CS 4306: Algorithm Analysis – Spring 2026

Department of Computer Science
Kennesaw State University

Assignment 2 – Chapter 3 – 100 Points

Note 1: If you re-upload revised files to D2L, you must re-upload **ALL** files as the system keeps only the most recent uploaded submission. No zip files, just individual Python files.

Note 2: In addition to the *Algorithm Design Block* (see below), please include comments for the code implementation of the algorithm. A *design block sample* file is posted with this assignment.

Note 3: Do not start with code then design. Submissions that revise code to be design receive no points. Pseudocode is language independent. See *Algorithm Design Block* below.

The goal of this assignment is to reinforce working with brute force and exhaustive search approaches for algorithm design and development, using textbook pseudocode syntax.

Problem #1 (50 points): Given string of text (**ignore case**), we are asked to identify and count all substrings that start with letter A and end with letter B using the brute force approach we discussed in chapter 3. For example, string CABAAXBYA contains 4 substrings. Can you see them?

Using textbook pseudocode syntax and following sample design block in D2L, **design** a brute-force algorithm of your own for this problem and then determine its Big-O efficiency. That is, design your algorithm; conduct performance analysis for its basic operation (comparisons) using table format, just like we did in class; and include the complete analysis in the design block, not just stating the Big-O notation (no correct/complete analysis in table format, no points). Type the performance analysis in word file and upload it with your submission. It must be complete and correct.

Next, **implement** your algorithm to verify its correctness. Name the program *Substrings.py* (saved in file *Substrings.py*). Make it an interactive program by integrating the following menu. The program must display the menu first, then allow the user to interact with the code. Do Not hard-code test data.

```
-----------------MAIN MENU--------------
1. Read input string
2. Run algorithm and display outputs
3. Exit program

Enter option number:
```

The program outputs are as follows. Test your code thoroughly with these and other sample inputs. Make sure your outputs are formatted to look exactly as follows (blank line before and after output).

```
Input string:           ABRAB
# of substrings:        3
Listing of substrings:  AB,  ABRAB,  AB
# of comparisons:       75 (this is a hypothetical value)
```

```
Input string:            CABAAXBYA
# of substrings:         4
Listing of substrings:   AB, ABAAXB, AAXB, AXB
# of comparisons:        41 (this is a hypothetical value)
```

Always re-display the menu after each option (other than option 3) is fully exercised with blank lines before and after the menu.

Note: Do not start with code then design. Submissions that revise code to be design block receive no points.

Problem #2 (50 points): Consider the partition problem: given a set of *n* positive integers, partition them into two disjoint subsets with the same sum of their elements. (Of course, the problem does not always have a solution, i.e., if the total sum of all integers is odd.) Using textbook pseudocode syntax and following sample design block in D2L, **design** a brute-force algorithm of your own for this problem and then determine its Big-O efficiency. That is, design your algorithm; conduct performance analysis for its basic operation (comparisons) using table format, just like we did in class; and include the complete analysis in the design block, not just stating the Big-O notation (no correct/complete analysis in table format, no points). Type the performance analysis in word file and upload it with your submission. It must be complete and correct.

Next, **implement** your algorithm to verify its correctness. Name the program *Partition.py* (saved in file *Partition.py*). Make it an interactive program by integrating the following menu. The program must display the menu first, then allow the user to interact with the code. Do Not hard code the test data.

```
-----------------MAIN MENU--------------
1. Read set size (number of integers)
2. Read set elements (integer values)
3. Run algorithm and display outputs
4. Exit program

Enter option number:
```

The program outputs are as follows. Test your code thoroughly with these and other sample inputs. Make sure your outputs are formatted to look exactly as follows (blank line before and after output). **Note** that your code may generate different disjoint subset.

```
Set size:                3 integers
Integer values:          2  1  3
Disjoint subsets with same sum: {1,2}
                                {3}

Set size:                7 integers
Integer values:          1  2  3  4  5  6  7
Disjoint subsets with same sum: {2,3,4,5}
                                {1,6,7}

Set size:                8 integers
Integer values:          1  2  3  4  5  6  7  8
Disjoint subsets with same sum: {1,2,3,4,8}
                                {5,6,7}

Set size:                5 integers
Integer values:          1  2  3  4  5
Disjoint subsets with same sum: No disjoint subsets with the same sum
                                of their elements found
```

Always re-display the menu after each option (other than option 4) is fully exercised with blank lines before and after the menu.

Note: Do not start with code then design. Submissions that revise code to be design block receive no points.

Algorithm Design Block:

The *algorithm design block* is a block of comments inserted into the code file right before the code, and after the author header (see posted *design block sample* file). The design block shows your detailed design of the algorithm at hand. This includes explanation of your design approach (logical steps section) and the pseudocode of your final algorithm (**following the textbook pseudocode syntax, see pages 4, 7, 18, and 23**). The pseudocode is language-independent, it should not use language-specific syntax (operator, functions, structures, reserved words, etc.). Again, this block of text comes after the author's header block and before the code as illustrated in the posted sample file.

Submission:

Do not forget to include the author header in each submitted file, as shown below, before the *Algorithm Design Block*. Do not forget to document your code as stated in **note 2** above. No zip files. No late submission. No author header, no points. No algorithm design block, no points. No proper code documentation, no points.

```
// Name:              <your name>
// Class:             CS 4306/03
// Term:              Spring 2026
// Instructor:        Dr. Abdur Rahman
// Assignment:        2
// IDE Name:          <your IDE name>

//***************************
Algorithm Design Block

Algorithm title: ...

Logical steps:
...

Pseudocode syntax:
...

Big-O Analysis:
...

***************************//

// Code Section
// Working code goes here
```

Upload your .py files, named as indicated above, to the assignment submission folder in D2L by the due date posted in D2L. Make sure that your code is running correctly right before you upload your files. Test your code with different sample input data to make sure the code handles special cases and give correct results that verify your algorithm correctness.

Grading Rubric:

1. Algorithm analysis documentation	
2. Design block	
3. Author header, documentation, organization	
4. Code output	
5. Other issues	