

Simulation and Implementation of Insertion Sort Algorithm to Sort Positive Integers Using C++ Programming Language

¹Tri Dharma Putra and ²Rakhmat Purnomo

^{1,2}Informatics, Universitas Bhayangkara Jakarta Raya, Jakarta, INDONESIA

e-mail :

¹tri.dharma.putra@dsn.ubharajaya.ac.id, ²rakhmat.purnomo@dsn.ubharajaya.ac.id

Publisher's Note: JPPM stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



Copyright: © 2026 by the authors. Submitted for possible open access publication under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

Corresponding Author: Rakhmat Purnomo

Abstract

Sorting refers to the systematic arrangement of data, initially in an unordered state, according to specific rules or criteria such as numerical or alphabetical order. This process is fundamental in various fields, including computer science, statistics, and data management, as it enables efficient searching, analysis, and further processing of information. Here, positive integers are sorted. The insertion algorithm is widely recognized for its effectiveness and is utilized in numerous applications. In the present implementation and simulation, six positive integers (30, 2, 5, 7, 11, 1) were sorted, with each undergoing five processing steps. The ascending order implementation required nine shifts, whereas the descending order required only six shifts. The implementation was conducted using the C++ programming language. Two pseudocode representations were provided, one for ascending and one for descending order. The results demonstrate a successful implementation and simulation in C++. Screenshots are included to verify that the programs executed correctly.

Keywords— Algorithm, Ascending, C++, Descending, Integer, Insertion, Shift

1 Introduction

Sorting is the systematic arrangement of data from an unordered state according to specific criteria, such as numerical or alphabetical order. This process is essential in many fields, including computer science, statistics, and data management, because it facilitates searching, analysis, and further processing of information[1][2].

Insertion sort processes data items one by one and moves each item to its correct location, making it suitable for certain data-processing tasks. The algorithm places each element where it belongs, much like sorting a hand of playing cards. Although its time complexity is $O(n^2)$, this method is not suitable for large datasets, such as student records for a single class[3][4].

The complexity of the algorithm depends on the initial state of the array. If the array is already sorted, the best-case is $O(n)$, where n denotes the size of the array. However, in the lower-dimensional case, the computational time is $O(n^2)$, since each value must be compared with every element in the dataset, so the process becomes quadratic as the dataset grows [5].

The basic mechanism of insertion sort is based on an iterative process: in each pass, an object is selected and compared with other objects to determine its correct position in the array. Sorting is performed "in place" by deleting the array and increasing the previously sorted space.

The choice to use the C++ Programming Language for the implementation of this journal is due to the language's well-established features as a platform-oriented system, which also play an important role in the computational complexity of the study.

The research done here is to provide a real implementation of the C++ Programming Language to be done in dummy dataset of integer numbers, whether it is ascending or it is descending. The research done here is to provide

©2026 Putra and Purnomo

real implementation of C++ Programming Language to be done in dummy dataset of integer numbers, whether it is ascending or it is descending.

2 Research methods

a. Concept of Insertion Sort Algorithm

When playing cards, you generally arrange them in your hand to maintain order. When you draw a new card, you place it in the correct spot to preserve that order.

The insertion sort algorithm works in a similar way: it divides the set into two parts, a sorted part and an unsorted part. The algorithm takes elements from the unsorted part one by one and inserts them into their proper place within the sorted part, thereby maintaining order throughout the process[6][7].

The sorted part is comparable to a neatly arranged hand of cards. The algorithm takes each element from the unsorted part and inserts it into its correct position in the sorted part. It repeats this process until the entire set is sorted. The simplicity of the insertion sort algorithm makes it especially useful for datasets with a low number of queries and swaps required [8][9][10].

b. Steps of Insertion Algorithm

The algorithm works by taking an element from the unsorted position of the array one at a time and placing it in the correct position in the sorted position[11][12][13]. The steps of the insertion algorithm are as follows:

- a. Start with the second element of the array (because the first element is considered sorted).
- b. Compare the current object with the sorted object to determine its correct location.
- c. Shift all elements to the right that are larger than the current one.
- d. Drop the current element in the correct position.
- e. Repeat steps b through d for all elements in the array until the entire array has been sorted

[14][15][16][17][18].

The simplicity of the insertion sort makes it especially useful for datasets with a low number of queries and swaps required.

c. Logic of Programming

Below, please find two pseudo codes of ascending and descending algorithm. The only difference between the two are just the mathematical symbol of '>' and '<' (bigger than and smaller than), in the 'while' code. Where in ascending, we use '>', bigger than, and in the descending, we use '<' smaller than.

Pseudo Code Ascending

Procedure Ascending

```
for (int i = 1; i < n; ++i)
{
    int key = arr[i];
    int j = i - 1;
    while (j >= 0 && arr[j] > key)
    {
        arr[j + 1] = arr[j];
        j = j - 1;
    }
    arr[j + 1] = key;
}
```

Pseudo Code Descending

Procedure Descending

```
for (int i = 1; i < n; ++i)
{
    int key = arr[i];
    int j = i - 1;
    while (j >= 0 && arr[j] < key)
```

```

{
  arr[j + 1] = arr[j];
  j = j - 1;
}
arr[j + 1] = key;
}

```

3 Results and Discussion

In the analysis below, dummy integer numbers are used. These are six numbers we chosen randomly to be implemented and simulated, namely: 30, 2, 5, 7, 11, 1.

a. Ascending Analysis

The first analysis is in ascending order. We use index array 0 until array 5. We put number 2 in temp. In the first process, only one comparison between 30 and 2. Since temp is smaller than 30, it is shifted. Please take a look at table 1., below.

Process 1

0	1	2	3	4	5
30	2	5	7	11	1

Table 1. Process 1

Temp	Check	Shifts
2	Temp<30	Data in array 0 shifts to array 1

Temp position is in array 0

0	1	2	3	4	5
2	30	5	7	11	1

In the second process, please take a look at process 2, below. In table 2. There are 2 comparisons and only the first comparison of temp smaller than 30, we shift the content of the array 1 and array 2.

Process 2

0	1	2	3	4	5
2	30	5	7	11	1

Table 2. Process 2.

Temp	Check	Shifts
5	Temp<30	Data in array 1 shifts to array 2
5	Temp>2	-

Temp position is in array 1

0	1	2	3	4	5
2	5	30	7	11	1

In the third process. Please take a look at table 3., below. There 3 comparisons, but only one comparison that shifted the array 2 and array 3.

Process 3

0	1	2	3	4	5
2	5	30	7	11	1

Table 3. Process 3.

Temp	Check	Shifts
7	Temp<30	Data in array 2 shifts to array 3
7	Temp>5	-
7	Temp>2	-

Temp position is in array 2

0	1	2	3	4	5
2	5	7	30	11	1

In the fourth process. There are 4 comparisons. But only 1 shift happened. The shift is between array 3 and array 4. Please take a look at table 4., below.

Process 4

0	1	2	3	4	5
2	5	7	30	11	1

Table 4. Process 4.

Temp	Check	Shifts
11	Temp<30	Data in array 3 shifts to array 4
11	Temp>7	-
11	Temp>5	-
11	Temp>2	-

Temp position is in array 3

0	1	2	3	4	5
2	5	7	11	30	1

In the fifth process, there are 5 comparison and all of the comparisons results in the index shift of array. Please take a look at table 5., below.

Process 5

0	1	2	3	4	5
2	5	7	11	30	1

Table 5. Process 5.

Temp	Check	Shifts
1	Temp<30	Data in array 4 shifts to array 5
1	Temp<11	Data in array 3 shifts to array 4
1	Temp<7	Data in array 2 shift to array 3
1	Temp<5	Data in array 1 shift to array 2
1	Temp<2	Data in array 0 shift to array 0

Temp position is in array 0

0	1	2	3	4	5
1	2	5	7	11	30

b. Descending Analysis

Again, in analysis of descending order, we use the same dummy integer numbers, 30, 2, 5, 7, 11, 1. Please take a look at table 6., below. Only one comparison and temp's content is 2. There is no shift.

Process 1

0	1	2	3	4	5
30	2	5	7	11	1

Table 6. Process 1.

Temp	Check	Shifts
2	Temp>30	-

Temp position is in array 1

0	1	2	3	4	5
30	2	5	7	11	1

In the second process. There are 2 comparisons. The temp's content is 5. But only one shift happened. Please take a look at table 7., below.

Process 2

0	1	2	3	4	5
30	2	5	7	11	1

Table 7. Process 2.

Temp	Check	Shifts
5	Temp>2	Data in array 2 shifts to array 1
5	Temp<30	-

Temp position is in array 1

0	1	2	3	4	5
30	5	2	7	11	1

In the third process. There are 3 comparisons. The temp's content is 7. Again here, only two shifts happened. Please take a look at table 8., below.

Process 3

0	1	2	3	4	5
30	5	2	7	11	1

Table 8. Process 3.

Temp	Check	Shifts
7	Temp>2	Data in array 2 shifts to array 3
7	Temp>5	Data in array 1 shifts to array 2
7	Temp<30	-

Temp position is in array 1

0	1	2	3	4	5
30	7	5	2	11	1

In the fourth process. The temp's content is 11. It is in array 4. There are 4 comparisons happened. But there are only 3 shifts here. Please take a look at table 9., below. And the last position of temp is in index array 1.

Process 4

0	1	2	3	4	5
30	7	5	2	11	1

Table 9. Process 4.

Temp	Check	Shifts
11	Temp>2	Data in array 3 shifts to array 4
11	Temp>5	Data in array 2 shifts to array 3
11	Temp>7	Data in array 1 shifts to array 2
11	Temp<30	-

Temp position is in array 1

0	1	2	3	4	5
30	11	7	5	2	1

In the last process, the temp's content is 1, it is in array 5. There are only 4 comparisons. However, there is no shift, not even one. Since in the fourth process, it was actually already ordered in descending, but we still need to perform the last process to make sure it was descending all of them. Please take a look at table 10., below. The temp's position is in array 5.

Process 5

0	1	2	3	4	5
30	11	7	5	2	1

Table 10. Process 5.

Temp	Check	Shifts
1	Temp<2	-
1	Temp<5	-
1	Temp<7	-
1	Temp<30	-

Temp position is in array 5

0	1	2	3	4	5
30	11	7	5	2	1

b. Running the C++ Program

Visual Studio Code is used in running this program. Both the ascending code and descending code were executed. The results of the execution were highlighted below.

Result of C++ Running Code, Ascending

Here is the result of dummy integer numbers ascending:

```
PS C:\Users\tridh\AppData\Local\Temp> cd "C:\Users\tridh\AppData\Local\Temp\" ; if ($?) { g++ tempCodeRunnerFile.cpp -o tempCodeRunnerFile } ; if ($?) { .\tempCodeRunnerFile }
1 2 5 7 11 30
```

Result of C++ Running Code, Descending

Here is the result of dummy integer numbers descending:

```
PS C:\Users\tridh\AppData\Local\Temp> cd "C:\Users\tridh\AppData\Local\Temp\" ; if ($?) { g++ tempCodeRunnerFile.cpp -o tempCodeRunnerFile } ; if ($?) { .\tempCodeRunnerFile }
30 11 7 5 2 1
```

From the analysis above, we found that in the ascending order there were 9 swaps, with the total process happening 5 times. In the descending order, there were only 6 swaps. Since we only use six integer figures in this simulation, the array is from 0 to 5. The swaps were done by comparing the temp's content with the next figure in the array. Both of these executions had a total process (iterations) of five times each. Both of them are equal.

Table 11. Comparisons between Ascending and Descending Order.

	Total Shifts	Total Processes
Ascending	9	5
Descending	6	5

4 Conclusion

Insertion sort is a well-established algorithm in data structures. The process involves iteratively comparing elements in the temporary array with adjacent elements. This algorithm is widely applied in various domains. The objective of this research is to implement C++ programming to sort positive integers. Pseudo code of ascending and descending algorithm were presented. In the present analysis and simulation, there were nine and six shifts, respectively, with each process iteration totaling five. The study utilized six sample integer values: 30, 2, 5, 7, 11, and 1. Both ascending and descending orderings were executed successfully, and the results were documented

through screenshots. C++ was selected due to its widespread adoption in both large-scale and small-scale applications, demonstrating its robustness and versatility.

Suggestion

For future works. It is suggested to use any other programming language, like python. Comparisons efficiency with other sorting methods are also suggested.

BIBLIOGRAPHY

- [1] A. S. Sabah *et al.*, “Comparative Analysis of the Performance of Popular Sorting Algorithms on Datasets of Different Sizes and Characteristics,” *IJAER Int. J. Acad. Eng. Res.*, vol. 7, no. 6, pp. 76–84, 2023, [Online]. Available: <https://philarchive.org/archive/SABCAO-2>
- [2] F. R. Wibowo, M. Faisal, P. S. Magisterinformatika, U. Negeri, I. Maulana, and M. Ibrahim, “Comparative Analysis of Sorting Algorithms : TimSort Python and Classical Sorting Methods,” *JISA J. Informatics Sci.*, vol. 07, no. 01, pp. 11–18, 2024, doi: DOI: <https://doi.org/10.31326/jisa.v7i1.1785>.
- [3] A. Ghofur, J. Nazila, N. Holidiyah, and S. Kholifah, “Penerapan Algoritma Insertion Sort Pada Aplikasi Pengolahan Data Mahasiswa Menggunakan Java Di Fakultas Sains Dan Teknologi Prodi Teknologi Informasi,” vol. 4, no. 01, pp. 12–19, 2025, doi: 10.58812/ejpcs.v4i01.
- [4] M. Rizky, S. Barokah, T. Saputra, and T. Sutabri, “Perbandingan Kinerja Algoritma Bubble Sort dan Insertion Sort dalam Pengurutan Data Penjualan UMKM,” *J. Manaj. Inform. Komput.*, vol. 5, no. 1, pp. 184–195, 2025, doi: <https://doi.org/10.51903/h0k3zg44>.
- [5] B. Elshqeirat, M. Altarawneh, and A. Al-oqaily, “Enhanced Insertion Sort by Threshold Swapping,” *IJACSA Int. J. Adv. Comput. Sci. Appl.*, vol. 11, no. 6, pp. 471–476, 2020, doi: 10.14569/IJACSA.2020.0110659.
- [6] R. Purnomo and T. D. Putra, “Theoretical Analysis of Standard Selection Sort Algorithm,” *Sinkron*, vol. 8, no. 2, pp. 666–673, 2023, doi: 10.33395/sinkron.v8i2.12153.
- [7] T. D. Putra and R. Purnomo, “Exchange Sort and Selection Sort Algorithms : Comparison and Theoretical Analysis,” vol. 7, pp. 732–741, 2025, doi: 10.30865/json.v7i2.9306.
- [8] S. N. Fatmaluna, N. G. Aulia, A. Rahma, S. Aulia, and I. P. Pujiono, “Comparison of Memory Efficiency and Computation Time of Bubble Sort , Insertion Sort , and Intro Sort Algorithms Using the C ++ Programming Language,” *J. Artif. Intell. Eng. Appl.*, vol. 5, no. 2, 2026, doi: <https://doi.org/10.59934/jaiea.v5i2.1858>.
- [9] M. N. Ilham, A. F. Setiawan, I. Kholifatun, and M. H. Aldiansyah, “Comparative Analysis of Memory Performance and Processing Time of Five Sorting Algorithms Using C ++ Programming Language,” *J. Artif. Intell. Engeneering Appl.*, vol. 4, no. 3, 2025, doi: <https://doi.org/10.59934/jaiea.v4i3.1051>.
- [10] I. P. Pujiono, E. H. Rachmawanto, and N. A. S. Winarsih, “Array Sorting Algorithm vs Traditional Sorting Algorithm: Memory and Time Efficiency Analysis,” *J. Manaj. Inform.*, vol. 15, no. 1, pp. 47–59, 2025, doi: <https://doi.org/10.34010/jamika.v15i1.13230>.
- [11] F. R. Hakim, H. S. Ismallah, M. L. Firdaus, P. Studi, T. Informatika, and P. N. Pontianak, “Penerapan Algoritma Insertion Sorting Terhadap Data Transaksi Saham per Kota di Indonesia,” in *Prosiding Seminar Nasional dan Teknologi Seri 02*, FST Universitas Terbuka, 2024, pp. 332–341. [Online]. Available: <https://conference.ut.ac.id/index.php/saintek/article/view/2636>
- [12] W. Azizah and S. M. Siregar, “Penerapan Algoritma Insertion Sort dalam Pengurutan Data Menggunakan Bahasa Pemrograman PHP,” *J. Inov. Inform. dan Bisnis Digit.*, vol. 1, no. 3, pp. 83–89, 2026, doi: <https://doi.org/10.59934/jiibd.v1i3.2536>.
- [13] Z. Sitorus, D. Prayogi, M. A. Rizko, A. G. Suteja, and M. Raihan, “Implementation of the Insertion Sort Algorithm to Sort Positive Integers in Ascending Order Using Flowgorithm,” vol. 1, no. 3, pp. 323–328, 2024, doi: 10.30596/jitcese.
- [14] H. Ali, H. Nawaz, A. Maitlo, and I. Soomro, “Performance Analysis of Heap Sort and Insertion Sort Algorithm,” *Int. J. Emerg. Trends Eng. Res.*, vol. 9, no. 5, pp. 580–586, 2021, doi: 10.30534/ijeter/2021/08952021.
- [15] R. Ayazuddin and G. Scholar, “A Comprehensive Study of Sorting Algorithm Performance Using Real-World Dataset Metrics A Comprehensive Study of Sorting Algorithm Performance Using Real-World Dataset Metrics,” pp. 0–12, 2025, doi: 10.20944/preprints202509.2550.v1.
- [16] R. R. Basir, “Analisis Kompleksitas Ruang dan Waktu Terhadap Laju Pertumbuhan Algoritma Heap Sort, Insertion Sort dan Merge dengan Pemrograman Java,” *STRING Satuan Tulisan Ris. dan Inov. Teknol.*, vol. 5,

- [17] T. B. Yulianto, A. J. Taufiq, and A. Suyadi, “Rancang Bangun Pengaturan Intensitas Sinar Uv (Ultraviolet) Dengan Mikrokontroler PIC Untuk Tanaman,” *J. Ris. Rekayasa Elektro*, vol. 1, no. 1, pp. 54–70, 2019, doi: [10.30595/jrre.v1i1.4929](https://doi.org/10.30595/jrre.v1i1.4929).
- [18] A. H. Saptadi and D. W. Sari, “Analisis algoritma insertion sort, merge sort dan implementasinya dalam bahasa pemrograman c++,” *J. INFOTEL (Informatics Telecommun. Electron.)*, vol. 4, no. 2, pp. 10–17, 2012, doi: <https://doi.org/10.20895/infotel.v4i2.103>.