and Space complexity for the Bubble Sort algorithm.

* Worst Case Time Complexity [ Big-O ]: O(n2)
* Best Case Time Complexity [Big-omega]: O(n)
* Average Time Complexity [Big-theta]: O(n2)
* Space Complexity: O(1)

### How Bubble Sort works with original diagram

The Python code used to implement Bubble Sort is shown in below. Essentially two loops exist, the outer loop runs in increments of one from the second last array item, while the inner loop runs from the first array item in the list to the items in the outer list being ran. The output of the program is the following:

Text

Description automatically generated

Graphical user interface, application

Description automatically generated

## Merge Sort

### Introduction

Merge sort is a sorting technique based on divide and conquer technique. With worst-case time complexity being Ο(n log n), it is one of the most respected algorithms [9]. It is one of the most efficient sorting algorithms. It works on the principle of Divide and Conquer. Merge sort repeatedly breaks down a list into several sublists until each sublist consists of a single element and merging those sublists in a manner that results into a sorted list. [10]. In figure XXX we can see the array being dividing down to its core singular elements, the “divide”. While in Figure X we see the “merge” happening into a sorted list.

Diagram

Description automatically generated

Diagram

Description automatically generated

### Space and Time Complexity

The total time for mergeSort function will become n(log n + 1), which gives us a time complexity of O(n\*log n).

* Worst Case Time Complexity [ Big-O ]: O(n\*log n)
* Best Case Time Complexity [Big-omega]: O(n\*log n)
* Average Time Complexity [Big-theta]: O(n\*log n)
* Space Complexity: O(n)

Time complexity of mergeSort is O(n\*Log n) in all the 3 cases (worst, average and best) as merge sort always divides the array in two halves and takes linear time to merge two halves. Therefore, it requires an equal amount of additional space as the unsorted array. Hence it’s not at all recommended for searching large and unsorted arrays [8].

### How Merge Sort Works With Original Diagram

The python code for MergeSort is detailed below along with its output. In essence, the array is split or divided down to its single elements and then merged based on order of ascending or descending.

Text

Description automatically generated

Calendar

Description automatically generated with low confidenceA picture containing text

Description automatically generated

A picture containing text

Description automatically generated

## Counting Sort – non Comparison.

### Introduction

Counting Sort is an Integer-based algorithm, essentially the numbers or values of the input received must be assumed to be integers. Count sort is one of the faster algorithms. The distinct feature of counting sort is in the way it creates a bucket for each element and then retains a counter in each bucket. For every instance, a value is encountered within the input array, an appropriate counter is applied and incremented. A maximum value is imposed beforehand as Counting sort creates a bucket for each element, therefore the maximum value in the input array is known. Counting sort is used when there are smaller integers with multiple or large counts in a particular array [13].

### Space and Time Complexity

Counting sort is an algorithm for sorting a collection of objects according to keys that are small integers; that is, it is an integer sorting algorithm.

[**Worst complexity**](https://www.google.com/search?sxsrf=ALeKk03j2u_ddo9TZvrkzgQkyVGb5G3z1g:1621107926338&q=counting+sort+worst+complexity&stick=H4sIAAAAAAAAAOPgE-LQz9U3KEjPKdDSyU620s8uiM8p1y_OLyrJzEuPT8xJzy_KLMnItSrPLyouiU_Ozy3ISa3ILKlcxCqXnF-aB1KlAFKtAFaggFAAABIAzWtbAAAA&sa=X&ved=2ahUKEwi-zN3QuczwAhViRxUIHbGLBy0Q6BMoADAgegQIKhAC)**:**n+r

[**Average complexity**](https://www.google.com/search?sxsrf=ALeKk03j2u_ddo9TZvrkzgQkyVGb5G3z1g:1621107926338&q=counting+sort+average+complexity&stick=H4sIAAAAAAAAAOPgE-LQz9U3KEjPKdDSy0620s8uiM8p1y_OLyrJzEuPT8xJzy_KLMnItUosSy1KTE-NT87PLchJrcgsqVzEqpCcX5oHUqcAUq8AVaKAUAIA0dWMg18AAAA&sa=X&ved=2ahUKEwi-zN3QuczwAhViRxUIHbGLBy0Q6BMoADAhegQIJxAC)**:**n+r

[**Space complexity**](https://www.google.com/search?sxsrf=ALeKk03j2u_ddo9TZvrkzgQkyVGb5G3z1g:1621107926338&q=counting+sort+space+complexity&stick=H4sIAAAAAAAAAOPgE-LQz9U3KEjPKdBSyk620s8uiM8p1y_OLyrJzEuPT8xJzy_KLMnItcpNzc0vqlzEKpecX5oHklMAqVEoLkhMTlVIzs8tyEmtyCypBACMdQCmUQAAAA&sa=X&ved=2ahUKEwi-zN3QuczwAhViRxUIHbGLBy0Q6BMoADAiegQIKRAC)**:**n+r

[**Stable**](https://www.google.com/search?sxsrf=ALeKk03j2u_ddo9TZvrkzgQkyVGb5G3z1g:1621107926338&q=counting+sort+stable&stick=H4sIAAAAAAAAAOPgE-LQz9U3KEjPKdBSyk620s8uiM8p1y_OLyrJzEuPT8xJzy_KLMnItSouSUzKSV3EKpKcX5oHklMAqVGACAMA3UsbgUcAAAA&sa=X&ved=2ahUKEwi-zN3QuczwAhViRxUIHbGLBy0Q6BMoADAjegQIJhAC)**:**Yes

[**Class**](https://www.google.com/search?sxsrf=ALeKk03j2u_ddo9TZvrkzgQkyVGb5G3z1g:1621107926338&q=counting+sort+class&stick=H4sIAAAAAAAAAOPgE-LQz9U3KEjPKdBSzE620s8uiM8p1y_OLyrJzEuPT8xJzy_KLMnItUrOSSwuXsQqnJxfmgeSUgApUQCLAgDXX-s6RQAAAA&sa=X&ved=2ahUKEwi-zN3QuczwAhViRxUIHbGLBy0Q6BMoADAkegQIJRAC)**:**[Non-comparison sort](https://www.google.com/search?sxsrf=ALeKk03j2u_ddo9TZvrkzgQkyVGb5G3z1g:1621107926338&q=counting+sort+non-comparison+sort&stick=H4sIAAAAAAAAAOPgE-LQz9U3KEjPKVDi1U_XNzRMMzcyqagoLNNSzE620s8uiM8p1y_OLyrJzEuPT8xJzy_KLMnItUrOSSwuXsSqmJxfmgeSUgApUcjLz9NNzs8tSCzKLM7PA4vtYGUEAN_airxlAAAA&sa=X&ved=2ahUKEwi-zN3QuczwAhViRxUIHbGLBy0QmxMoATAkegQIJRAD)

### How Counting Sort Works With Original Diagram

Counting sort takes O(n+k)0(n+k) time and O(n+k)0(n+k) space. Counting sort iterates though the input items twice, that is once to populate counts and the other to fill in the output array. Both iterations are O(n)O(n)time. Similarly, we iterate though counts once to fill the next index, which of course is also O(k)O(k) time.

Text

Description automatically generated

Timeline

Description automatically generated

Chart

Description automatically generated

## Selection Sort

### Introduction

Selection sort is a simple comparison-based sorting algorithm. One benefit of this algorithm is that is requires no extra memory. The concept behind this algorithm is pretty simple. The array is divided into two parts, a sorted and an unsorted part. The left side is the sorted subarray while the right side is the unsorted subarray. At first, the sorted subarray is empty, and the unsorted array is the full array. The following steps are conducted unit the unsorted array becomes empty (consider the below diagram) [12]:

1. Select the lowest element from the unsorted subarray.
2. Swap it with the leftmost element of the unsorted subarray.
3. Now the leftmost element of unsorted subarray becomes a part (rightmost) of sorted subarray and will not be a part of unsorted subarray.

Diagram

Description automatically generated

### Space and Time Complexity

Selection sort is an in-place comparison sorting algorithm. It has an O(n²) time complexity, which makes it inefficient on large lists, and generally performs worse than the similar insertion sort. The time and space complexity are as follows:

[**Worst complexity**](https://www.google.com/search?sxsrf=ALeKk00P-49MbYH04gQzJYR57zHdrUh85w:1621103882339&q=selection+sort+worst+complexity&stick=H4sIAAAAAAAAAOPgE-LQz9U3MLdIStfSyU620s8uiM8p1y_OLyrJzEuPT8xJzy_KLMnItSrPLyouiU_Ozy3ISa3ILKlcxCpfnJqTmlySmZ-nAFKuAFahgFABAB3xjlxcAAAA&sa=X&ved=2ahUKEwjb1LPIqszwAhXnShUIHXfqA7EQ6BMoADAhegQILhAC)**:**n^2

[**Average complexity**](https://www.google.com/search?sxsrf=ALeKk00P-49MbYH04gQzJYR57zHdrUh85w:1621103882339&q=selection+sort+average+complexity&stick=H4sIAAAAAAAAAOPgE-LQz9U3MLdIStfSy0620s8uiM8p1y_OLyrJzEuPT8xJzy_KLMnItUosSy1KTE-NT87PLchJrcgsqVzEqlicmpOaXJKZn6cA0qAAVaOAUAMA1TV6_GAAAAA&sa=X&ved=2ahUKEwjb1LPIqszwAhXnShUIHXfqA7EQ6BMoADAiegQILRAC)**:**n^2

[**Best complexity**](https://www.google.com/search?sxsrf=ALeKk00P-49MbYH04gQzJYR57zHdrUh85w:1621103882339&q=selection+sort+best+complexity&stick=H4sIAAAAAAAAAOPgE-LQz9U3MLdIStfSzk620s8uiM8p1y_OLyrJzEuPT8xJzy_KLMnItUpKLS6JT87PLchJrcgsqVzEKlecmpOaXJKZn6cAUq0AUqCAUAAArkphG1oAAAA&sa=X&ved=2ahUKEwjb1LPIqszwAhXnShUIHXfqA7EQ6BMoADAjegQILBAC)**:**n^2

[**Space complexity**](https://www.google.com/search?sxsrf=ALeKk00P-49MbYH04gQzJYR57zHdrUh85w:1621103882339&q=selection+sort+space+complexity&stick=H4sIAAAAAAAAAOPgE-LQz9U3MLdIStdSyk620s8uiM8p1y_OLyrJzEuPT8xJzy_KLMnItcpNzc0vqlzEKl-cmpOaXJKZn6cAUqRQXJCYnKqQnJ9bkJNakVlSCQDXQC9yUgAAAA&sa=X&ved=2ahUKEwjb1LPIqszwAhXnShUIHXfqA7EQ6BMoADAkegQIKxAC)**:**1

[**Method**](https://www.google.com/search?sxsrf=ALeKk00P-49MbYH04gQzJYR57zHdrUh85w:1621103882339&q=selection+sort+method&stick=H4sIAAAAAAAAAOPgE-LQz9U3MLdIStdSyk620s8uiM8p1y_OLyrJzEuPT8xJzy_KLMnItcpNLcnIT1nEKlqcmpOaXJKZn6cAUqQAEQcAOuLtyEgAAAA&sa=X&ved=2ahUKEwjb1LPIqszwAhXnShUIHXfqA7EQ6BMoADAlegQIKBAC)**:**[Selection](https://www.google.com/search?sxsrf=ALeKk00P-49MbYH04gQzJYR57zHdrUh85w:1621103882339&q=selection+sort+selection&stick=H4sIAAAAAAAAAOPgE-LQz9U3MLdISlfi1U_XNzRMMzczKzBILtZSyk620s8uiM8p1y_OLyrJzEuPT8xJzy_KLMnItcpNLcnIT1nEKlGcmpOaXJKZn6cAUqQA5-5gZQQAdyIRN10AAAA&sa=X&ved=2ahUKEwjb1LPIqszwAhXnShUIHXfqA7EQmxMoATAlegQIKBAD)

[**Stable**](https://www.google.com/search?sxsrf=ALeKk00P-49MbYH04gQzJYR57zHdrUh85w:1621103882339&q=selection+sort+stable&stick=H4sIAAAAAAAAAOPgE-LQz9U3MLdIStdSyk620s8uiM8p1y_OLyrJzEuPT8xJzy_KLMnItSouSUzKSV3EKlqcmpOaXJKZn6cAUqQAEQcAmAen6kgAAAA&sa=X&ved=2ahUKEwjb1LPIqszwAhXnShUIHXfqA7EQ6BMoADAmegQIKRAC)**:**No

[**Class**](https://www.google.com/search?sxsrf=ALeKk00P-49MbYH04gQzJYR57zHdrUh85w:1621103882339&q=selection+sort+class&stick=H4sIAAAAAAAAAOPgE-LQz9U3MLdIStdSzE620s8uiM8p1y_OLyrJzEuPT8xJzy_KLMnItUrOSSwuXsQqUpyak5pckpmfpwBSowAWBgDrhg2pRgAAAA&sa=X&ved=2ahUKEwjb1LPIqszwAhXnShUIHXfqA7EQ6BMoADAnegQIJxAC)**:**[Comparison sort](https://www.google.com/search?sxsrf=ALeKk00P-49MbYH04gQzJYR57zHdrUh85w:1621103882339&q=Comparison+sort&stick=H4sIAAAAAAAAAOPgE-LQz9U3MLdISlfiBLEsKg0zLLQUs5Ot9LML4nPK9Yvzi0oy89LjE3PS84sySzJyrZJzEouLF7HyO-fnFiQWZRbn5ymAFO1gZQQAWkor808AAAA&sa=X&ved=2ahUKEwjb1LPIqszwAhXnShUIHXfqA7EQmxMoATAnegQIJxAD)

### How Selection Sort Works With Original Diagram

The python code for selection sort can be seen below along with its output. Selection sort improves a small bit on bubble sort by only using one exchange for every iteration though the array. To accomplish this selection sort finds the largest element in the array as it passes though, then it places that element in the correct position. As with a bubble sort, after the first pass, the largest item is in the correct place. After the second pass, the next largest is in place. This process continues and requires n−1n−1 passes to sort n items, since the final item must be in place after the (n−1)(n−1) pass. [12]

Text

Description automatically generated

Graphical user interface

Description automatically generated

## Insertion Sort

### Introduction

With insertion sort the first element of the array is taken as being sorted. The second element is therefore stored in a variable. The variable is then compared with the first element, if the variable is greater than the first element then element in the array (the sorted element), then the variable is placed to the right in front of that element. From this point two elements are now sorted. The third element is then taken and compared with the element to the left. It will be placed behind the element smaller than it, if there is no element smaller than it, then it is placed at the beginning of the array. The process continues. The four steps are detailed below in diagrams obtained from (programiz.com, 2021).

Diagram

Description automatically generatedA picture containing diagram

Description automatically generatedGraphical user interface, diagram

Description automatically generated with medium confidenceGraphical user interface, chart

Description automatically generated

### Space and Time Complexity

|  |  |
| --- | --- |
| **Time Complexity** |  |
| Best | O(n2) |
| Worst | O(n) |
| Average | O(n2) |
| **Space Complexity** | O(1) |
| **Stability** | Yes |

### How Insertion Sort Works With Original Diagram

Below is how the python code for insertion sort works along with the output produced from the input array. Insertion sort assumes that the first element is already sorted and then selected unsorted elements to compare against. If the unsorted element is greater than the first sorted element then the former is placed to the right, or otherwise to the left. This is the essence of the approach of insertion sort.

Text

Description automatically generated

Graphical user interface

Description automatically generated

# Implementation and Benchmarking

Description of the process when implementing the application

## Results of the benchmarking

Discuss how the measured performance of the algorithms differed – were the results similar to what you would expect, given the time complexity of each chosen algorithm? In this section you should use both a table and a graph to summarise the results

## Summary of Results- Graph and Table

# References

[1]. TutorialsPoint.com <https://www.tutorialspoint.com/time-and-space-complexity-in-data-structure>

[2] Danziger, P., 2010. Big o notation. *Source internet: http://www. scs. ryerson. ca/mth110/Handouts/PD/bigO. pdf, Retrieve: April*.

[3]Krueger, R., Rudnicki, P. and Shelley, P., 1999. Asymptotic notation. Part II: Examples and problems. Journal of Formalized Mathematics, 11(199), p.9.

[4]Introduction to Asymptotic Notations Big O (O), Big Omega (Ω) and Big Theta (Ө) <https://madhawacperera.medium.com/introduction-to-asymptotic-notations-big-o-o-big-omega-%CF%89-and-big-theta-%D3%A9-cb84d0d1caf7>

[5]Adrean mejia (2020). How to find time complexity of an algorithm?

<https://adrianmejia.com/how-to-find-time-complexity-of-an-algorithm-code-big-o-notation/>

[6]stackoverflow(2021).https://stackoverflow.com/questions/1517793/what-is-stability-in-sorting-algorithms-and-why-is-it-important#:~:text=A%20sorting%20algorithm%20is%20said,Sort%2C%20Bubble%20Sort%2C%20etc.

[]7unestone.academ(2021). <https://runestone.academy/runestone/books/published/pythonds/SortSearch/TheBubbleSort.html>

[8]Bubble sort. <https://www.studytonight.com/data-structures/bubble-sort>

[9] MergeSort. <https://www.tutorialspoint.com/data_structures_algorithms/merge_sort_algorithm.htm>

[10]. Merge Sort Algorithm. <https://www.interviewbit.com/tutorial/merge-sort-algorithm/>

[11]. Selection Sort. <https://runestone.academy/runestone/books/published/pythonds/SortSearch/TheSelectionSort.html>

[12] Selection Sort. <https://www.hackerearth.com/practice/algorithms/sorting/selection-sort/tutorial/>

[13]. Counting Sort. <https://codezup.com/implementation-of-counting-sort-algorithm-in-python/>

[14]. Inplace sorting. https://www.geeksforgeeks.org/sorting-terminology/

[15]. Insertion Sort. https://www.programiz.com/dsa/insertion-sort