Due Date: April 19, 2019 (F), 11:59pm, on Blackboard

We have studied various algorithms in the class so far. In this assignment, you will choose a favorite algorithm and do some in-depth learning. The algorithm can be any one that we have learned in the textbook or in the class, or can be the ones in the homework. *The goal of this assignment is for you to research the applications of this algorithm.* You will need to read scholarly academic papers (e.g., published on IEEE and/or ACM). Online references may additionally be cited, but you must include citations to at least three published articles.

Then you are going to write a report to summarize your findings. In your report, cite the papers you read as follows: you must list all the sources you use in the reference section. The list of citations is arranged alphabetically by first author's last name and numbered sequentially. In the main text, sources are referred to by number in square brackets. (This is known as IEEE format.)

In the report, please include at least the sections shown in the sample report below, namely: **Introduction**, **Applications**, **Conclusions**, and **References**.

Unlike other assignments, this homework needs to be submitted on Blackboard electronically. All submitted reports will be checked for plagiarism using Turnitin.com. Do not submit a report with unattributed quotations or unattributed summarizations of others' work.

ACKNOWLEDGMENT
With permission, material for this assignment, including the writing excerpt in the example report, was borrowed and adapted from Prof. Fred Martin's web page, http://www.cs.uml.edu/ecg/index.php/HowToCite. Please see that URL for additional guidance.

DANG NHLT NGO

Grading Rubrics for this assignment

Feature	Value	Comments
Summary of algorithm	20	Clear presentation of how the selected algorithm works, e.g., • writing is clear and correct (20 pts); • writing is mostly clear (16 pts); • writing has some errors (12 pts); • writing is not coherent (0 pt)
Discussion of lit references	50	Clear presentation of how algorithm connects to real-world applications, at least 2 applications needed, e.g., • writing is clear and correct (50 pts); • writing is mostly clear (40 pts); • writing has some errors (30 pts); • writing is not coherent (10 pt)
Literature references	20	References has full & correct citations, e.g., • three citations (20 pts) • two citations (13 pts) • one citation (7 pts); • no citation (0 pts)
All sections of paper exist	10	Contains sections: Introduction, Applications, Conclusions, and References, as required
Total	**100**	

Your Report Title

Your name

INTRODUCTION

This section provides background information for your algorithm. For example, you will describe the detailed information about the algorithm, including how it works, the pseudo code, the analysis of the running time. If there is data structure involved, please also introduce the data structure and its use. Use diagrams or figures to help your explanation if needed. You may also discuss the general use of the algorithm. This is the section to summarize what you have learned in the class.

APPLICATIONS

This section introduces the real-world applications of the algorithm you selected. *Discuss at least two applications.* Explain why the algorithm was chosen to solve a certain problem and the benefits of using it. You may also discuss the performances under various situations and the limitation if there is any.

While you write, please cite the papers you read appropriately. All facts, algorithms, tables, or figures taken from the paper should be listed at the end of the document in a section titled "REFERENCE."

Here is an example of a passage that properly cites others' ideas:

> There is much evidence that there is a need for greater high-tech skills in today's workforce (e.g., [4]). There is substantial under-representation by women and ethnic minorities in technical fields, including computer science [2]. This is a matter of social justice and international competitiveness [3]. Addressing this, since 1999 NSF has spearheaded a series of funding programs to "broaden participation in computing" and other STEM fields [1]. Most recently, the White House announced Computer Science For All, which strives to "empower a generation of American students with the computer science skills they need to thrive in a digital economy" [5].

Notice that each statement/idea is attributed to the respective author(s). Sometimes their point is summarized, and other times it's presented verbatim with quotes. But in both cases, the paragraph builds its argument based on the work of others. Then, in the bibliography, the full reference for each cited work is given (please refer to the REFERENCES section).

Diagrams

In papers submitted for publication, you cannot reproduce diagrams from others' work **unless you have explicit permission from the publication journal** or if you can pass the four-factor test for fair use. If you ask for permission, there is often a fee that must be paid to license such use.

In papers submitted in this class, it is allowable to include diagrams borrowed from others work, given the following is satisfied:

1. You must provide that a full citation to the source of the diagram in your References section
2. Include verbatim statement, next to the diagram, ***"From <insert author(s) name and publication year>. Reproduced without permission."***

Alternatively, you may redraw diagrams you found elsewhere provided that you give credit. In this case (for a redrawn image), you would (1) provide the full citation in your References section, and then (2) next to the drawing, include the verbatim statement ***"Adapted from <insert author(s) name and publication year>."***

CONCLUSION

This section summarizes the algorithm and your findings in the report. You may discuss future work if you are interested in exploring the research area further.

REFERENCES

[1] Aspray, W. (2016). Opening Computing Careers to Underrepresented Groups. In *Participation in Computing* (pp. 13-52). Springer International Publishing.

[2] Jackson, D. L., Starobin, S. S., & Laanan, F. S. (2013). The shared experiences: Facilitating successful transfer of women and underrepresented minorities in STEM fields. *New Directions for Higher Education*, 2013(162), 69-76.

[3] Leggon, C., McNeely, C. L., & Yoon, J. (2015). Advancing Women in Science: Policies for Progress. In *Advancing Women in Science* (pp. 307-340). Springer International Publishing.

[4] Olson, S., & Riordan, D. G. (2012). Engage to Excel: Producing One Million Additional College Graduates with Degrees in Science, Technology, Engineering, and Mathematics. Report to the President. *Executive Office of the President.*

[5] Smith, M. (2016). Computer Science For All. https://obamawhitehouse.archives.gov/blog/2016/01/30/computer-science-all. Accessed December 9, 2018.

Algorithms -- COMP.4040 Honor Statement
(Courtesy of Prof. Tom Costello and Karen Daniels with modifications)

Must be attached to each submission, otherwise, your homework will not be graded.

Academic achievement is ordinarily evaluated on the basis of work that a student produces independently. Infringement of this Code of Honor entails penalties ranging from reprimand to suspension, dismissal or expulsion from the University.

Your name on any exercise is regarded as assurance and certification that what you are submitting for that exercise is the result of your own thoughts and study. Where collaboration is authorized, you should state very clearly which parts of any assignment were performed with collaboration and name your collaborators.

In writing examinations and quizzes, you are expected and required to respond entirely on the basis of your own memory and capacity, without any assistance whatsoever except such as what is specifically authorized by the instructor.

I certify that the work submitted with this assignment is mine and was generated in a manner consistent with this document, the course academic policy on the course website on Blackboard, and the UMass Lowell academic code.

Date: 04/22/2019

Name (please print): DANGNHI NGO

Signature: 

QUICKSORT ALGORITHM AND PRACTICAL APPLICATIONS

DangNhi Ngoc Ngo

I. INTRODUCTION

In this paper, we have discussed the Quicksort Algorithm, its complexity, implementation, and applications. It can be said that Quicksort is one of the greatest sorting algorithms of choice because of its fast speed and efficiency as of now. It was first introduced in 1961 by Hoare and has been widely utilized in a wide variety of applications to enhance the performance.

Quicksort algorithm works as follows: for sorting an array, it selects an element called *arbitrary pivot*, then it rearranges the array by moving to the left side all elements with a key value that are smaller than or equal to the pivot, and the array's right side consists of elements that are greater than or equal to the pivot – this is called *partitioning* [3].

Therefore, generally, Quicksort applies the divide-and-conquer paradigm including three steps as below:

Divide: the data is partitioned into two subarrays, elements on the left side are less than or equal to the middle element, and all elements moved to the right side are greater than or equal to the middle element. [3]

Conquer: call Quicksort for each sub-partition

Combine: no work is needed.

Quicksort Algorithm Pseudo Code: The Quicksort procedure is described in this algorithm below: [3]

1: **procedure** QUICKSORT (A[l,…, r]) [3]

```
2:      if r > l then                                   [3]
3:          pivot ← choosePivot (A[l,…, r])             [3]
4:          cut ← partition (A[l,…, r], pivot)          [3]
5:          Quicksort (A[l,…, cut-1])                   [3]
6:          Quicksort (A[cut,…, r])                     [3]
7:      end if                                          [3]
8: end procedure                                        [3]
```

From Edelkamp and Weiß 2019. Reproduced without permission.

Quicksort running time analysis:

Quicksort's running time is $\Theta(n\log n)$ and irrespective of the inputs, because its algorithm is not adaptive, which is similar to Mergesort and Heapsort [1].

Best-case: The recurrence $T(n) = \Theta(n\log n)$ when the partitions are balanced – their sizes are equal or within one of each other.

Worst-case: The recurrence $T(n) = \Theta(n^2)$ when the array has most unbalanced partitions possible, or one sub-partition is empty and another one comprises $(n - 1)$ elements.

Average-case: The recurrence $T(n) = \Theta(n\log n)$. It is a bit close to the best-case scenario.

One of the biggest advantages of the Quicksort algorithm is that it runs fasts and does not require the additional memory (or called in-place processing), except logarithmic size's recursion stack [5]. However, the worst-case running time $\Theta(n^2)$ can be mentioned as the major disadvantage of this algorithm [5].

General use of Quicksort Algorithm:

In general, Quicksort Algorithm is the greatest choice in most practical situations to avoid the costs in the usage of references.

Quicksort is often applied in commercial applications because of its fast running time and no need for extra memory space. Furthermore, it is also a selection for various applications that do not need to guarantee the response time, such as in medical field or aircraft and spacecraft controller, which are closely related to life-threatening and dangerous missions.

II. APPLICATIONS

1. Memory Performance Enhancement

Sorting algorithms play an essential role in contributing fundamentals in commercial and scientific applications because of its sensation to computer memory and data types [5]. Therefore, one of the most widely used applications of Quicksort Algorithm is enhancing memory performance. A study in 2000 by Xiao, et al. stated that two Quicksort algorithms called *memory-tuned quicksort* that is modified from the base quicksort and *multi-quicksort* that divides the data into segments which are smaller than the cache capacity are applied for cache optimization. Besides, *flash quicksort* and *inplaced flash quicksort* that are also presented in this research are able to implement impressively on both balanced and unbalanced data sets, which brings positive impacts to cache usage.

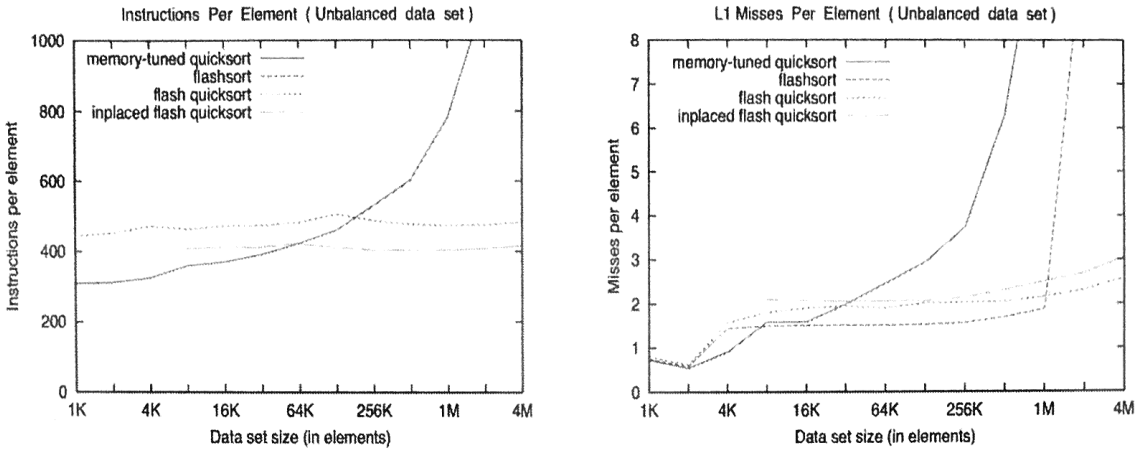

Figure 2.1

From Xiao, Zhang, and Kubricht 2000. Reproduced without permission.

Figure 2.1 is the simulation comparisons of the instruction counts and the L1 misses of the *memory-tuned quicksort*, the *flashsort*, the *flash quicksort,* and the *inplaced flash quicksort*. With flash quicksort performance, this result is explained that the sorting algorithm is a combination of flashsort and quicksort's advantages. The values that are maximum and minimum are first stored in the data set to identify the data range. Three steps called classification, permutation, and straight insertion are progressed in the flash quicksort algorithm, which makes the number of elements in each sub-partition small to fit the cache capacity [5]. However, the temporal locality issue that has not been solved completely in the *flash quicksort* has the answer in the *inplaced flash quicksort*. This algorithm uses an additional array to hold the permuted elements, which contributes to the decrease in data elements' instruction count [5]. It has proven that these Quicksort algorithms perform effectively even on unbalanced data sets, which results in increasing the required memory space and enhancing computer cache and implementation [5].

2. *Robust Index Model Improvement and Energy Cost Reduction*

In addition, another common application of Quicksort algorithm can be mentioned here is improving the robust index model and reducing the energy cost of home energy local network based on the load shifting strategy [4]. In this application, the Quicksort algorithm is used to adjust the operating order or set the priority of different types of load schedules, which is able to improve the robust index of appliances. In order to prove this assumption, Ran, et al. carried out an experiment with the operation order of appliances and load shift strategy based on Quicksort algorithm. This simulation required not only the information of the household energy management system but also the wind power and PV generation data [4].

Here is the Quicksort algorithm used for the Order of Loads, where S[.] is an array of Robust indexes for household appliances:

1: **Set** the desired objective function

2: **Initialize** the importance and deadline of appliances

3: **Quick Sort** (int S[], int i = first, j = last) {

```
    If (i < j) {
        Int split = part (S, i, j)
        Quicksort (S, i, split -1)
        Quicksort (S, split + 1, j)
    }
}
```

4: **Return** results

From Ran and Leng 2019. Reproduced without permission.

Based on the calculation results, Ran and Leng concluded that the robustness of the load schedule and energy consumption model has been maximized, and the energy cost of the entire household has significantly decreased with the usage of the Quicksort algorithm [4].

3. Graphics Processors Sorting Implementation

Another efficient Quicksort algorithm called GPU-Quicksort has been designed to use in parallel multi-core graphics processors' implementation. GPU-Quicksort is able to minimize the bookkeeping amount and the need of inter-thread synchronization by using high bandwidth of GPUs [2].

GPU-Quicksort works based on two major principles: keeping the inter-thread synchronization low by utilizing a two-pass design and minimizing memory accesses by constraining threads or coalescing read operations [2].

Algorithm 1 GPU-Quicksort (CPU Part)

procedure GPUQSORT($size, d, \hat{d}$) ▷ d contains the sequence to be sorted.
  $startpivot \leftarrow$ median($d_0, d_{size/2}, d_{size}$) ▷ d_x is the value at index x.
  $work \leftarrow \{(d_{0 \to size}, startpivot)\}$ ▷ $d_{x \to y}$ is the sequence from index x to y.
  $done \leftarrow \emptyset$
  while $work \neq \emptyset \wedge |work| + |done| < maxseq$ **do** ▷ Divide into $maxseq$ subsequences.
    $blocksize \leftarrow \sum_{(seq,pivot) \in work} \frac{||seq||}{maxseq}$ ▷ $||seq||$ is the length of sequence seq.
    $blocks \leftarrow \emptyset$
    for all $(seq_{start \to end}, pivot) \in work$ **do** ▷ Divide sequences into blocks.
      $blockcount \leftarrow \lceil \frac{||seq||}{blocksize} \rceil$ ▷ Number of blocks to create for this sequence.
      $parent \leftarrow (seq, seq, blockcount)$ ▷ Shared variables for all blocks of this sequence.

      for $i \leftarrow 0, i < blockcount - 1, i \leftarrow i + 1$ **do**
        $bstart \leftarrow start + blocksize \cdot i$
        $blocks \leftarrow blocks \cup \{(seq_{bstart \to bstart + blocksize}, pivot, parent)\}$
      $blocks \leftarrow blocks \cup \{(seq_{start + blocksize \cdot (blockcount-1) \to end}, pivot, parent)\}$

    $news \leftarrow$ gqsort$\ll |blocks| \gg (blocks, d, \hat{d})$ ▷ Start $|blocks|$ thread blocks.
    $work \leftarrow \emptyset$
    for all $(seq, pivot) \in news$ **do** ▷ If the new sequences are too long; partition them further.
      if $||seq|| < size/maxseq$ **then**
        $done \leftarrow done \cup \{(seq, pivot)\}$
      else
        $work \leftarrow work \cup \{(seq, pivot)\}$
  $done \leftarrow done \cup work$ ▷ Merge the sets done and work.
  lqsort$\ll |done| \gg (done, d, \hat{d})$ ▷ Do final sort.

Figure 2.3: GPU-Quicksort Algorithm in CPU Part

From Cederman and Tsigas 2009. Reproduced without permission.

The reason why we should use Quicksort is that even though the worst-case running time is $\Theta(n^2)$, in practice, when using a random pivot, the complexity is close to $\Theta(n\log n)$. Especially, in the experiments of GPU-Quicksort, it has shown the best performance, compared to radix sort and hybrid approach. Quicksort is faster because it is easy to select better pivot points and does not require an extra check to determine if the sequence is completely sorted. Consequently, the parallel Quicksort algorithm – GPU-Quicksort is the best method of choice for sorting large quantities of data on graphics processors [2].

III. CONCLUSIONS

The purpose of this paper is to present the Quicksort Algorithm that is taking the lead in sorting algorithms because of its efficiency and high practicality.

Quicksort is, in general, faster than most of the sorting algorithms because it is relatively cache-efficient. Compared to other sorting methods that require the extra space, Quicksort does not need additional storage memory and exhibits good cache locality. Moreover, the worst-case scenario of complexity $\Theta(n^2)$ can be solved by Randomized Quicksort with a high prospect of choosing the right pivot, which results in enhancing the algorithm performance.

Last but not least, Quicksort has been utilized in an extensive range of applications, for instance, memory performance improvements, the robustness of home energy local network and graphics processor implementation. It is irrefutable that Quicksort algorithm has significantly contributed to commercial applications with large data sets and brought far-reaching impacts to practical purposes.

IV. REFERENCES

[1] Brodal, G. S., Fagerberg, R., & Moruz, G. (2008). On the adaptiveness of Quicksort. *Journal of Experimental Algorithmics (JEA)*, vol. 12, pp. 3.2.1-3.2.20.

[2] Cederman, D., & Tsigas, P. (2009). GPU-Quicksort: A practical Quicksort algorithm for graphics processors. *Journal of Experimental Algorithmics (JEA)*, vol. 14.

[3] Edelkamp, S., & Weiß, A. (2019). BlockQuicksort: Avoiding Branch Mispredictions in Quicksort. *Journal of Experimental Algorithmics (JEA)*, vol. 24, no. 1, pp. 1-22.

[4] Ran, L., & Leng, S. (2019). Enhanced Robust Index Model for Load Scheduling of a Home Energy Local Network with a Load Shifting Strategy. *IEEE Access*, vol. 7, pp. 19943-19953.

[5] Xiao, L., Zhang, X., & Kubricht, S. A. (2000). Improving memory performance of sorting algorithms. *Journal of Experimental Algorithmics (JEA)*, vol. 5, no. 3.

Grading Rubrics for this assignment Dang Nhi Ngoc Ngo

Feature	Value	Comments
Summary of algorithm	20	Clear presentation of how the selected algorithm works, e.g., • writing is clear and correct (20 pts); • writing is mostly clear (16 pts); • writing has some errors (12 pts); • writing is not coherent (0 pt)
Discussion of lit references	50	Clear presentation of how algorithm connects to real-world applications, at least 2 applications needed, e.g., • writing is clear and correct (50 pts); • writing is mostly clear (40 pts); • writing has some errors (30 pts); • writing is not coherent (10 pt)
Literature references	20	References has full & correct citations, e.g., • three citations (20 pts) • two citations (13 pts) • one citation (7 pts); • no citation (0 pts)
All sections of paper exist	10	Contains sections: Introduction, Applications, Conclusions, and References, as required 10
Total	**100**	16+40+20+10 = 86/100