**QuickSort Algorithm**

1. **Introduction**

This paper describes the Quicksort algorithm, its implementation, running time and applications in the real world. Sorting is one of the most important functions in many computer science applications to improve search performance, and Quicksort was placed among 10 most importance algorithms for sorting big data numbers [2]. Quicksort was developed by Tony Hoare in 1959 and published in 1961, and Quicksort has been considered as one of the most efficient and fastest sorting algorithms in practice.

Quicksort is also a divide-and-conquer algorithm by dividing the input array into two smaller subarrays and recursively sort these subarrays. Generally, Quicksort has three steps:

* ***Divide****:* Selecting a pivot among the element in the array (usually the rightmost element) and dividing the array into two smaller arrays, where all elements on the left side array are smaller than the pivot and all elements on the right side of the array are greater than the pivot.
* ***Conquer****:* Recursively call the quicksort for these sub-arrays.
* ***Combine****:* No work is needed since the subarrays are sorted.

**Pseudo Code for Quicksort: [4]**

***QUICKSORT*** (A, p, r)

1: if p < r then

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

**Quicksort running time:**

***Worst case***: Happens when the partitions are unbalanced after dividing the big array that one sub array has empty element, while another one has n – 1 elements. The recurrence T(n) = Ɵ(*n*2).

**Best case**: Happens when the partitions are balanced after dividing the big array that two subarrays are equal in size. The recurrence T(n) = Ɵ(*n*lg*n*).

**Average case**: The recurrence T(n) = Ɵ(*n*lg*n*).

We can avoid the worst case by using the alternative algorithm of Quicksort, called Randomized Quicksort, where we choose a random element of the pivot. Thus, we can highly avoid the case that the pivot is the smallest or the largest element in the array comparing to choosing the leftmost or rightmost element in the array. [2]

Quicksort is an in-place sorting algorithm, so we do not need to create other arrays to work with like other algorithms like Merge-sort.

1. **Applications**

***Graphic Processors Quicksort***

The first application for the Quicksort is an efficient parallel algorithmic implementation of Quicksort, named GPU-Quicksort. The research of Cederman shows that GPU-Quicksort is better in the performance comparing to two best GPU-sorting algorithms: GPUSort and radix sort. GPU-Quicksort minimizes the number of bookkeeping, by constraining threads and coalescing read, and keeps inner-thread synchronization low, by using two-pass design, to take advantage of high bandwidth GPUs. [1]

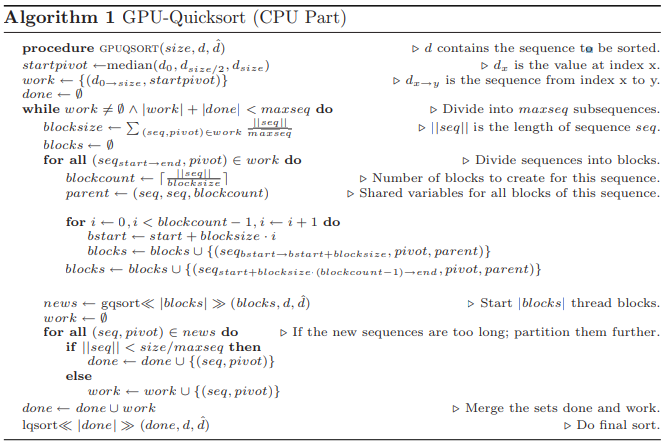


Figure 2.1: GPU-Quicksort Algorithm

Form <Cederman and Rsigas 2009>. Reproduced without permission.

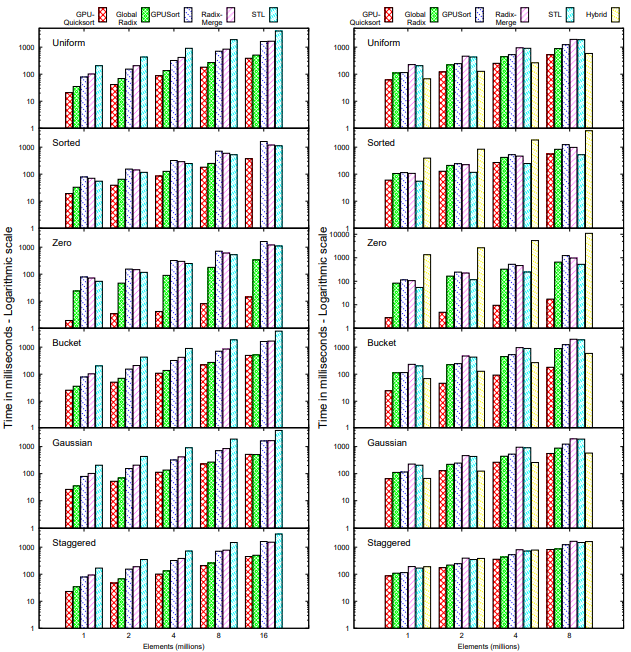


Figure 2.2: Results on the 8800GTX Figure 2.3: Results on the 8600GTX

Form <Cederman and Rsigas 2009>. Reproduced without permission.

As we can see from the results in figures 2.2 and 2.3, GPU-Quicksort has provided the best performance compared to other algorithms. It only picks a pivot that available in the distribution, which will be marked as sorted element, and it does not require to have an extra step to check if the array is completely sorted. [1]

***Preprocessing Large Data Sets***

The second application of Quicksort aims to test the new Quicksort implementation with random reference element using unfavorable poses of large data set. The author has run those tests on a quad core AMD opteron processor, with random samples of 100 series in each class of frequencies including unfavorable positioning. [3]

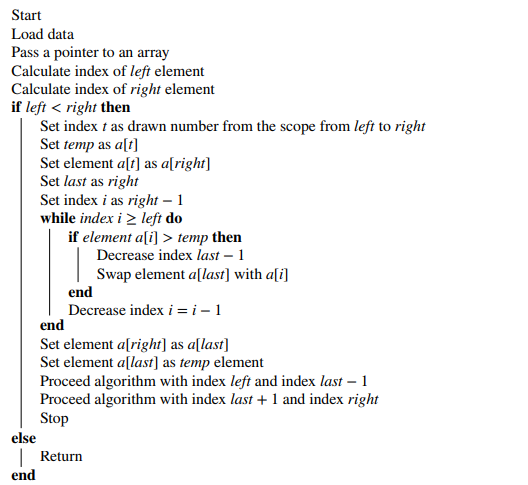


Figure 2.4: Quicksort with random reference element

Form < Woźniak, Marszałek, Gabryel, and Nowicki 2016>. Reproduced without permission.

This algorithm also provides the running time for average case is Ɵ(*n*lg*n*). In the figure 2.5 shows that the classic version of Quicksort faster when sorting the sets of less than 1000 elements, and new version of Quicksort would be more stable for the set over 100,000 elements. [3]

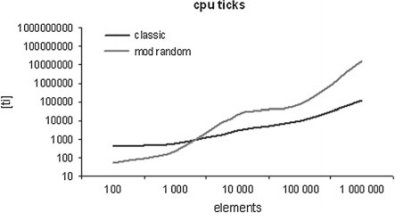
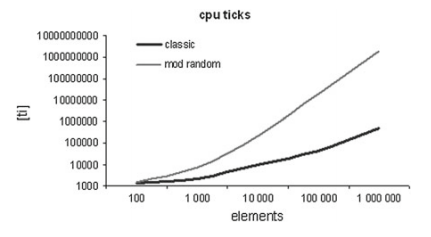


Figure 2.5: Comparing the value of characteristics of CPU clock cycles

Form < Woźniak, Marszałek, Gabryel, and Nowicki 2016>. Reproduced without permission.

Overall, the Quicksort with random reference element method allows to sort a large data set with unfavorable input without complication and would increase the stability of the algorithm. So the Quicksort process would be more efficient. [3]

1. **Conclusions**

In conclusion, Quicksort is better than other sorting algorithms in practice. Even with the worst case of Ɵ(*n*2), we can use Randomized Quicksort to avoid the worst case in quicksort most of the time. Quicksort has a benefit that most other algorithms do not have is cache-efficient since Quicksort is an in-place sorting algorithm. For the applications, Quicksort algorithm can be optimized for the high bandwidth of GPUs usage, and for the large data set with more stable and efficient.

1. **References**

[1] D. Cederman and P. Tsigas, “GPU-Quicksort,” Journal of Experimental Algorithmics, vol. 14, p. 1.4, Dec. 2009.

[2] M. Ragab and U. Roesler, “The Quicksort process,” Stochastic Processes and their Applications, vol. 124, no. 2, pp. 1036–1054, Feb. 2014.

[3] M. Woźniak, Z. Marszałek, M. Gabryel, and R. K. Nowicki, “Preprocessing Large Data Sets by the Use of Quick Sort Algorithm,” Advances in Intelligent Systems and Computing, pp. 111–121, 2016.

[4] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, “Introduction to Algorithms,” 3rd ed. Cambride, pp.170-180, 2009.