Analysis of Algorithms

BLG 335E

Project 1 Report

Şafak Özkan Pala
palas21@itu.edu.tr

Faculty of Computer and Informatics Engineering
Department of Computer Engineering
Date of submission: 26.11.2023

1. Implementation

1.1. Implementation of QuickSort with Different Pivoting Strategies

1.1.1. Implementation Details

First naive quicksort function is implemented. **takeDataFromCsvFile()** function is implemented first to get data from .csv files. Data stored in a vector of pairs which contains name and population **pair<string name, int population>**. After getting data from the file and store it using <chrono> starting time measured and **quickSort()** function started with k=1 to use naive quicksort. Since k=1 program skips the insertionsort part and only quicksort part of algorithm is works. In this section according the pivoting strategy, pivot is replaced by chosen pivot. In 'l' mode normal quicksort algorithm works pivot is the rightmost element of subarray. In 'r' mode pivot index is chosen random than swapped with previous pivot of subarray. In 'm' mode three index are chosen random than medium of these three is swapped with previous pivot. If the verbose mode is on, in partition of subarray **logger()** function is called and it will overwrite the file named 'log.txt' and log the pivot and the subarray. At the end pivoting strategy and quicksort time will be printed.

1.1.2. Recurrence Relation

Recurrence relation of this **quickSort()** function is considered in 3 different modes:

- **For mode 'l':** Recurrence relation of this function is in the worst case:

$$T(n) = T(n-1) + \Theta(n) \tag{1.1}$$

  in the average and best case:

$$T(n) = 2T(n/2) + \Theta(n) \tag{1.2}$$

- **For mode 'r' and 'm':** Recurrence relation of these functions are always the same because whatever the order of the input data is we always choose pivots randomly. So recurrence relation of these functions are:

$$T(n) = 2T(n/2) + \Theta(n) \tag{1.3}$$

1.1.3. Time and Space Complexity

Time complexity of this function can calculate by master method using recurrence relation.

- **For mode 'l':** Time complexity for the worst case it is: $O(n^2)$ for the average and best case it is: $O(n\log(n))$
  Space complexity for the worst case it is: $O(n)$ for the average and best case it is: $O(\log(n))$

- **For mode 'r' and 'm')** Time complexity for all cases it is: $O(n\log(n))$
  Space complexity for all cases it is: $O(log(n))$

	Population1	**Population2**	**Population3**	**Population4**
Last Element	89293740 ns	1443036117 ns	602097537 ns	4926545 ns
Random Element	4194948 ns	2855417 ns	3872431 ns	4981370 ns
Median of 3	4046170 ns	3259709 ns	3807761 ns	4700102 ns

Table 1.1: Comparison of different pivoting strategies on input data.

Result: This experiment shows that to make this function more stabil for every input we need to increase randomness of the chosen pivot because if the chosen pivot is the median of the sorted subarray to make partition even .it will take optimum time and increasing randomness helps.

1.2. Hybrid Implementation of Quicksort and Insertion Sort

1.2.1. Implementation Details

This section of implementation is similar to the previous one with one small different. **quickSort()** function's base case is determined by subarray size and threshold given by user. When function hits the base case insertion sort for this subarray is called and sorts the subarray. This method helps user to sort faster the almost sorted subarrays.

1.2.2. Recurrence Relation

Recurrence relation of this hybrid **quickSort()** function is same as previous one because k is a constant value. It does not affect the time and space complexity of the function. Function is considered in 3 different modes:

- **For mode 'l':** Recurrence relation of this function is in the worst case:

$$T(n) = T(n-1) + \Theta(n) + \Theta(k^2) \tag{1.4}$$

  in the average and best case:

$$T(n) = 2T(n/2) + \Theta(n) \tag{1.5}$$

- **For mode 'r' and 'm':** Recurrence relation of these functions are always the same because whatever the order of the input data is we always choose pivots randomly.

So recurrence relation of these functions are:

$$T(n) = 2T(n/2) + \Theta(n) \tag{1.6}$$

1.2.3. Time and Space Complexity

This function has the same recurrence relation as non-hybrid implemtation of quicksort. Time complexity of this function can calculate by master method using recurrence relation.

- **For mode 'l':** Time complexity for the worst case it is: $O(n^2)$ for the average and best case it is: $O(n\log(n))$
  Space complexity for the worst case it is: $O(n)$ for the average and best case it is: $O(\log(n))$
- **For mode 'r' and 'm')** Time complexity for all cases it is: $O(n\log(n))$
  Space complexity for all cases it is: $O(logn)$

Threshold (k)	1	10	25	50	100
Population4	7654249 ns	9283538 ns	10810242 ns	13929496 ns	19838538 ns

Threshold (k)	250	500	1000	2000
Population4	37977047 ns	75524391 ns	129497848 ns	266532996 ns

Table 1.2: Comparison of different thresholds on the hybrid QuickSort algorithm on input data.

Result: If datas are nearly sorted it is expected that insertion sort gives better result than quicksort. Thus if k value increases, array partition should be bigger therefore insertion sort's effect is shown more than quicksort. If array is not sorted than quicksort gives better result than insertion sort and when insertion sort affect bigger part of the array time complexity increase.