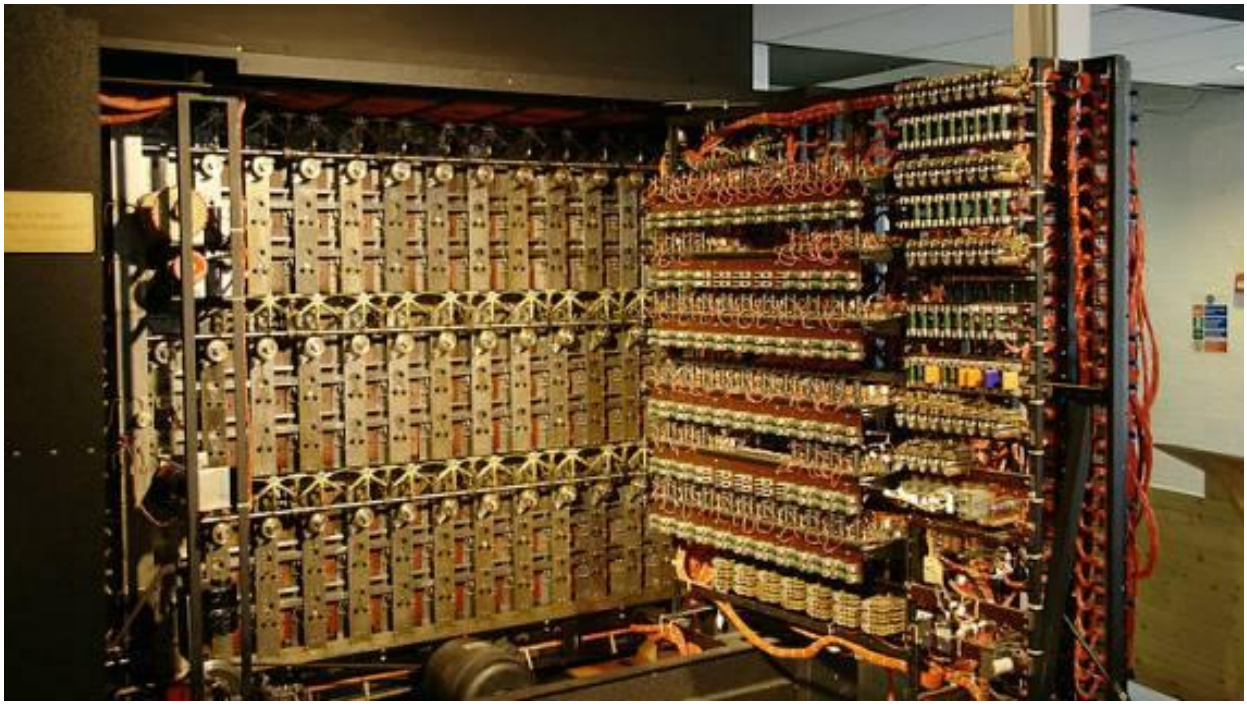

Practice Document

Sorting Algorithms

4/11/2023

—

Pablo Martínez Pedrosa
Programming Module
1º DAM

ÍNDICE

Sorting Algorithms: Introduction

Sorting algorithms are essential in programming and are used in a wide variety of applications. Each algorithm has its own advantages and disadvantages, so it is important to choose the most suitable one based on the size and nature of the data to be sorted. It is crucial to understand the time complexity and behavior of each algorithm to make informed decisions in software development.
In summary, selecting the right sorting algorithm is crucial to ensure efficiency and optimal performance in data manipulation.

Classification of Sorting Algorithms

There are numerous sorting algorithms, each with its own characteristics and complexities.

Sorting algorithms can be classified in the following ways:

- The most common is to classify according to the place where the ordering is carried out.
    - Internal ordering algorithms: in the computer memory.
    - External sorting algorithms: in an external location such as a hard drive.
- By the time it takes to perform the sorting, given inputs already sorted or inversely sorted:
    - Natural sorting algorithms: Takes as little time as possible when the input is sorted.
    - Unnatural sorting algorithms: Takes as little time as possible when the input is inversely sorted.
- For stability: a stable ordering maintains the relative order that the elements with equal keys originally had. For example, if a list sorted by date is reordered in alphabetical order with a stable algorithm, all elements whose alphabetic key is the same will be in date order. Another case would be when the case is not important, but you want that if a key aBC was before AbC, in the result both keys appear together and in the original order: aBC, AbC. When the elements are indistinguishable (because each element is ordered by the full key) stability is not important. Sorting algorithms that are not stable can be implemented to make them stable. One way to do this is to artificially modify the sort key so that the original position in the list participates in the sorting in case of a match.

- The algorithms are distinguished by the following characteristics:
    - Computational complexity (worst case, average case, and best case) in terms of n, the size of the list or array. For this, the concept of order of a function is used and the notation O(n) is used. The best behavior for sorting (if you don't take advantage of the key structure) is O(n log n). The simplest algorithms are quadratic, that is, $O(n^2)$. Algorithms that take advantage of the sort key structure (e.g. bucket sort) can sort in O(kn) where k is the size of the key space. As this size is known a priori, it can be said that these algorithms have linear performance, that is, O(n).
    - Use of memory and other computational resources. O(n) notation is also used.

Examples of Sorting Algorithms

1. Bubble Sort

The bubble sort algorithm is one of the simplest and most basic algorithms. It consists of iterating over the list of elements comparing adjacent pairs and, if they are in the wrong order, exchanging them. This process is repeated until the list is completely sorted.

Advantages:

- Easy to implement and understand.
- Requires little additional memory.

Disadvantages:

- Inefficient for large lists.
- It has a time complexity of O(n^2), which makes it inefficient for large data sets.

2. Selection Sort

The selection algorithm works by dividing the list into two parts: the ordered sublist and the unordered sublist. In each iteration, the smallest element in the unordered sublist is searched and swapped with the first element in the ordered sublist.

Advantages:

- Simple to implement and understand.
- It performs well on small lists.

Disadvantages:

- It has a time complexity of O(n^2), making it inefficient for large data sets.
- Perform the same number of comparisons regardless of the state of the list.

3. Insertion Sort

The insertion sort algorithm constructs a sorted list of elements one by one by taking an element from the list and placing it in the correct position in the sorted sublist.

Advantages:

- Efficient in almost ordered lists.
- It has acceptable performance on small lists.

Disadvantages:

- Inefficient for large data sets.
- It has a time complexity of O(n^2).

4. Merge Sort

El algoritmo de ordenación por fusión (Merge Sort) es un algoritmo de tipo dividir y conquistar. Divide la lista en sublistas más pequeñas, las ordena y luego combina las sublistas para obtener la lista ordenada completa.

Advantages:

- Efficient for large data sets.
- It has a time complexity of O(n log n), which makes it fast in most cases.

Disadvantages:

- Requires additional memory for mixing.

5. Quick Sort

The Quick Sort algorithm is also a divide and conquer algorithm. An element is chosen as the pivot and the list is rearranged so that elements smaller than the pivot are on the left and elements larger than the pivot are on the right. The same process is then applied to the left and right sublists.

Advantages:

- Efficient for large data sets.
- It has an average time complexity of O(n log n), making it fast in most cases.

Disadvantages:

- It can be inefficient on nearly sorted or already sorted lists.

Example of Sorting Algorithm in Java: Selection Sort

```java
public class SelectionSort {
    public static void selectionSort(int[] arr) {
        int n = arr.length;
        for (int i = 0; i < n-1; i++) {
            int minIndex = i;
            for (int j = i+1; j < n; j++) {
                if (arr[j] < arr[minIndex]) {
                    minIndex = j;
                }
            }
            int temp = arr[minIndex];
            arr[minIndex] = arr[i];
            arr[i] = temp;
        }
    }
}
```

Bibliography

https://es.wikipedia.org/wiki/Algoritmo_de_ordenamiento

https://www.freecodecamp.org/espanol/news/algoritmos-de-ordenacion-explicados-con-ejemplos-en-javascript-python-java-y-c/

https://elbauldelprogramador.com/algoritmos-de-ordenacion/

https://es.wikipedia.org/wiki/Ordenamiento_por_selecci%C3%B3n

Programming

Pablo Martínez Pedrosa
Programming Module
1º DAM