CS211FZ Data Structures & Algorithms (II)

Lab 4 – Quicksort and its Performance

Objectives

- Understand how quicksort algorithm works
- Understand how the selection of pivot affects the performance of quicksort
- Reflect the knowledge learned in the class

NOTE:

- **Do NOT use "package" in your source code**
- **You must submit the source code files, i.e., the ".java" files.**
- **You are allowed to use course reference books or class notes during the lab.**
- **Sharing your work with others is NOT allowed.**

Task 1: Understanding Quicksort and the Importance of Choosing a Right Pivot

A *nearly* optimal implementation of the Quicksort algorithm is given on the Moodle course page (***DynamicQuicksort.java***). The implementation uses the original ***Hoare***'s partition algorithm for splitting subarrays. The source code is provided without any comment. Your first task is to write a comment to explain how the Hoare's partition algorithm works, i.e., add comment to the "***hoarePartition***" method.

Task 2: Implementing Partitioning Strategies

In the class, you have learned that choosing a right pivot has a significant impact on the overall performance of the quicksort algorithm. In this task, you are required to write a "***randomizedPartition***" strategy and a "***medianOfThreePartition***" strategy for the Dynamic Quicksort.

Task 3: Dynamic Partition Strategies

In this task, you need to implement a mechanism that allows the quicksort to dynamically select which partition strategy to use based on the conditions outlined below:

1. You might have noticed that for the Median-of-Three strategy to work, a subarray must contain a minimum of three elements. To this end, if a subarray contains less than or equal to **20** elements, you should use an insertion sort to sort the subarray.
2. If a subarray contains less than or equal to **64** elements, you should use the "***randomizedPartition***" strategy.
3. If a subarray contains less than or equal to **256** elements, you should use the default ***Hoare***'s partitioning strategy.
4. Otherwise, use the "***medianOfThreePartition***" strategy.

Task 4: Test your Algorithms

Read the file "***Lab4_RandomNumbers_1M.txt***" into your program. The file contains 1 million integer numbers. Use your dynamic quicksort algorithm to sort the numbers, and then print out how long your algorithm took to sort 1 million numbers in milliseconds.

Task 5 (OPTIONAL, 3 extra marks): Compare the Performance and Further Optimization

1. Try to sort the same file using an insertion sort, which takes $O(N^2)$ time on average. Compare its performance with the Quicksort. (1 extra mark)
2. As mentioned, the quicksort implementation is nearly optimal, but still there are spaces for you to further optimize the code that will result in even better performance. (2 extra marks)

NOTE: the extra marks can only be added to your continuous assessments.