****

**CZ2001 Algorithms Example Class 3**

**Report**

**SS3 Team 2**

**Joshua Martin Tjahjadi**

**Kwek Ai Hui**

**Jason Lee**

**Shantanu Kamath**

**Jordan Kwong**

**Suyash Lakhotia**

SCHOOL OF COMPUTER ENGINEERING

NANYANG TECHNOLOGICAL UNIVERSITY

Contents

[1. INTRODUCTION 3](#_Toc433007266)

[1.1 Description of Problem 3](#_Toc433007267)

[2. IMPLEMENTATION 4](#_Toc433007268)

[2.1 Quick Sort 4](#_Toc433007269)

[2.2 Merge Sort 5](#_Toc433007270)

[3. STATISTICS 6](#_Toc433007271)

[3.1 Quick Sort 6](#_Toc433007272)

[3.2 Merge Sort 6](#_Toc433007273)

[4. CONCLUSION 10](#_Toc433007274)

[5. REFERENCE 11](#_Toc433007275)

# 1. INTRODUCTION

*Description of the Quick Sort and Merge Sort*

## Description of Problem

In this example class, we will be showing the empirical comparison of time efficiency between the two sorting algorithms, Merge Sort and Quick Sort, in ascending order. The data set sizes range from 2000, 4000, 6000, 8000 and 10000. For each of the various sizes, the comparison of the three types of data will be shown. Firstly, a data of randomly generated data sets of integers in the range of

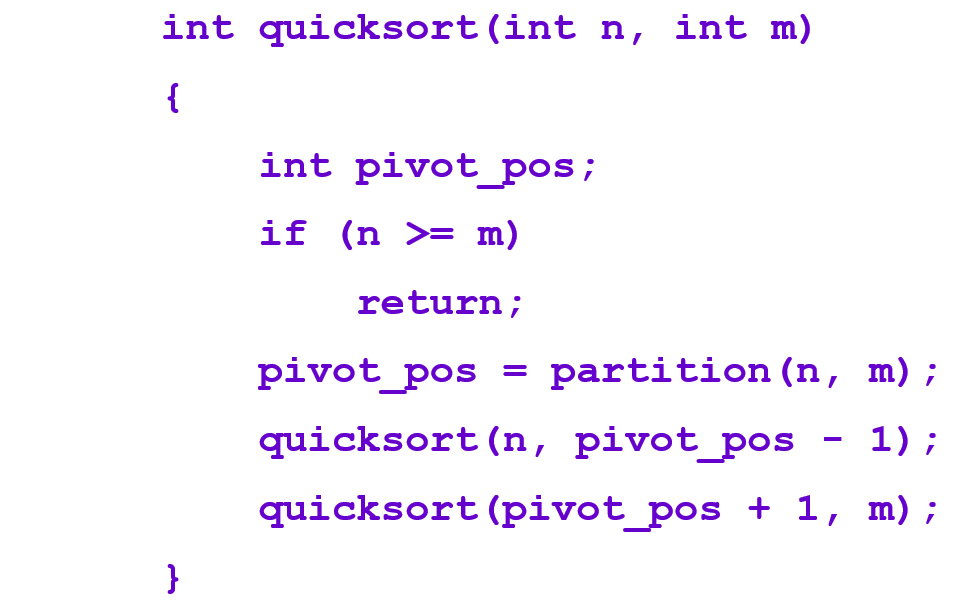
[1 … n]. Secondly, integers from 1 to n, which are sorted in ascending order. Last but not least, integers from n to 1, which are sorted in descending order. The results, number of key comparisons and the computational time, computed will then be saved into a text file for recording.

# 2. IMPLEMENTATION

## 2.1 Quick Sort

Quick Sort is the fastest general purpose in-memory sorting algorithm in the average case. The steps of sorting in quick sort are:

1. Selecting one element in the array as the pivot.
2. Partition list into two sub lists with respect to the pivot such that all elements in left sub list are less than pivot and all elements in the right sub list are greater than or equal to the pivot.
3. Recursively partition until input list has one or zero element.

****

**Figure 1: Quick Sort Algorithm**

As seen in Figure 1, the quick sort algorithm has two parameters n and m, where n represents the integer in the first position of the array while m represents the last position of the array. The terminating condition is seen in the if statement, where the recursive function will exit once the first position of the array equals to the last position of the array.



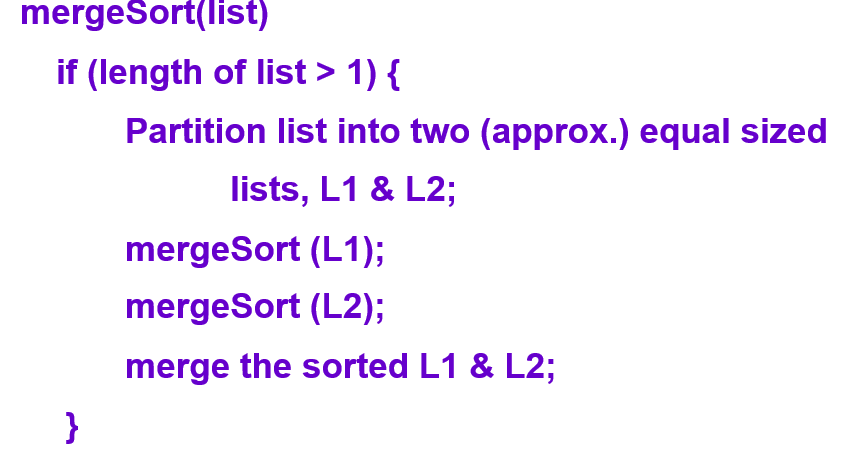
**Figure 2: Pivot of Quick Sort**

In Figure 2, the pivot of our quick sort algorithm is the middle element of the array.

## 2.2 Merge Sort

Merge Sort is a divide-and-conquer sorting algorithm, where a contiguous array of elements is partitioned into two sub arrays. It is the most consistent sorting algorithm, where the best case, average case and the worst case has the same time complexity, nlgn. The steps of sorting in merge sort are:

1. Partition array into two sub arrays.
2. Recursively partition until there are n sublists, containing 1 element each.
3. Merge the sub lists repeatedly, sorting the elements in the meantime, until there is only 1 list remaining.



**Figure 3: Quick Sort Algorithm**

As seen in Figure 3, the merge sort algorithm partitions the list recursive as seen by mergeSort(L1) and mergeSort(L2). This will cause the list to be divided, ending with n sub lists with 1 element in each sub list. The sub lists are then sorted and merged back together. The terminating condition is when the list is merged into one list, which creates the sorted list.

# 3. STATISTICS

## 3.1 Quick Sort

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Data Size | Random | | Ascending | | Descending | |
| Number of Comparison | Average CPU time (nanoseconds) | Number of Comparison | Average CPU time (nanoseconds) | Number of Comparison | Average CPU time (nanoseconds) |
| 2000 | 22318 | 378081 | 18987 | 55614 | 17996 | 49077 |
| 4000 | 45417 | 459641 | 41964 | 116412 | 39974 | 104481 |
| 6000 | 72357 | 1087225 | 67774 | 144127 | 64786 | 144561 |
| 8000 | 103137 | 1417685 | 91917 | 183970 | 87928 | 151235 |
| 10000 | 141142 | 2243798 | 119535 | 237417 | 114548 | 167315 |

## 3.2 Merge Sort

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Data Size | Random | | Ascending | | Descending | |
| Number of Comparison | Average CPU time (nanoseconds) | Number of Comparison | Average CPU time (nanoseconds) | Number of Comparison | Average CPU time (nanoseconds) |
| 2000 | 1673 | 7868894 | 1000 | 31972 | 1000 | 1478697 |
| 4000 | 3342 | 14932111 | 2000 | 125412 | 2000 | 3121683 |
| 6000 | 4999 | 16726038 | 3000 | 107380 | 3000 | 8376899 |
| 8000 | 6687 | 22300891 | 4000 | 233410 | 4000 | 12138751 |
| 10000 | 8384 | 12682896 | 5000 | 183092 | 5000 | 18963273 |

NEED TO CHANGE THIS

# 4. CONCLUSION

In theory, the time complexity for both Merge Sort and Quick Sort is O(n log(n)) on average case scenario.

From the statistics in our experiment, it can be seen that the number of key comparisons for quick sort is larger than merge sort. Despite that fact, the running time for quick sort is much smaller compared to merge sort.

Hence in this case we conclude that quick sort is better than merge sort in terms of running time with large datasets.

# 5. REFERENCE

1. Lecture materials on NTULearn.