Quick Sort Partitioning Algorithm

Shubham Sharma

April 15, 2016

Contents

List of Tables

List of Figures

1 Introduction

Figure 1: Sorting

Sorting is nothing but storage of data in , it can be in ascending or descending order. The term Sorting comes into picture with the term Searching. There are so many things in our real life that we need to search, like a particular record in database, roll numbers in merit list, a particular telephone number, any particular page in a book etc. Sorting arranges data in a sequence which makes searching easier. Every record which is going to be sorted will contain one key. Based on the key the record will be sorted.

There are many techniques for sorting. Implementation of particular sorting depends upon situation. techniques mainly depends on two parameters. First parameter is the execution time of program, which means time taken for execution of program. Second is the space, which means space taken by the program[1].

2 Quick Sort

2.1 Algorithm

Quick Sort is a algorithm. It picks an element as and partitions the given array around the picked pivot. There are many different versions of quickSort that pick pivot in different ways.

- Always pick first element as pivot.
- Always pick last element as pivot.
- Pick a random element as pivot.
- Pick median as pivot.

The key process in quickSort is (). Target of is, given an array and an element x of array as pivot, put x at its correct position in sorted array and put all smaller elements (smaller than x) before x, and put all greater elements (greater than x) after x. All this should be done in linear time[2].

2.2 Partition Algorithm

To complete the , we need to implement the method. We use the following general strategy: First, we arbitrarily choose a[lo] to be the partitioning itemthe one that will go into its final position. Next, we scan from the left end of the array until we find anentry that is greater than (or equal to) the partitioning item, and we scan from the right end of the array until we find an entry less than (or equal to) the partitioning item.[3]

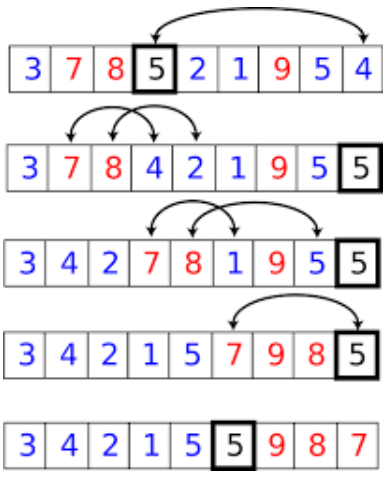

Figure 2: Partitioning

Items that stopped the scans are out of place in the final , so we exchange them. When the scan indices cross, all that we needto do to complete the partitioning process is to exchange the partitioning item a[lo] with the rightmost entry of the left subarray (a[k]) and return its index k.

2.3 Pseudocode

```
QUICKSORT(A, p, r)
1  if p < r
2      q = PARTITION(A, p, r)
3      QUICKSORT(A, p, q − 1)
4      QUICKSORT(A, q + 1, r)
```

```
PARTITION(A, p, r)
1  x = A[r]
2  i = p − 1
3  for j = p to r − 1
4      if A[j] ≤ x
5          i = i + 1
6          exchange A[i] with A[j]
7  exchange A[i + 1] with A[r]
8  return i + 1
```

Figure 3: Pseudocode

2.4 Analysis of Quicksort

2.4.1 Time Complexity

Time taken by QuickSort in general can be written as following[4].

$$T(n) = T(k) + T(n - k - 1) + \theta(n) \tag{1}$$

The first two terms are for two recursive calls, the last term is for the partition process. k is the number of elements which are smaller than pivot.
The time taken by QuickSort depends upon the input array and partition strategy. Following are three cases.

Worst Case: The worst case occurs when the partition process always picks greatest or smallest element as pivot. If we consider above partition strategy where last element is always picked as pivot, the worst case would occur when the array is already sorted in increasing or decreasing order. Following is recurrence for worst case[4].

$$T(n) = T(n - 1) + \theta(n) \tag{2}$$

The solution of above recurrence is O(n^2).

Best Case: The best case occurs when the partition process always picks the middle element as pivot. Following is recurrence for best case[4].

$$T(n) = 2T(n/2) + \theta(n) \tag{3}$$

The solution of above recurrence is O(nLog(n)) [5]

Average Case : To do average case analysis, we need to consider all possible permutation of array and calculate time taken by every permutation. Following is the Time complexity [6]

$$T(n) = 2nlog(n) - 2.84n + O(1) \tag{4}$$

Solution of above recurrence is also O(nLogn) [6]

2.4.2 Auxiliary Space

O(log n) extra space for recurrence

3 Selection Sort

3.1 Algorithm

The selection sort algorithm sorts an array by repeatedly finding the minimum element (considering ascending order) from unsorted part and putting it at the beginning. The algorithm maintains two subarrays in a given array.

1. The subarray which is already sorted.

2. Remaining subarray which is unsorted.

In every iteration of selection sort, the minimum element (considering ascending order) from the unsorted subarray is picked and moved to the sorted subarray.

3.2 Pseudocode

```
Algorithm SelectionSort
Inputs A: Array of Integers;
       N:Integer;
Variables i,j,min: Integer;
Begin
     for i:=0 to N-2 do
       min:=i;
       for j:=i+1 to N-1 do
         if (A[j]<A[min]) then min:=j fi
       od
       swap(A,i,min);
     od
End
```

Figure 4: Pseudocode

3.3 Analysis of Selection Sort

3.3.1 Time Complexity

O(n^2) as there are two nested loops.

3.3.2 Auxiliary Space

$O(1)$

4 Comparison Between Quicksort and Selection Sort

4.1 Time Complexity

Cases	Selection Sort	Quick Sort
Worst Case	$O(n^2)$	$O(n^2)$
Best Case	$O(n^2)$	$O(nlog(n))$
Average Case	$O(n^2)$	$O(nlog(n))$

Table 1: Time Complexity Analysis

4.2 Auxiliary Space

Selection Sort	Quick Sort
$O(1)$	$O(log(n))$

Table 2: Space Complexity Analysis

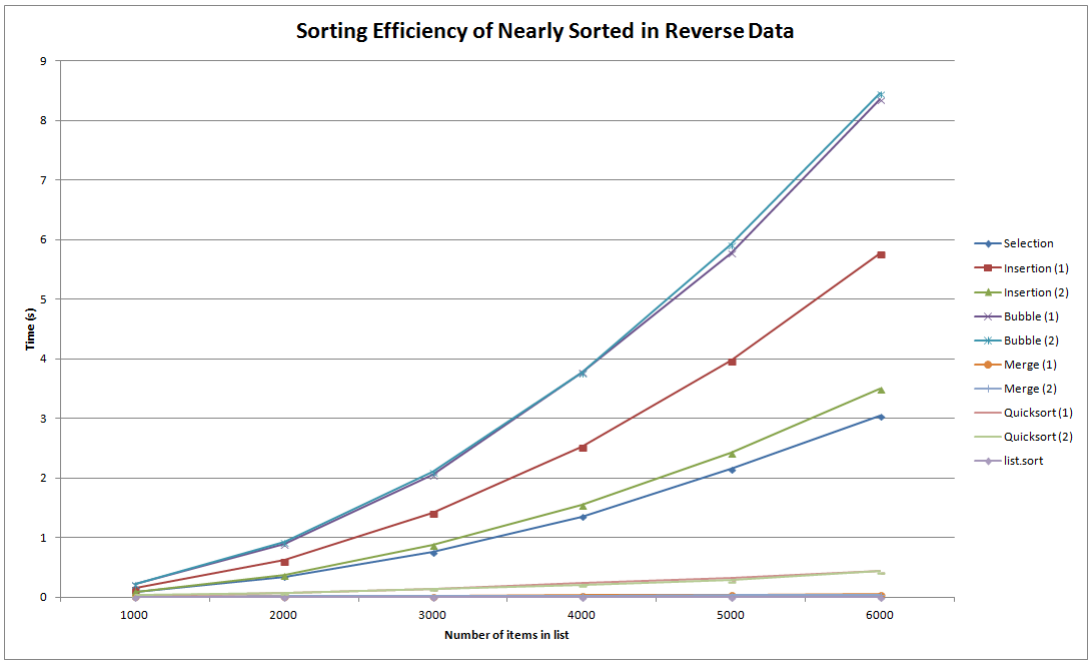

Figure 5: Analysis of Different Sorting Algorithms

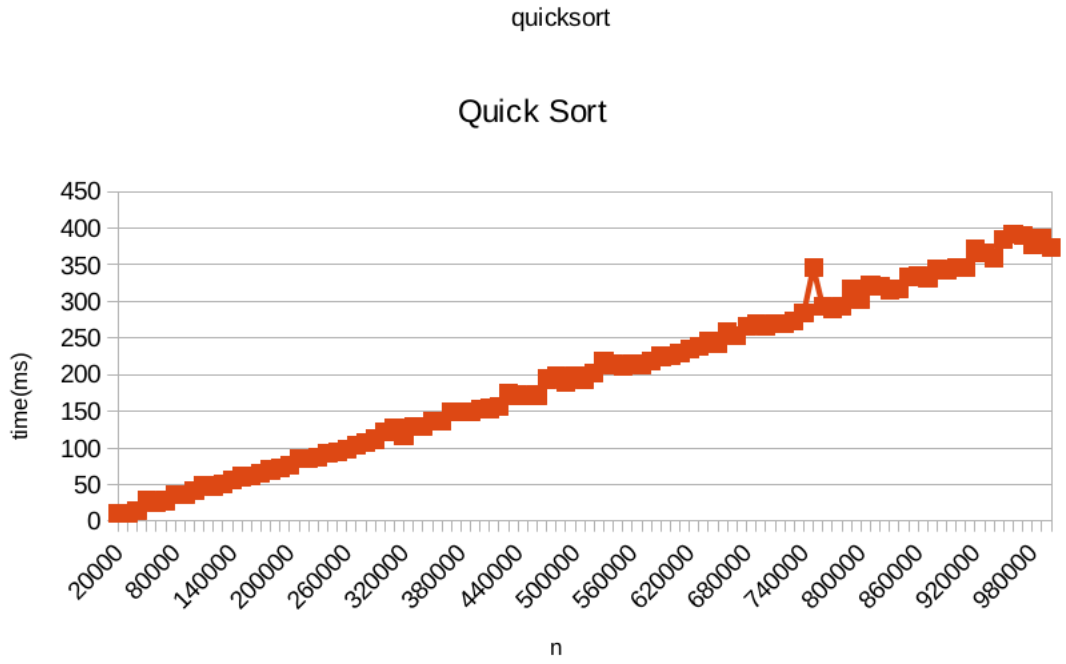

Figure 6: Performance

Quicksort is a well-known sorting algorithm that, on average, makes O(n log n) comparisons to sort n items. However, in the worst case, it makes O(n2) comparisons. Typically, quicksort is significantly faster than other O(n log n) algorithms, because its inner loop can be efficiently implemented on most architectures, and in most real-world data, it is possible to make design choices which minimize the probability of requiring quadratic time. Through the analysis of Time and Space Complexity between Quick Sort and Selection Sort it can be concluded that Quick Sort is generally more efficient that Selection Sort.

Advantages of Quick Sort include the efficient average case compared to any aforementioned sort algorithm, as well as the elegant recusrive definition, and the popularity due to its high efficiency. Disadvantages include the difficulty of implementing the partitioning algorithm and the average efficiency for the worst case scenario, which is not offset by the difficult implementation.

References

[1] Sorting algorithms. http://en.wikipedia.org/wiki/Quicksort.

[2] Mcgraw Hill. *An Introduction to data structures with applications.*

[3] D. E. Knuth. *The Art of Computer Programming.* Pearson Education, 1998.

[4] C. A. R. Hoare. Quicksort. *Computer Journal5(1)*, 1962.

[5] chapter Master Theorem.

[6] Cormen, editor. *Introduction to Algorithms.*

Index