Assignment Instructions – please read carefully.

1. This is a group assignment. **No individual submissions will be accepted**.
2. You may **choose to form your own group** of exactly two students. If you do not build your own group by the deadline, you will be assigned **a group randomly**.
3. Groups may be different from assignments 2 and 3. If you choose to work in the same pair, make sure you indicate that.

   Here are the general rules for teams:

   - You should form the pair on Canvas by **Wednesday November 12 at 4 pm** using one of the groups under *People* → *assignment4-groups*.
   - **Build your team even if it is the same as in assignment 3**. Groups are not carried over.
   - You should work on the whole assignment together.
   - You should **not** just partition the work between you.
   - You should upload only one submission to Canvas.
4. Written Parts:
   - Please type your answers for **Part 1 questions 2 and 3, Part 2 and Part 3 question 3** and hand them in as a **single PDF** file through Canvas.
   - You are encouraged to use a diagramming platform or program to draw the call tree.
   - Please include your full names on the first page of the PDF.
   - **DO NOT include the PDF file in the ZIP file**; please submit it separately on Canvas.
5. Coding Parts:
   - Hand in all program source files as soft copies using Canvas
   - Be sure that they properly execute. Failure to do so will mean that your program is not graded.
     Note: assignments are graded using the terminal and `C++11` standard.
   - All source files should be compressed into a *ZIP* archive.
   - Use the following naming format:
     `<firstNameLastname1>_<firstNameLastname2>_asmt<#>.zip`.
     Example: `SaraElAlaoui_JaneDoe_asmt4.zip`
   - You must use the code provided on Canvas.
   - Your ZIP file should include two folders with their respective files: (1) the folder `Part1`, and (2) the folder `Part3`.

6. **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: Recursive Function – 50 points
Use the starter code provided in the folder `Part1`. Do not modify the `main` function.

Consider the following sequence, where each term is defined by its three previous terms. The second term is multiplied by two, and the third term is multiplied by four.

$$J(n) = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ 1 & \text{if } n = 2 \\ J(n-1) + 2 \times J(n-2) + 4 \times J(n-3) & \text{if } n > 2 \end{cases}$$

1. (25 points) Write a recursive function to compute the n-th number of the sequence. Since this sequence grows exponentially, you will quickly reach the limit of what a regular `int` variable can store. To support larger values, your return type should be a `long long` (typically a 64-bit integer allowing you to store values up to 9,223,372,036,854,775,807).

2. (10 points) Manually draw the call tree for the recursive function for n = 6 and include it in your PDF.

3. As discussed in class, a recursive function can be very inefficient with a large call stack. This function is also inefficient in that it computes the same value multiple times. To remove this redundancy, we use the technique called **memoization** (this is not a typo :)).

   (a) (5 points) Define *memoization* in your own words. Please cite your sources.

   (b) (10 points) In bullet points, briefly describe how you would implement memoization in your program.

Part 2: Sorting – 20 points + 5 EC points
Choose a non-trivial array of 8 random integers then sort it using two of the following algorithms (three if you do the 5-point extra credit). Show the contents of the array and program variables at each iteration of the algorithm while sorting the array into ascending order.

- (10 points) Choose **one** of the Basic Algorithms:
  1. Selection Sort
  2. Bubble Sort
  3. Insertion Sort
- (10 points) Choose **one** of these two faster algorithms:
  1. Merge Sort
  2. Quick Sort

- **Extra credit** – (5 points) 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 – 50 points + 10 EC points
Using the starter code provided in the folder `Part3` and the pseudocode discussed in class, **implement merge sort** in the `LinkedBag` data structure, which we have been using for the `EventTicket340` application.

1. (10 points) Use GenAI to generate an implementation of merge sort for a **linked list**, not for `LinkedBag`. Save this code into a file `mergeSortGenAI.cpp` and include it in the submission ZIP file. See `quickSortGenAI.cpp` for an example.

2. (25 points) Using the code generated by GenAI as a starter code, write your own implementation of merge sort in the `LinkedBag` data structure.
   For this part:
   - you should modify/add the 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 `EventTicket340` application.

3. Briefly answer the following questions:
   (a) (10 points) How did you check GenAI's code for correctness and hallucination?
   (b) (5 points) What changes did you make to the code to adapt it to the `LinkedBag` data structure?

Extra Credit (10 points)
For extra credit, implement **quick sort** in `LinkedBag`. `quickSortGenAI.cpp` is provided for this part.