Assignment Instructions Please read the following instructions carefully. Failure to follow the instructions will results in losing at least 5 points.

- This is a group assignment. You may choose to form your own group of **at most two students**. If you do not build your own group, you will be assigned a group randomly. Groups may be different from assignment 2 groups.
  Here are the general rules for teams:

  1. You should form the pair on Canvas by **Thursday November 7 at 5 pm** using one of the groups under *People* → *assignment4-groups*.
  2. You should work on the whole assignment together.
  3. You should **not** just partition the work between you.
  4. You should upload only one submission to canvas.
  5. You may pair up with any student across the two sections.

- Hand in all program header/source files softcopy using Canvas and be sure that they properly execute. Failure to do so will mean that your program is not graded.

- All header/source files should be compressed into a *ZIP* archive using the following naming format: `firstname1_lastname1_firstname2_lastname2_asmt2.zip`.

- Your ZIP file should include: (1) folder `Part1` (2) and folder `Part3`.

- Your answer for **parts 2** should be submitted as a single PDF file **through Canvas**. Do **not include it in the ZIP** file.

- Please include your full names in the PDF.

- Please refer to the university integrity policy on the syllabus, and remember that **you should be able to explain and reproduce any code you write and submit as part of this assignment.**

Assignment Goals and Objectives

1. Programming methodology:
   - Understand the difference between various sorting algorithms.
2. C++ implementation:
   - Implement recursive functions.
   - Implement sorting algorithms.

Part 1: The Jacobsthal Function – 25 points + 25 EC points

Use the starter code provided in the folder `Part1`.

The Jacobsthal sequence is very similar to the Fibonacci sequence in that it is defined by its two previous terms. The difference is that the second term is multiplied by two.

$$J(n) = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ J(n-1) + 2 \times J(n-2) & \text{if } n > 1 \end{cases}$$

Write a recursive function to compute the n-th Jacobsthal number. Since this sequence grows exponentially, you will quickly reach the limit of what a regular int variable can represent. To support larger values, your return type should be a `long long` (typically a 64-bit integer allowing you to represent values up to 9,223,372,036,854,775,807).

Extra Credit

As discussed in class, this recursive function is very inefficient in that it compute the same value multiple times. To remove the redundancy, we use the technique called memoization which stores previously computed values in order not to compute them multiple times. In general, this means:

- We use recursion while maintaining a table of values
- At every recursive call of the function, we check if the value has already been computed:
  1. If it has been computed, we use the value and return it (and make no further recursive calls)
  2. If it has not, we make the recursive call, but instead of immediately returning the computed value, we make sure to store it in the table.

For this to work, you will need to set up a table (e.g. map) and fill it up with base case values. Modify your code to implement memoization technique.

Part 2: Sorting – 45 points + 20 EC points
Choose an array of 8 random integers then sort it using three of the following algorithms. Show the contents of the array and program variables at each iteration of the algorithm while sorting the array into ascending order.

- Choose **two** of the Basic Algorithms:
  1. Selection Sort
  2. Bubble Sort
  3. Insertion Sort
- Choose **one** of these two faster algorithms:
  1. Merge Sort
  2. Quick Sort
- **Extra credit** – one of the algorithms we have not covered in class:
  1. Heap sort
  2. Shell sort

You may type your answer (preferred) or write it clearly and attach all scans/photos in one PDF.

Part 3: Recursive Sorting – 30 points + 25 EC points
Using the starter code provided in the folder `Part3` and the pseudo code discussed in class, **implement merge sort** in the `LinkedBag` data structure we have been using for the `Instagram340` application.

You are also provided with a code generated by ChatGPT. `mergeSortChatGPT.cpp` includes an implementation of merge sort for a linked list. The code is fully generated by ChatGPT and is not necessarily suited for `LinkedBag`. You may use it as a model to write your own implementation for a `LinkedBag`.

For this part, you should add a prototype for merge sort in `LinkedBag.h`, then add the implementation to `LinkedBag.cpp`.

You do not need to make any changes to `linkedBagSortingMain.cpp`.

You do not have to incorporate it with the Instagram340 application.

Extra Credit
For extra credit, implement **quick sort** in `LinkedBag`. `quickSortChatGPT.cpp` is provided for this part.