Individual Project: Unit 1

By

James Thomas

Design and Analysis of Algorithms: CS627

Colorado Technical University

07/30/2023

Table of Contents

[Introduction 3](#_Toc141738576)

[Base Code for Sort 3](#_Toc141738577)

[Time and Space complexity 4](#_Toc141738578)

[True Runtime Function 5](#_Toc141738579)

[References 7](#_Toc141738580)

# Introduction

We have often noticed that sorting algorithms have frequent applications in business and technology. Optimizing memory and space usage is crucial for applications to achieve better efficiency with a lower cost. This instance is especially true for applications that process audio files, as these files can be large and require a significant amount of memory to store. This document will review and analyze elements related to time and space efficiency using Python code for sorting audio content in reverse order.

# Base Code for Sort

The code below contains an algorithm designed to reverse the array after the function is entered and the request is initiated.

1. def reverse (array):

2. size=len(array)

3. for i in range(size//2): #defines the i in the bottom half of the array

4. j = size-1-i #finding the corresponding value to swap

5. array[i], array[j] = array[j],array[i] #swaps values

In the first line there is a function that “defines” the process as “reverse” and that we are creating this code with an “array” as mentioned in parentheses. In line 2, we define the size of the array. In line 3, we define the “ i ” in the bottom half of the array. In line 4, “ j ” is the identified for finding the corresponding value to swap. Line 5 is where we execute the swap of elements in the array (Langtangen, 2008).

A diagram of a diagram

Description automatically generatedIn order to visualize line 5 as a function, Figure 1. is a diagram that shows an example with 6 integers. The first three integers would be represented by the “ i ” variable while the last three integers are represented by the letter “ j ” (Langtangen, 2008).

Figure 1.

# Time and Space complexity

To analyze the time complexity of the code, we must review the worst-case time complexity. It is more common to define an algorithm’s worst-case time complexity to guarantee the upper bound of the running time, which in turn, helps in designing reliable and efficient algorithms (Levitin, A. 2016). In this code, the first half of the array swaps with the corresponding elements. The number of iterations depends on the size of the array, which is “N.” The “for loop” iterates “size//2” time, or in the form of an equation: Number of iterations = (N/2). Inside the loop, the code reflects constant time operations and, to be more specific, “ j,” for example. The algorithm continues to swap these particular elements. With these instances in mind, understanding the “for loop” and the constant time operations, we can conclude the time complexity of the “reverse” algorithm as O(N/2). With this in mind, it is more common to consider the dominant factor, which reflects approximated values of how the time complexity changes with large input sizes. The constant factor of “/2” or “1/2” is not significant compared to the linear factor of size, which is “N.” When using big O notation, the constant factors are not taken into account. Therefore, the final time complexity of the algorithm is O(N).

In terms of the space complexity of the algorithm, we will review the memory locations. The input array occupies the “size” memory locations where “Size = len(array)” or, in other words, where size is the length of the array. In the instance of the “for loop,” there are two additional elements: “ i ” and “ j,” which represent one memory location each. The information provided shows that our total number of memory locations can be “Size + 1 + 1 = Size + 2”. Because the variables in the code are constants in amounts of memory, the space complexity of this algorithm is O(1). If we take “Size + 2” and remove the constant, we would be left with O(size) or, in other words: O(1).

# True Runtime Function

To accurately measure the algorithm's runtime, we have a function that efficiently helps to determine the true time :

# Import the `timeit` module to measure the execution time of the `reverse` #function.

import timeit

# Create an input array `x1` containing integers from 0 to 99 using the `range` #function and convert it to a list.

x1 = list(range(100))

# Measure the execution time of the `reverse` function with input array `x1`.

# The `timeit.timeit` function runs the `reverse` function once (number=1) and #returns the time taken in seconds.

# It uses a lambda function to call `reverse(x1)` as the code to be timed.

print(timeit.timeit(lambda: reverse(x1), number=1))

x2 = list(range(1\_000))

print(timeit.timeit(lambda: reverse(x2), number=1))

x3 = list(range(10\_000))

print(timeit.timeit(lambda: reverse(x3), number=1))

x4 = list(range(100\_000))

print(timeit.timeit(lambda: reverse(x4), number=1))

Figure 2

|  |  |
| --- | --- |
| N | Time |
| 100 | 2.33E-05 |
| 1000 | 0.000122 |
| 10,000 | 0.003317 |
| 100,000 | 0.023165 |
| 1,000,000 | 0.228669 |
| 10,000,000 | 2.741688 |

x5 = list(range(1\_000\_000))

print(timeit.timeit(lambda: reverse(x5), number=1))

x6 = list(range(10\_000\_000))

print(timeit.timeit(lambda: reverse(x6), number=1))

In this code, multiple and varying sizes of arrays were used. The varying sizes are represented in Figure 2 as “ N ” and the column of times adjacent where the output of seconds is provided after calling the function (Agha & Nawaz 2021). A Cartesian plane was created on an Excel spreadsheet, and a plot was populated after entering these details into the plot seen in Figure 3.

# References

Agha, F. A., & Nawaz, H. (2021). Comparison of bubble and insertion sort in rust and python language. Internat. l J., 10, 2.

Langtangen, H. P. (Ed.). (2008). Python scripting for computational science. Berlin, Heidelberg: Springer Berlin Heidelberg.

Levitin, A. (2016). Introduction to the Design and Analysis of Algorithms (3rd ed.). Pearson Learning Solutions. <https://bookshelf.vitalsource.com/books/9781323417638>