SAN FRANCISCO STATE UNIVERSITY
Computer Science Department

CSC510 – Analysis of Algorithms
Algorithm Challenge 2: Divide and Conquer

Instructor: Jose Ortiz

Full Name: Edel Jhon Cenario
Student ID: 921121224

Assignment Instructions. Must read!

Note: Failure to follow the following instructions in detail will impact your grade negatively.

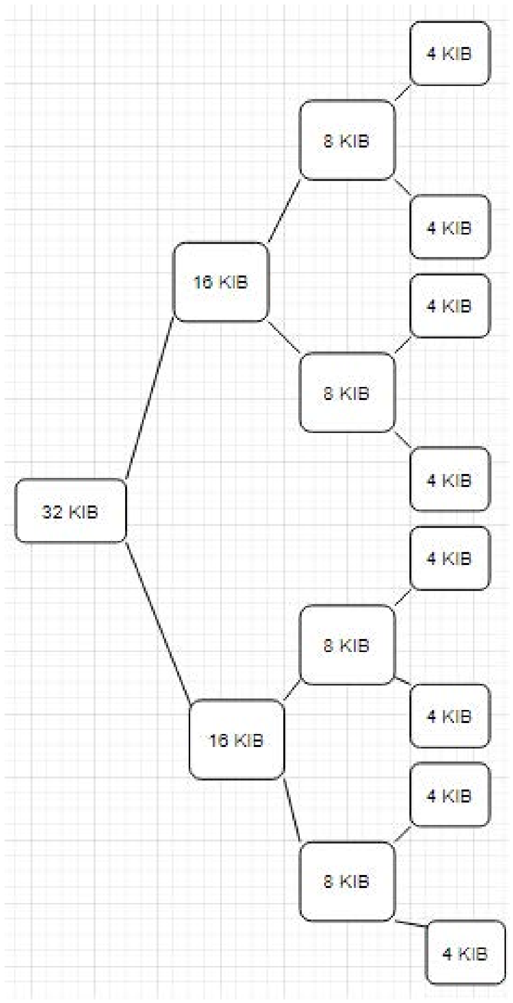
1. This algorithm challenge is worth 9%, and will be graded using a grading point scale where the maximum possible grade is 9 points
2. Handwriting work or screenshots of your work are not allowed. In addition, all the pseudocode done in this algorithm challenge must be done using LaTeX. Students who fail to meet this policy won't get credit for their work. Note that for pseudocode, I only want to see the compiled PDF psudocode, instead of the code to create the pseudocode.
3. Each section of this algorithm challenge is worth 2.25 points
4. Take into account that in this type of assignments, I am more interested in all the different approaches you take to solve the problem rather than on the final solution.

Problem Statement

1. Sort (in ascending order) the items in a file of size 2^x KIB using limited memory. Note that x is a unsigned integer where $x > 0$.
   (a) Rules:
      i. The file is located in disk (not in memory)
      ii. Memory is limited to 2 input buffers and 1 output buffer (4KIB each) Total memory capacity 12 KIB
      iii. Assume that the contents of the file are unsigned integers separated by a comma delimiter. (i.e 3,1,3,100,99...)
      iv. The unsigned integers are not sorted
      v. The file can contain duplicated integers
      vi. When in a file, a digit from an integer is 1 byte (datatype is char). When in a buffer, an integer is 4 bytes (data type is integer).
      vii. All the buffers in memory support ± 4 *bytes* of additional memory allocation.
      viii. You must use a Divide and Conquer algorithm to solve this problem
      ix. Temporary files, in disk, can only hold a max size of $((\#pass + 1) * 4)KIB$
   (b) Input and Output
      i. Input: a file containing unsorted unsigned integers in the range of 0 and 100 (both inclusive). For example: 100,67,99,99,1,1,3,24,88,96,37,10,10,88,100,99,99
      ii. Output: A file containing the sorted integers from the input file. For example: 1,1,3,10,10,24,37,67,88,88,96,99,99,99,99,100,100

Your work starts here

1. Describe the algorithm to solve the problem for a given file of size 2^5 first, and then describe the algorithm for a given file of size 2^x (any given x). Note that x is a unsigned integer where $x > 0$. You can use tables, diagrams, paragraph description..... to describe the algorithm. Be as clear as possible, and define clearly each step taken during the process. **Credit for this problem will be only given to those students that clearly define a step by step approach to solve this problem.**



The Divide and Conquer algorithm allows us to process our file by dividing, conquering, and finally combining. With the DAC algorithm, we can take our file and divide it into smaller temporary files until we reach the point that they can be processed directly.

We will take our 32KIB file, split it into two 16KIB files, split it into four 8KIB files, split it into eight 4KIB files.

Once we reach eight 4KIB files, we will now process it and merge it back together using the merge sort method. With this process, we are essentially sorting them as we merge them back to its original size.

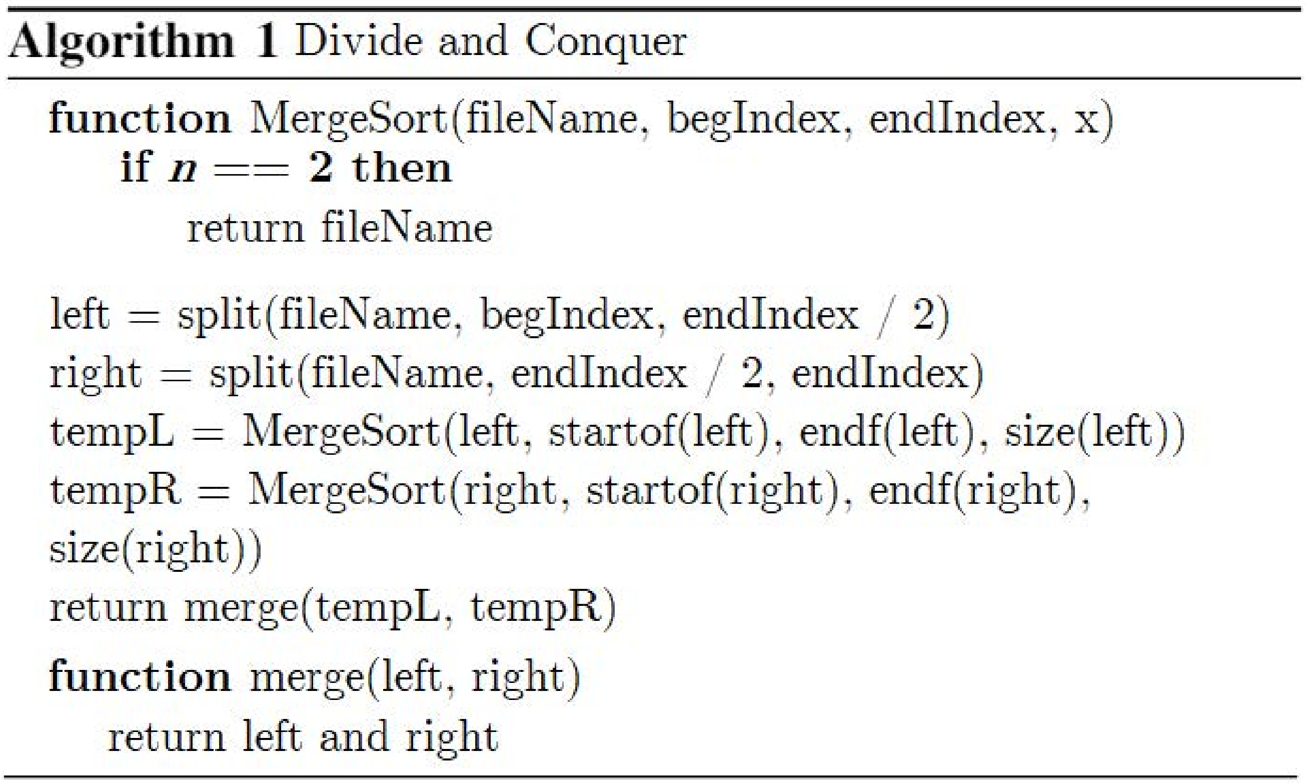
The division process is calculated by first finding the middle, we will then keep splitting it evenly until we reach the maximum amount of passes. As previously explained from the earlier paragraphs, we kept splitting our 32KIB file into its middle, until we reached 4KIB files. The reason why we stopped at 4KIB is because we have reached the point that we are able to process them directly. We then merge them back together by following the amount of passes we have.

To calculate our passes, we take the log2N +1 of the ceiling. We calculate N by the size of our file/size of our buffer. In our case, N = 32/4 = 8KIB.

This means that our passes = 3 + 1 = 4.

To put this into perspective:
Pass 0: 8x 4KIB files
Pass 1: 4x 8KIB files
Pass 2: 2x 16KIB files
Pass 3: 1x SORTED 32KIB file.

2. Write the pseudocode that represents your algorithm from problem (1). **Note that in this problem, I am asking for the compiled LaTeX pseudocode (PDF format) instead of the LaTeX code that creates this pseudocode**

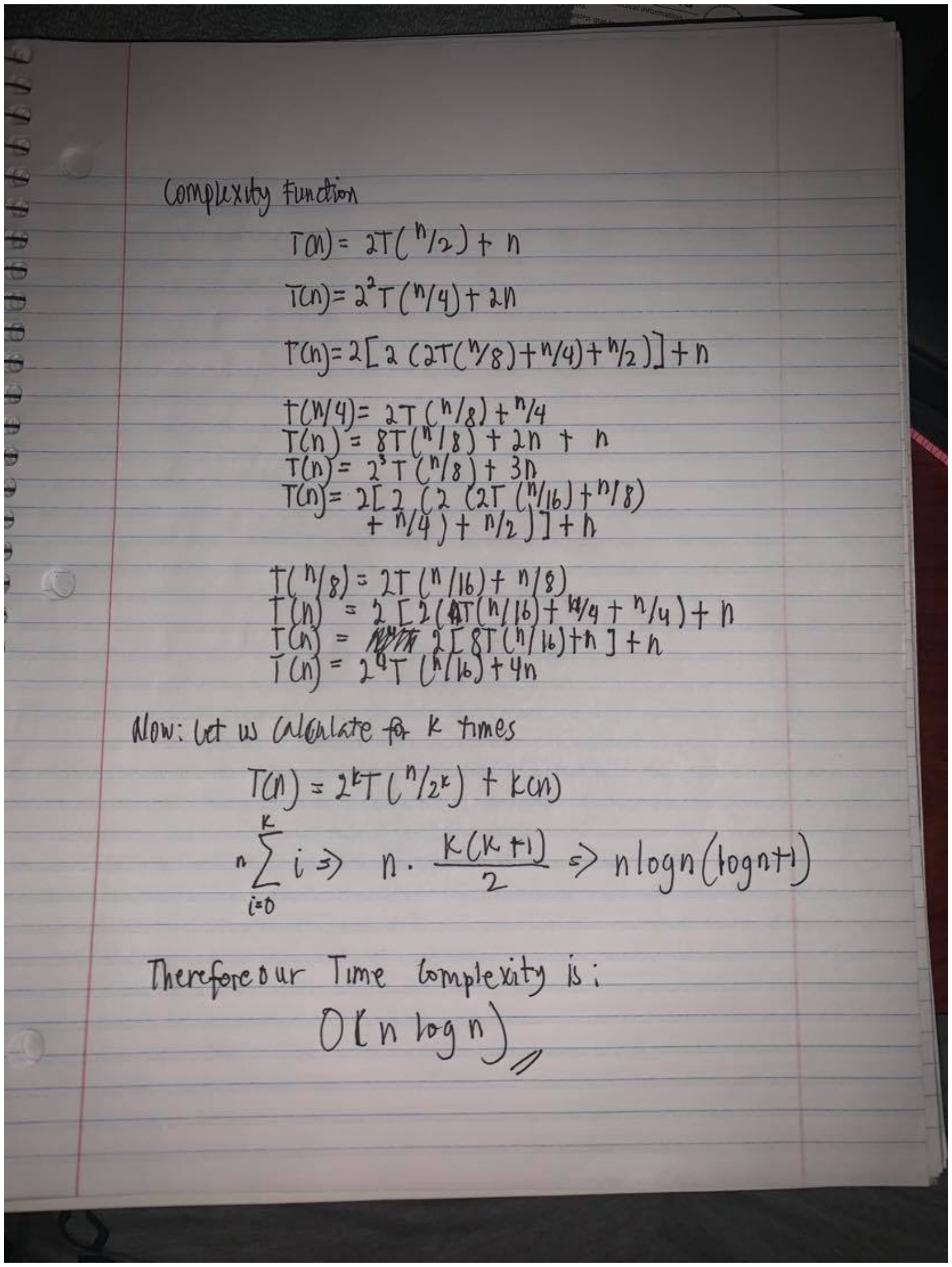


Algorithm 1 Divide and Conquer

```
function MergeSort(fileName, begIndex, endIndex, x)
    if $n == 2$ then
        return fileName

left = split(fileName, begIndex, endIndex / 2)
right = split(fileName, endIndex / 2, endIndex)
tempL = MergeSort(left, startof(left), endf(left), size(left))
tempR = MergeSort(right, startof(right), endf(right),
size(right))
return merge(tempL, tempR)
function merge(left, right)
    return left and right
```

3. Compute the complexity function, the I/O cost of operations, and the $\Theta(g(n))$ time complexity of your algorithm using your pseudocode from problem (2). For the complexity function and time complexity use the **substitution method** and then check your result with the **Master Theorem**. For the I/O cost define it as a function of N. Where N is the number of blocks (4KIB) in pass 0 of this algorithm



Complexity Function

$T(n) = 2T(n/2) + n$

$T(n) = 2^2T(n/4) + 2n$

$T(n) = 2[2(2T(n/8) + n/4) + n/2)] + n$

$T(n/4) = 2T(n/8) + n/4$

$T(n) = 8T(n/8) + 2n + n$

$T(n) = 2^3T(n/8) + 3n$

$T(n) = 2[2(2(2T(n/16) + n/8) + n/4) + n/2)] + n$

$T(n/8) = 2T(n/16) + n/8)$

$T(n) = 2[2(4T(n/16) + n/4 + n/4) + n$

$T(n) = 2[8T(n/16) + n] + n$

$T(n) = 2^4T(n/16) + 4n$

Now: Let us calculate for k times

$T(n) = 2^kT(n/2^k) + k(n)$

$$n\sum_{i=0}^{k} i \Rightarrow n \cdot \frac{k(k+1)}{2} \Rightarrow n\log n(\log n+1)$$

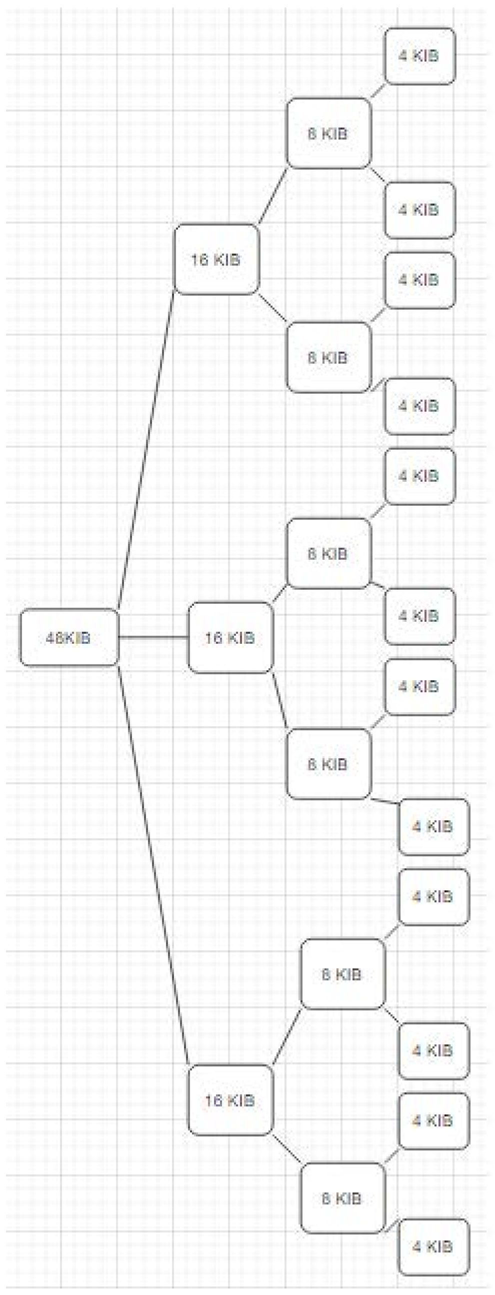
Therefore our Time complexity is:

$O(n \log n)$ //

4. Modify your algorithm from problem (1) to support files of any size (not only 2^x). For instance in problem (1), we were assuming that the file size is in the form of 2^x such as 2KIB, 4KIB, 16KB..... However, your new modification should include files of any size such as 3KIB, 37KIB.....

   In addition, do you think that the time complexity of this algorithm has been optimized as a result of the modifications applied? Why?

Using the same algorithm described in #3, this diagram is to show how my optimized algorithm: This time, I will be using a file size 48KIB


48KIB
16 KIB
16 KIB
16 KIB
8 KIB
8 KIB
8 KIB
8 KIB
8 KIB
8 KIB
4 KIB
4 KIB
4 KIB
4 KIB
4 KIB
4 KIB
4 KIB
4 KIB
4 KIB
4 KIB
4 KIB
4 KIB