On the Convergence of the Dual-Pivot Quicksort Process

Mahmoud Ragab¹, Beih El-Sayed El-Desouky², Nora Nader²

¹Department of Mathematics, Faculty of Science, Al Azhar University, Cairo, Egypt
²Department of Mathematics, Faculty of Science, Mansoura University, Mansoura, Egypt

Email: meragab23@yahoo.mail, bdesouky@mans.edu.eg, scinora2012@yahoo.com

Received 19 November 2015; accepted 26 January 2016; published 29 January 2016

Abstract

Sorting an array of objects such as integers, bytes, floats, etc. is considered as one of the most important problems in Computer Science. Quicksort is an effective and wide studied sorting algorithm to sort an array of \( n \) distinct elements using a single pivot. Recently, a modified version of the classical Quicksort was chosen as standard sorting algorithm for Oracles Java 7 routine library due to Vladimir Yaroslavskiy. The purpose of this paper is to present the different behavior of the classical Quicksort and the Dual-pivot Quicksort in complexity. In particular, we discuss the convergence of the Dual-pivot Quicksort process by using the contraction method. Moreover we show the distribution of the number of comparison done by the duality process converges to a unique fixed point.

Keywords

Randomized Quicksort, Convergence, Dual-Pivot Quicksort Process, Running Time Analysis

1. Introduction

Quicksort is one of the important sorting algorithms. Hoare [1] proposed an algorithm depended on selecting an arbitrary element from the array. This element called a pivot element such that Quicksort algorithm used for parting the arrays into two sub-arrays: those smaller than the pivot and those larger than the pivot [2].

After that Quicksort depends on recursive sorting of the two subarrays. Later Sedgewick studied several variants. Regnier [3] studied the limiting distribution of the number of comparisons done by Quicksort algorithm when suitably normalized. It converges with uncertain unknown limit. The first accounts were computed by Hennequin who proved that this distribution is not a normal distribution. The limiting distribution is characterized by a stochastic fixed point equation [4] [5]. The cost of the Quicksort algorithm depends on the position of the se-
lected pivot. There are many cases to choose the pivot element. The worst-case, the best-case and the average case express the performance of the algorithm. We will discuss some of them and for more details; we refer to Ragab [6] and [7]. The worst-case occurs when the pivot is the smallest (or largest) element at partitioning on array of size \( n \), yielding one empty sub-array, one element (pivot) in the correct place and one sub-array of size \( n - 1 \). So, the two sub-arrays are lopsided so this case is defined by worst case [8]. We found the recursion depth is \( n - 1 \) levels and the complexity of Quicksort is \( O(n^2) \). The best case occurs when the pivot is in the median at each partition step, i.e. after each partitioning, on array of size \( n \), yielding two sub-arrays of approximately equal size and, the pivot element in the middle position takes \( n \) data comparisons [9]. There are various methods to choose a good pivot, like choosing the First element, Last element, and Median-of-three elements (selection three elements, and find the median of these elements), and so on. In this case the Quicksort algorithm selects a pivot by random selection each time. This choice reduces probability that the worst-case ever occurs. The other method, which essential prevents the worst case from ever occurring, picks a pivot as the median of the array each time. When we chose the pivot, we compare all other elements to it and we have \( n - 1 \) comparisons to divided the array. The choosing of the pivot divided the array into one sub-array of size 0 and one sub-array of size \( n - 1 \), or into a sub-array of size 1 and other one of size \( n - 2 \), and so on up to a sub-array of size \( n - 1 \) and one of size 0. We have \( n \) possible positions and each one is equality in probability \( 1/n \). Hennequin studied comparisons for array by using Quicksort with \( r \) pivots when \( r = 2 \), same comparisons as classic Quicksort in one partitioning. When \( r > 2 \), he found the problem is compiled. Yaroslavskiy [10] introduced a new implementation of Dual-pivot Quicksort in Java 7’s runtime library. In 2012, Wild and Nabel denoted exact numbers of swaps and comparisons for Yaroslavskiy’s algorithm [10]. In this paper, our aim is to analyze the running time performance of Dual-pivot Quicksort. The limiting distribution of the normalized number of comparisons required by the Dual-pivot Quicksort algorithm is studied. It is known to be the unique fixed point of a certain distributional transformation \( T \) with zero mean and finite variance.

We show that using two pivot elements (or partitioning to three subarrays) is very efficient, particularly on large arrays. We propose the new Dual-pivot Quicksort scheme, faster than the known implementations, which improves this situation (see in [11] and [12]). The implementation of the Dual-pivot Quicksort algorithm has been inspected on different inputs and primitive data types.

The new Quicksort algorithm uses partitioning a source array \( T[g] \), where \( g \) is primitive array which we need to sort it. Such as int, float, byte, char, double, long and short, to three parts defined by two pivot elements \( p \) and \( q \) (and therefore, there are pointers \( A, B, C \) and left and right indices of the first and last elements respectively). The aim of this paper is to present such a version arising from an algorithm depending on the work in [13] and [14]. The Dual-pivot Quicksort is explained clearly in [15] and it works as follow:

1. For small arrays (length < 17), use the Insertion sort algorithm [10].
2. Choose two pivot elements \( p \) and \( q \). We can get, for example, the first element \( g[left] \) as \( p \) and the last element \( g[right] \) as \( q \).
3. \( p \) must be less than \( q \), otherwise they are swapped. So, we have the following parts.
   - **Part I** with indices from \( left + 1 \) to \( A - 1 \) with elements, which are less than \( p \).
   - **Part II** with indices from \( A \) to \( B - 1 \) with elements, which are greater or equal to \( p \) and less or equal to \( q \).
   - **Part III** with indices from \( C + 1 \) to \( right - 1 \) with elements greater than \( q \).
   - **Part IV** contains the rest of the elements to be examined with indices from \( B \) to \( C \).  
   4. The next element \( g[B] \) from the part IV is compared with two pivots \( p \) and \( q \), and placed to the corresponding part I, II, or III.
   5. The pointers \( A, B, \) and \( C \) are changed in the corresponding directions.
   6. The steps 4 - 5 are repeated while \( B < C \).
   7. The pivot element \( p \) is swapped with the last element from part I, the pivot element \( q \) is swapped with the first element from part III.
   8. The steps 1 - 7 are repeated recursively for every part I, part II, and part III as in Figure 1.

![Figure 1.](image) Graph explains the dual-pivot quicksort algorithm.
2. Run-Time Performance

In this section, we introduce some running time of the Dual-pivot Quicksort. An efficient procedure is described by Vasileios Iliopoulos and David B. Penman [13], where they analyze the Dual pivot Quicksort algorithm. Their approach can be here provided and for more details we refer to [13] and [14]. First we introduce the algorithm of it and we compare between it and the classical Quicksort as follows [16].

The following graphs show the relation between the size of array which need to sort and the time of complexity which represent by the number of comparisons and swaps as in Figure 2. We found the Dual-pivot Quicksort is faster than classical Quicksort.

3. The Dual-Pivot Quicksort Average Case Analysis

To find the distributional equation, we note the following: for the underlying process, there are two parts. The first part is partitioning and the second is the total number of comparisons to sort an array of \( n \geq 2 \) keys, when the pivot is a uniform random variable \( \{1, 2, 3, \ldots, n\} \) is equal to the number of comparisons to sort the sub-array of on \( U_n = 1 \) keys below the first pivot [17].

In addition, we need to compute the number of comparisons to sort the sub-array of \( n - U_n \) elements above the second pivot plus the number of comparisons to sort the sub-array of \( U_n - U_{n-1} \) elements between the first and the second pivot.

Plus \( 2n - U_n - 2 \) comparisons done to partition the array which come from when the all elements compare one time with the first pivot and the remain elements compare two times with the second and the first pivot. Therefore,

![Figure 2. Comparison between the classical Quicksort and the Dual-pivot Quicksort.](image-url)
where the random variables \( X_{U_{n-1}}, X_{U_{n-2}}^* \) and \( X_{U_{n-2}} - U_{n-1} \) are identically distributed and independent of \( U_{n_1} \) and \( U_{n_2} \). Here \( d \) refers to the equality in distribution.

The array is partitioned into three subarrays one with \( U_{n_1} \) keys smaller than the first pivot, a subarray of \( U_{n_2} - U_{n_1} \) keys between two pivots and the part of \( n - U_{n_2} \) elements greater than the second pivot. The algorithm is then recursively applied to each of these subarrays. The number of comparisons during the first stage is

\[
A_n = 1 + \left[ (U_{n_1} - 1) + 2(U_{n_2} - U_{n_1} - 1) + 2(n - U_{n_2}) \right],
\]

where \( U_{n_1} = 1, \ldots, n - 1 \) and \( U_{n_2} = U_{n_1} + 1, \ldots, n \). Using [11], the average value of \( A_n \) can be calculated as follow:

\[
E(A_n) = \frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{n} \left[ 1 + \left( (i-1) + 2(j-i-1) + 2(n-j) \right) \right]
\]

\[
= \frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{n} (2n - i - 2) = \frac{2}{n(n-1)} \left( \frac{5}{6} n^3 - 2n^2 + \frac{7}{6} n \right) = \frac{5n - 7}{3}
\]

### 4. Expected Number of Comparisons

Here by Equation (1) and using [13], it is easy to determine the recurrence for the expected number of comparisons due to the duality as follow:

\[
E(X_n) = \frac{5n - 7}{3} + \frac{2}{n(n-1)} \left( \sum_{i=1}^{n} \sum_{j=1}^{n} E(X_{i-1}) + \sum_{i=1}^{n} \sum_{j=1}^{n} E(X_{n-i})^* + \sum_{i=1}^{n} \sum_{j=1}^{n} E(X_{j-i-1}) \right).
\]

Since the three double sums above are equal, then the recurrence becomes

\[
E(X_n) = \frac{5n - 7}{3} + \frac{6}{n(n-1)} \sum_{i=1}^{n} (n-i) E(X_{i-1}),
\]

setting \( a_n = E(X_n) \),

\[
a_n = \frac{5n - 7}{3} + \frac{6}{n(n-1)} \sum_{i=1}^{n} (n-i) a_{i-1}, n \geq 2.
\]

By initial conditions we have \( a_0 = a_1 = 0 \). Multiplying both sides by \( \binom{n}{2} \), we obtain

\[
\binom{n}{2} a_n = \binom{n}{2} \left( \frac{5n - 7}{3} + \frac{6}{n(n-1)} \sum_{i=1}^{n-1} (n-i) a_{i-1} \right) = \frac{n(n-1)(5n-7)}{6} + 3 \sum_{i=1}^{n-1} (n-i) a_{i-1}.
\]

We introduce a difference operator for the solution of this recurrence. The operator is defined by

\[
\Delta F(n) := F(n+1) - F(n).
\]

And for higher orders

\[
\Delta^k F(n) := \Delta^{k-1} F(n+1) - \Delta^{k-1} F(n).
\]

Thus, we have
\[ \Delta \left( \frac{n+1}{2} \right) a_n = \left( \frac{n+1}{2} \right) a_{n+1} - \left( \frac{n}{2} \right) a_n = \frac{5n^2 - 3n}{2} + \sum_{j=1}^{n-1} a_j. \]

By definition (2),
\[ \Delta^2 \left( \frac{n+1}{2} \right) a_n = \Delta \left( \frac{n+1}{2} \right) a_{n+1} - \Delta \left( \frac{n}{2} \right) a_n = \left( \frac{n+2}{2} \right) a_{n+2} - 2 \left( \frac{n+1}{2} \right) a_{n+1} + \left( \frac{n}{2} \right) a_n. \]

\[ (n+1)(n+2)a_{n+2} - 2n(n+1)a_{n+1} + n(n-1)a_n = 2(5n+1). \]

Dividing by \( (n+1)(n+2) \), we obtain the telescoping recurrence
\[ \frac{(n+2)a_{n+2} - (n-2)a_{n+1}}{n+2} = \frac{(n+1)a_{n+1} - (n-3)a_n + 2(5n+1)}{(n+1)(n+2)}, \]
which yields
\[ \frac{(n+2)a_{n+2} - (n-2)a_{n+1}}{n+2} = 2 \sum_{j=1}^{n} \frac{5j+1}{(j+1)(j+2)} = \frac{18}{n+2} + 10H_{n+1} - 18n. \]

Multiplying by \( \frac{(n-1)(n-2)(n-3)}{24} \), this recurrence is transformed to a telescoping one
\[ \left( \frac{n}{4} \right) a_n = \left( \frac{n-1}{4} \right) a_{n+1} + \frac{18(n-1)(n-2)(n-3)}{24} + 10 \left( \frac{n}{4} \right) H_{n+1} - 18 \left( \frac{n}{4} \right). \]

By using maple V. Iliopoulos and D. B. Penman [13] get
\[ \sum_{j=1}^{n} (j-1)(j-2)(j-3) = \binom{n}{4}. \]

And for the other sums in Equation (3):
\[ \sum_{j=1}^{n} \left( \frac{j}{4} \right) H_j = \binom{n+1}{5} \left( H_{n+1} - \frac{1}{5} \right). \]

Therefore,
\[ \sum_{j=1}^{n} \left( \frac{j}{4} \right) H_{j+1} = \sum_{j=1}^{n} \left( \frac{j}{4} \right) \left( H_j - \frac{1}{j} \right) = \sum_{j=1}^{n} \left( \frac{j}{4} \right) H_j - \sum_{j=1}^{n} \left( \frac{j}{4} \right) \frac{1}{j} = \binom{n+1}{5} \left( H_{n+1} - \frac{1}{5} \right) - \frac{1}{24} \sum_{j=1}^{n} (j-1)(j-2)(j-3) = \binom{n+1}{5} \left( H_{n+1} - \frac{1}{5} \right) - \binom{n}{4} \frac{1}{4}. \]

Now the equation becomes
\[
\begin{aligned}
\left(\begin{array}{c}
\frac{n}{4}
\end{array}\right) a_n &= \left(\begin{array}{c}
\frac{9}{2}
\end{array}\right) + 10 \left(\begin{array}{c}
\frac{n+1}{5}
\end{array}\right) \left[H_{n+1} - \frac{1}{5}\right] - \left(\begin{array}{c}
\frac{n}{4}
\end{array}\right) \left(\begin{array}{c}
\frac{1}{4}
\end{array}\right)
- 18 \left(\begin{array}{c}
\frac{n+1}{5}
\end{array}\right)
\end{aligned}
\]
\[
\begin{aligned}
a_n &= \left(\begin{array}{c}
\frac{9}{2}
\end{array}\right) + 10 \left(\begin{array}{c}
\frac{n+1}{5}
\end{array}\right) \left[H_{n+1} - \frac{1}{5}\right] - \left(\begin{array}{c}
\frac{n}{4}
\end{array}\right) \left(\begin{array}{c}
\frac{1}{4}
\end{array}\right)
- 18 \frac{n+1}{5}.
\end{aligned}
\]

Finally, the expected number of comparisons, when two pivots are chosen is
\[
a_n = 2(n+1)H_n - 4n - 2n \log(n),
\]

where \(H_n\) is the harmonic number defined by \(H_n = \sum_{k=1}^{n} \frac{1}{k}\) (see [18] and [19]).

This is the same value of the expected number of comparisons, when one pivot chosen in the classical Quicksort [20]. Note that this result for the dual Quicksort is identical with the expected number of comparisons in [13].

5. Variance of Comparisons

The main result of this section was obtained by [13] (see following results for explanation and notation). Now we compute the variance of the number of comparisons by Dual-pivot Quicksort. Recall that
\[
A_n = 2n - i - 2 \text{ and } E(A_n) = \frac{5n - 7}{3}.
\]

From Equation (1), we have
\[
P(X_n = t) = \left(\begin{array}{c}
\frac{n}{2}
\end{array}\right) \sum_{i=1}^{n-1} \sum_{j=1}^{n-i} \sum_{l=1}^{n-i} \sum_{m=1}^{n-i-j} P(A_{i} + X_{i,j} + X_{j-i-1} + X_{n-j} = t),
\]

noting that the resulting subarrays are independently sorted, then we get
\[
P(X_n = t) = \left(\begin{array}{c}
\frac{n}{2}
\end{array}\right) \sum_{i=1}^{n-1} \sum_{j=1}^{n-i} \sum_{l=1}^{n-i} \sum_{m=1}^{n-i-j} \left[P(X_{i,j} = l)P(X_{j-i-1} = m)P(X_{n-j} = t - m - l - 2n + i + 2)\right].
\]

Letting
\[
f_n(z) = \sum_{t=0}^{\infty} P(X_n = t)z^t,
\]
be the ordinary probability generating function for the number of comparisons needed to sort \(n\) keys, we obtain
\[
f_n(z) = \left(\begin{array}{c}
\frac{n}{2}
\end{array}\right) \sum_{i=1}^{n-1} \sum_{j=1}^{n-i} z^{2n-i-2} f_{j-i-1}(z) f_{n-j}(z)
\]

It holds that \(f_n(1) = 1\) and \(f_n'(1) = 2(n+1)H_n - 4n\). Moreover, the second order derivative of Equation (6) evaluated at \(z = 1\) is recursively given by
\[
f_n''(z) = \frac{2}{n(n-1)} \left[\sum_{i=1}^{n-1} \sum_{j=1}^{n-i} (2n-i-2)^2 - \sum_{i=1}^{n-1} \sum_{j=1}^{n-i} (2n-i-2) + 2 \sum_{i=1}^{n-1} \sum_{j=1}^{n-i} (2n-i-2) E(X_{i,j})
\]
\[
+ 2 \sum_{i=1}^{n-1} \sum_{j=1}^{n-i} E(X_{i,j}) E(X_{j,i-1}) + 2 \sum_{i=1}^{n-1} \sum_{j=1}^{n-i} E(X_{i,j}) E(X_{n-j}) + 2 \sum_{i=1}^{n-1} \sum_{j=1}^{n-i} E(X_{j-i-1}) E(X_{n-j})
\]
\[
+ \sum_{i=1}^{n-1} \sum_{j=1}^{n-i} f_{i-1}'(1) + \sum_{i=1}^{n-1} \sum_{j=1}^{n-i} f_{j-i-1}'(1) + \sum_{i=1}^{n-1} \sum_{j=1}^{n-i} f_{n-j}'(1)\right].
\]
By simple manipulation of indices, the sums of the products of expected values are equal. The double sum of the product of the mean number of comparisons can be simplified as follows:

\[
\sum_{i=0}^{n} \sum_{j=0}^{n} E(X_{i-1})E(X_{n-j}) = \sum_{i=0}^{n-1} E(X_{i-1}) \sum_{j=0}^{n-i} E(X_{j})
\]

\[
= \sum_{i=0}^{n-1} \left[ 2(i+1)H_{i+1} - 4(i-1) \left( \frac{n+i+1}{2} \right) + \frac{n-i-5(n-i)^2}{2} \right].
\]

We find

\[
\sum_{i=1}^{n-1} \left( \frac{n-i+1}{2} \right) H_{i+1} H_{n-i} = \sum_{i=1}^{n-1} \left( \frac{(i-1)+1}{2} \right) H_{i+1} H_{n-i}
\]

\[
= \sum_{i=1}^{n-1} \left( \frac{n-i}{2} \right) H_{i+1} H_{n-i} + \sum_{i=1}^{n-1} \left( \frac{n-i}{2} \right) H_{i+1} H_{n-i}
\]

\[
+ \sum_{i=1}^{n-1} (i-1)(n-i)H_{i+1} H_{n-i} + \sum_{i=1}^{n-1} (n-i)H_{i+1} H_{n-i}.
\]

The recurrence becomes

\[
f_n^*(1) = 2(n+1)(n+2)(H_n^2 - H_n^{(2)}) - H_n \left( \frac{17}{3} n^2 + \frac{47}{3} n + 6 \right)
\]

\[
+ \frac{209}{36} n^2 + \frac{731}{36} n + \frac{13}{6} + \frac{6}{n(n-1)} \sum_{i=1}^{n-1} (n-i)f_n^*(1),
\]

where \( H_n^{(2)} \) is the second order harmonic number defined by \( H_n^{(2)} := \sum_{k=1}^{n} \frac{1}{k^2} \) (see [17] and [18]). Letting \( d_n = f_n^*(1) \) and subtracting \( \left( \frac{n}{2} \right) d_n \) from \( \left( \frac{n+1}{2} \right) d_{n+1} \), we get

\[
\Delta \left( \frac{n}{2} \right) d_n = 4n(n+1)(n+2)(H_n^2 - H_n^{(2)}) - \frac{nH_n}{9}(84n^2 + 198n + 42)
\]

\[+ 3 \sum_{i=1}^{n} d_{i+1} + \frac{n}{6}(79n^2 + 231n + 14).
\]

By using the identity [4]

\[
H_n^2 - H_n^{(2)} = \left( H_n^2 - H_n^{(2)} \right) + \frac{2H_n}{n+1}.
\]

It holds that

\[
\Delta^2 \left( \frac{n}{2} \right) d_n = 12(n+1)(n+2)(H_n^2 - H_n^{(2)}) - H_n \left( 20n^2 + 32n - 12 \right) + 17n^2 + 37n + 3d_n.
\]

The previous equation is the same as

\[
\left( \frac{n+2}{2} \right) d_{n+2} - 2 \left( \frac{n+1}{2} \right) d_{n+1} + \left( \frac{n}{2} \right) d_n.
\]

And our recurrence becomes

\[
(n+1)(n+2)d_{n+2} - 2n(n+1)d_{n+1} + n(n-1)d_n
\]

\[
= 2 \left( 12(n+1)(n+2)(H_n^2 - H_n^{(2)}) - H_n \left( 20n^2 + 32n - 12 \right) + 17n^2 + 37n + 3d_n \right).
\]
Dividing by \((n+1)(n+2)\), we obtain the telescoping recurrence
\[
\frac{(n+2)d_{n+2}-(n-2)d_{n+1}}{n+2} = \frac{(n+1)d_{n+1}-(n-3)d_n}{n+1} + 2\left(12\left(H_{n+1}^2-H_{n+1}^{(2)}\right) - \frac{H_n}{(n+1)(n+2)}\left(20n^2+32n-12\right) + \frac{17n^2+37n}{(n+1)(n+2)}\right).
\]
which is equivalent to
\[
\begin{align*}
nd_n &- (n-4)d_{n-1} \\
&= (24n^2+4n)\left(H_n^2-H_n^{(2)}\right) - H_{n+1} \left(88n^2-60n-8\right) + 122n^2-142n+20.
\end{align*}
\]
Again as before, multiplying both sides by \(\frac{(n-1)(n-2)(n-3)}{24}\), the recurrence telescopes with solution
\[
f_n^*(1) = 4(n+1)^2 \left(H_{n+1}^2-H_{n+1}^{(2)}\right) - \frac{4H_{n+1} \left(n+1\right)(4n+3)}{24} + 23n^2 + 33n + 12.
\]
Using the well known fact that
\[
Var(C_n) = f_n^*(1) + f_n^*(1) - \left(f_n^*(1)\right)^2,
\]
the variance of the number of key comparisons of the Dual-pivot Quicksort is (see [17] [19] and [20])
\[
7n^2 - 4(n+1)^2 H_{n+1}^{(2)} - 2(n+1)H_n + 13n,
\]
where \(H_n^{(2)}\) is the second order harmonic number defined by (see [18] and [19])
\[
H_n^{(2)} = \sum_{k=1}^{n} \frac{1}{k^2}.
\]

6. Asymptotic Distribution

In this section, we show the convergence results which are essential for the main purpose.

Defining a random variables
\[
Y_n := \frac{X_n - E(X_n)}{n}, \quad n \geq 1
\]
Equation (8) can be rewritten in the following form
\[
Y_n = \frac{1}{n} \left(2n - U_{n-1} - 2 + X_{n-1} + X_{n-1}^* + X_{n-1} - E(X_n)\right),
\]
and so,
\[
Y_n = \frac{1}{n} \left(\frac{X_{n-1} - E(X_{n-1})}{U_{n-1}} \left(n_{n-1} - 1 \right) + \frac{X_{n-1}^* - E(X_{n-1}^*)}{n-U_{n-1}} \left(n - U_{n-1} \right) + \frac{\bar{X}_{n-2} - E(\bar{X}_{n-2})}{U_{n-1} - U_{n-1}} \left(U_{n-1} - U_{n-1} - 1 \right) + 2n - U_{n-1} - 2 + E(X_{n-1}) + E(X_{n-1}^*) + E(\bar{X}_{n-2} - U_{n-1}) - E(X_n)\right).
\]
By a simple manipulation, one gets

\[
Y_n = Y_{n-1} \left( \frac{U_n - 1}{n} \right) + Y_{n-2} \left( \frac{n - U_{n-1}}{n} \right) + \sum_{k_2=2}^{n-1} \left( \frac{U_n - U_{n-1}}{n} \right) + C_n \left( U_{n-1} + U_{n-2} \right),
\]

where the cost function \( C_n \left( U_{n-1} + U_{n-2} \right) \) is given as and it seems to be like in [6] and [7], and given by

\[
C_n(i, j) = \frac{2n - i - 2}{n} + \frac{1}{n} \left( E(X_{i+1}) + E(X_{*}) + E(X_{j-1}) - E(X_n) \right). \tag{10}
\]

Now, we show the random vector \( \left( \frac{U_n}{n}, \frac{U_{n-1}}{n} \right) \) converges to a uniformly distributed random vector \( (U_1, U_2) \) on \([0,1]\). So,

\[
\left( \frac{U_n}{n}, \frac{U_{n-1}}{n} \right) \xrightarrow{p} (U_1, U_2). \tag{11}
\]

Here \( (U_1, U_2) \) is uniformly distributed random vector on \([0,1]\). The moment generating function of \( (U_{n-1}, U_{n-2}) \) is given by

\[
M \left( U_{n-1}, U_{n-2} \right) (s_1, s_2) = \sum_{k_1=1}^{n-1} \sum_{k_2=1}^{n-1} e^{s_1 k_1 + s_2 k_2} P \left[ (U_{n-1}, U_{n-2}) = (K_1, K_2) \right]
\]

\[
= M_{U_{n-1}}(s_1) M_{U_{n-2}}(s_2)
\]

\[
= \frac{1}{n} \left( e^{s_1} - 1 \right) \frac{1}{n} \left( e^{s_2} - 1 \right)
\]

\[
= \left( \frac{2}{n(n-1)} \right)^2 \frac{e^{s_1} - e^{s_1/n}}{e^{s_1} - 1} \frac{e^{s_2} - e^{s_2/n}}{e^{s_2} - 1}.
\]

For the random vector \( \left( \frac{U_n}{n}, \frac{U_{n-1}}{n} \right) \),

\[
M \left( \frac{U_n}{n}, \frac{U_{n-1}}{n} \right) (s_1, s_2) = M_{U_n}(s_1) M_{U_{n-1}}(s_2)
\]

\[
= \frac{1}{n} \left( e^{s_1} - e^{s_1/n} \right) \frac{1}{n} \left( e^{s_2} - e^{s_2/n} \right)
\]

\[
= \left( \frac{2}{n(n-1)} \right)^2 \frac{e^{s_1} - e^{s_1/n}}{e^{s_1} - 1} \frac{e^{s_2} - e^{s_2/n}}{e^{s_2} - 1}.
\]

Now, the random vector \( (U_1, U_2) \) has the following moment generating function

\[
M_{U_1, U_2}(s_1, s_2) = \int_0^1 e^{s_1 t + s_2 t^2} dt = \int_0^1 e^{s_1 t} \int_0^t e^{s_2 t^2} dt = \left( \frac{e^{s_1} - 1}{s_1} \right) \left( \frac{e^{s_2} - 1}{s_2} \right). \tag{12}
\]
By the above Equation (12) the moment generating function of \( \frac{U_m}{n} \) is an approximation to the average value of \( e^{\alpha x} \) over the interval \([0,1]\). The moment generating function of \( \frac{U_n}{n} \) is an approximation to the average value of \( e^{\alpha x} \) over the interval \([0,1]\) (see \([8]\) and \([9]\)).

For the cost function

\[
C_n(i, j) = \frac{2n-i-2}{n} + \frac{1}{n} \left[ E(X_{i+1}) + E(X_{i+j}^*) + E(X_{j-i+1}^*) - E(X_n) \right].
\]

By using asymptotically, the expected complexity of Dual-pivot Quicksort is \( 2n \log n \) given in Equation (4), it follows that

\[
\lim_{n \to \infty} C_n \left( n \cdot \frac{U_m}{n}, n \cdot \frac{U_n}{n} \right) = \lim_{n \to \infty} \left[ \frac{2n-U_m-2}{n} + \frac{1}{n} \left( E(X_{U_m-1}) + E(X_{U_n}^*) + E(X_{U_n}^*) - E(X_n) \right) \right]
\]

\[
= \lim_{n \to \infty} \left[ \frac{2n-U_m-2}{n} + \frac{1}{n} \left( 2 \frac{n \cdot U_m}{n} - 1 \right) \log (U_m-1) + 2 \left( n - \frac{n \cdot U_n}{n} \right) \log (n-U_n) \right]
\]

\[
+ 2 \left( \frac{n \cdot U_m}{n} - \frac{n \cdot U_n}{n} - 1 \right) \log (U_m-U_n-1) - 2n \log (n)
\]

\[
= 2 + 2e_1 \log (e_1) + 2(1-e_1) \log (1-e_1) + 2(e_2-e_1) \log (e_2-e_1), \quad \forall e_1, e_2 \in [0,1].
\]

Thus \( C_n \left( n \cdot \frac{U_m}{n}, n \cdot \frac{U_n}{n} \right) \) converges to some \( C(U_1, U_2) \), defined as

\[
C(U_1, U_2) = 2 + 2U_1 \log (U_1) + 2(1-U_1) \log (1-U_1) + 2(U_2-U_1) \log (U_2-U_1),
\]

where \( U_1 \) and \( U_2 \) are uniformly distributed random variables on \([0,1]\). Therefore, if we assume for moment that \( Y_n \) converges in distribution to some \( Y \), we obtain

\[
L(Y) = L\left( YU_1 + Y^* (1-U_2) + (U_2-U_1) + C(U_1, U_2) \right).
\]

Here \( U_1, U_2, Y, Y^* \) and \( Y^* \) are independent. \( Y^* \) and \( Y \) have the same distribution as \( Y \). Finally we show that \( Y_n \) converges in fact to the fixed point \( Y \).

Let \( D \) be the space of distribution functions \( F \) with finite second moments \( \int x^2 dF(x) < \infty \) and the first moment \( \int x dF(x) = 0 \). We use the Wasserstein metric \([4]\) on \( D \).

\[
d(F, G) = \inf \|X-Y\|_2,
\]

where \( \| \cdot \|_2 \) denotes the \( L_2 \) norm. Defining a map \( T : D \to D \) by

\[
T(F) = L\left( \tau_1 X + (1-\tau_2) X^* + \tau_2 (\tau_2 - \tau_1) + C(\tau_1, \tau_2) \right),
\]

where \( X, X^*, X, \tau_1 \) and \( \tau_2 \) are independent.

\[
L(X) = L(X^*) = L(X) = F.
\]

Here \( \tau_1 \) and \( \tau_2 \) are uniformly distributed random variables on \([0,1]\) and \( C \) is a map defined as \( C : [0,1] \to \mathbb{R} \). We have to refer to Roesler (see in \([4]\) \([21]\) and \([22]\)) for the main idea for the next lemma.

**Lemma 1**

The map \( T : D \to D \) is a contraction on \( (D, d) \) and has a unique fixed point. Moreover, every sequence
\( F, T(F), T^2(F), \cdots, F \in D \) converges in the \( d \)-metric to fixed point of \( T \).

**Proof**

Let \( F \) and \( G \) are in \( D \)

\[
T(F) = L\left( \tau_1 X + (1 - \tau_2) X^* + \bar{X} (\tau_2 - \tau_1) + C(\tau_1, \tau_2) \right).
\]

\[
T(G) = L\left( \tau_1 Y + (1 - \tau_2) Y^* + (\tau_2 - \tau_1) \bar{Y} + C(\tau_1, \tau_2) \right).
\]

\[
L(X) = L\left( X^* \right) = L\left( \bar{X} \right) = F.
\]

\[
L(Y) = L\left( Y^* \right) = L\left( \bar{Y} \right) = G.
\]

The random variables \( \tau_1, \tau_2, X, X^* \) and \( \bar{X} \) are independent. Also \( \tau_1, \tau_2, Y, Y^* \) and \( \bar{Y} \) are independent. Here \( \tau_1 \) and \( \tau_2 \) are uniformly distributed on \([0,1]\). Then

\[
d^2\left( S(F), S(G) \right)
\leq \left\| T(F) - T(G) \right\|
\leq \left\| T(F) - T(G) \right\|
\leq E\left( (X - Y)^2 \right) + E\left( (X^* - Y^*)^2 \right)
\leq E\left( (X - Y)^2 \right) + E\left( (X^* - Y^*)^2 \right)
\leq \frac{1}{3} E\left( (X - Y)^2 \right) + \frac{1}{3} E\left( (X^* - Y^*)^2 \right) + \frac{1}{6} E\left( (\bar{X} - \bar{Y})^2 \right)
\leq \frac{5}{6} E\left( (X - Y)^2 \right),
\]

where

\[
E\left( \tau_1^2 \right) = \int_0^{\tau_1} \tau_1^2 \, d\tau_1 = \left[ \frac{\tau_1^3}{3} \right]_0^1 = \frac{1}{3},
\]

\[
E\left( (1 - \tau_2)^2 \right) = -\int_0^{(1 - \tau_2)^2} (1 - \tau_2)^2 \, d\tau_2 = \left[ -\frac{(1 - \tau_2)^3}{3} \right]_0^1 = \frac{1}{3},
\]

\[
E\left( (\tau_1 - \tau_2)^2 \right) = \int_0^{(\tau_1 - \tau_2)^2} (\tau_1 - \tau_2)^2 \, d\tau_1 \, d\tau_2 = \int_0^{(\tau_1 - \tau_2)^2} (\tau_1^2 - 2\tau_1\tau_2 + \tau_2^2) \, d\tau_1 \, d\tau_2
\]

\[
= \int_0^{(\tau_1^2 - 2\tau_1\tau_2 + \tau_2^2)} (\tau_1^2 - 2\tau_1\tau_2 + \tau_2^2) \, d\tau_2 = \int_0^{(\tau_1^2 - 2\tau_1\tau_2 + \tau_2^2)} (\frac{1}{3} - 2\tau_1 + \tau_2) \, d\tau_2
\]

\[
= \left[ \frac{\tau_2^3}{3} - \frac{(\tau_2^2)}{2} + \frac{(\tau_2)}{3} \right]_0^1 = \frac{1}{3} - \frac{1}{2} + \frac{1}{3} = \frac{1}{6}.
\]

Taking the infimum over all possible \( (X, Y) \) we obtain

\[
d\left( T(F), T(G) \right) \leq \frac{5}{\sqrt{6}} d(F, G),
\]

using Banach fixed point theorem completes the proof (also see[13]).

**Acknowledgments**

We thank the editor and the referee for their comments.
References


Appendix

A1. The Dual-Pivot Quicksort Algorithm [15]

DUALITY-PIVOT QUICKSORT (G, left, right)
// Sort G[left, ⋯, right] (including end points).
1) If right – left < M // i.e. the sub-array has n ≤ M elements
2) INSERTIONSORT (G, left, right)
3) Else
4) If G[left] > G[right]
5) p := G[right]; q := G[left]
6) Else
7) p := G[left]; q := G[right]
8) End If
9) A := left + 1; B := right – 1; B := A
10) While B ≤ C
11) If G[B] < p
12) Swap G[B] and G[A]
13) A := A + 1
14) Else
15) If G[B] ≥ q
16) While A[C] > q and B < C do C := C − 1 End While
17) If G[C] ≥ p
18) Swap G[B] and G[C]
19) Else
20) Swap G[B] and G[B]; Swap G[B] and G[A]
21) A := A + 1
22) End if
23) C := C − 1
24) End if
25) End if
26) B := B + 1
27) End While
28) A := A − 1; C := C + 1
30) G[right] := G[C]; G[C] := q
31) DUALITY-PIVOT QUICKSORT (G, left, L − 1)
32) DUALITY-PIVOT QUICKSORT (G, L + 1, g − 1)
33) DUALITY-PIVOT QUICKSORT (G, g + 1, right)
34) End if

A2. The Implementation of the New Dual-Pivot

Here’s the implementation of the new Dual-Pivot (Yaroslavskiy) in java:

```java
public main void sort(double[] g) {
    sort(g, 0, g.length);
}
public main void sort(double[] g, double fromIndex, double toIndex) {
    rangeCheck(g.length, fromIndex, toIndex);
    Yaroslavskiy(g, fromIndex, toIndex - 1, 3);
}
private main void rangeCheck(double length, double fromIndex, double toIndex) {
    if (fromIndex > toIndex) {
        throw new IllegalArgumentException("fromIndex > toIndex");
    }
```

```java
throw new IllegalArgumentException("fromIndex > toIndex");
```
if (fromIndex < 0) {
    throw new ArrayIndexOutOfBoundsException(fromIndex);
}
if (toIndex > length) {
    throw new ArrayIndexOutOfBoundsException(toIndex);
}

private main void swap(double[] g, double i, double j) {
    int tem = g[i];
    g[i] = g[j];
    g[j] = tem;
}

private static void dualPivotQuicksort(double[] g, double left, double right, double div) {
    double lenth = right - left;
    if (lenth < 27) { // insertion sort for tiny array
        for (double i = left + 1; i <= right; i++) {
            for (int j = i; j > left && g[j] < g[j - 1]; j--) {
                swap(g, j, j - 1);
            }
        }
        return;
    }
    int third = lenth / div;
    // "medians"
    int s1 = left + third;
    int s2 = right - third;
    if (s1 <= left) {
        s1 = left + 1;
    }
    if (s2 >= right) {
        s2 = right - 1;
    }
    if (g[s1] < g[s2]) {
        swap(g, s1, left);
        swap(g, s2, right);
    } else {
        swap(g, s1, right);
        swap(g, s2, left);
    }
    // choose the pivots
    double first pivot = g[left];
    double second pivot = g[right];
    // pointers
    double less = left + 1;
    double great = right - 1;
    // sorting the array by the Dual pivot Quicksort
    for (int k = less; k <= great; k++) {
        if (g[k] < first pivot) {
            swap(g, k, less++);
        } else if (g[k] > second pivot) {
            swap(g, k, less--);
        } else {
            swap(g, k, less);
        }
    }
}
until (k > great && g[great] < second pivot) {
    great--;
}
swap(g, k, great--);
if (g[k] < first pivot) {
    swap(g, k, less++);
}

// swaps
double Dis = great - less;
if (Dis < 13) {
    div++;
}
swap(g, less - 1, left);
swap(g, great + 1, right);
// recursive the algorithm for the arrays
Yaroslavskiy(g, left, less - 2, div);
Yaroslavskiy(g, great + 2, right, div);
// subarray
if (first pivot < second pivot) {
    Yaroslavskiy(g, less, great, div);
}
}