Goals of the project

The overarching goal of this project is to enable you to identify and plot the growth rate of several different algorithms. In particular,

1. Implement the recursive algorithm to compute the Fibonacci sequence and determine the growth rate of the recursive algorithm. Thereafter, use the consecutive elements of the Fibonacci sequence to find the growth rate of *Euclid's* algorithm in the worst-case.
2. Implement algorithms related to the problem of exponentiation based on 3 design techniques and compare their growth rates: i) Decrease-by-one, ii) Decrease-by-constant-factor, and iii) Divide-and-Conquer.
3. Implement two sorting algorithms: i) Selection Sort, and ii) Insertion Sort, and compute their growth rate in the best-case, average-case, and worst-case.

You will write a short report summarizing the outcome of the above tasks in the form of scatter-plots.

Important note: When computing the total number of basic operations in any of the algorithms, do **not** explicitly use any formula (that may have been derived in the class) but simply place a counter corresponding to the basic operation and increment it at the correct place in the code.

Task 1: Fibonacci Sequence and GCD

Task 1a: Implement the **recursive algorithm** fib(k) to compute the kth Fibonacci Number and compute the corresponding number of additions A(k) employed by the recursive algorithm.

- Produce a *scatterplot* of A(k) and indicate the algorithm's asymptotic complexity. Note that A(**k**) is a function of **k**; your program must choose the different values of **k** to generate the scatterplot.

Task 1b: Compute the **worst-case efficiency** of *Euclid's algorithm* to compute GCD. Note that the worst-case inputs for Euclid's algorithm happen to be the consecutive elements of the Fibonacci sequence (you may store these from Task 1a). So, compute GCD(**m**, **n**) where **m** = fib(k+1) and **n** = fib(k) for several values of k >= 1.

- Produce a *scatterplot* showing the number of modulo divisions D(n) taken to compute GCD as a function of **n** and indicate the algorithm's asymptotic complexity. Note that D(**n**) is a function of **n**; your program must choose the different values of **n** (obtained by choosing different values of k) to generate the scatterplot.

Task 2: Exponentiation

The problem of **exponentiation** computes a^n, where **a** is a constant. Implement the following algorithms for solving the problem of exponentiation:

Decrease-by-One:

$if\ n = 0: \quad exp(a, n) = 1$

$if\ n > 0: \quad exp(a, n) = a \bullet exp(a, n - 1)$

Decrease-by-Constant-Factor:

$if\ n = 0: \quad exp(a, n) = 1$

$if\ n\ is\ even: \quad exp(a, n) = exp(a, \frac{n}{2})^2$

$if\ n\ is\ odd: \quad exp(a, n) = a \bullet exp(a, \frac{n-1}{2})^2$

Divide-and-Conquer:

$if\ n = 0: \quad exp(a, n) = 1$

$if\ n\ is\ even: \quad exp(a, n) = exp(a, \frac{n}{2}) \bullet exp(a, \frac{n}{2})$

$if\ n\ is\ odd: \quad exp(a, n) = a \bullet exp(a, \frac{n-1}{2}) \bullet exp(a, \frac{n-1}{2})$

- Produce a *scatterplot* showing the number of multiplications M(n) made by each of the algorithms and indicate their asymptotic complexity. Note that M(**n**) is a function of **n**; your program must choose the different values of **n** to generate the scatterplot. You can choose any value of **a**.

Task 3: Sorting

Implement and analyze the complexity of **Selection Sort** and **Insertion Sort**

Check out this awesome animation comparing different sorting algorithms

Run your code on test data that is sorted, random, and reverse sorted, which corresponds to the best case, average case, and worst case, respectively.

- Produce a *scatterplot* showing the number of comparisons C(**n**) made by each of the algorithms and indicate their asymptotic complexity. Note that **n** here is the size of the list (that is being sorted).
    - Test data is provided for the **range of n** between 100 - 10000 at an increment of 100. A folder called "smallSet" is included to test early iterations of your program on small values of **n**. However, your scatterplot should be shown for the data in folder "testSet".
    - When plotting the number of comparisons, you may use another increment factor of your choice for speed purposes but it must span the entire range. For example, you may increment n by 500 (instead of 100).
- Keep in mind that a distinct scatterplot needs to be computed for each of the best-case, average-case, and worst-case situations.

Formatting your output in two modes: User testing mode and Scatter plot mode.

Your program should give choice for the user to enter two modes:

1. **User testing mode**: This mode prompts the user for values of certain parameters and returns an output based on the goals of each task:

    - **Task 1:** User is prompted for the value of **k**; the program outputs the value of fib(k) and GCD(m, n) where **m** = fib(k+1) and **n** = fib(k).
    - **Task 2:** User is prompted for the value of **a** and **n**; the program outputs the value a^n using each of the three algorithms.
    - **Task 3:** User is prompted for the size of the list **n** (between 10 - 100 at an increment of 10), the program loads the appropriate file data{n}.txt from the folder "*smallSet*" in the test data and outputs the sorted output based on the two algorithms on terminal.
2. **Scatter plot mode**: This mode does **NOT** prompt the user for any values. It generates a scatter plot for each task, as described in the description of each of the tasks:
    - **Task 1:** Fib(k) for different values of k generated using the recursive algorithm may be stored for this purpose. OR, for sake of speed, you may use an iterative algorithm to compute the Fibonacci sequence once. Use consecutive elements as input to the GCD(m, n) function for computing and plotting D(n). (Think about what should be the upper bound of **k** and clearly indicate it in your report)?
    - **Task 2:** You may choose any value of the constant **a**. Compute and plot M(n) for the three different algorithms in the same plot (it will help you to compare them!).
    - **Task 3:** For different sizes of the list **n** (range 10 - 10000), read data from the folder "test*Set*" in the test data that is sorted (data{n}_sorted.txt), random (data{n}.txt), and reverse sorted (data{n}_rSorted.txt). Use that **data** to compute C(**n**) for each of the two sorting algorithms. Produce three scatter plots that compare the complexity of the two algorithms in the Best-case, Average-case, and Worst-case.

Formatting your report

Write a short report summarizing the outcomes of each of the three tasks.

- Clearly label the x-axis and y-axis of each of the scatter plots, and a legend, wherever applicable (i.e. when plotting multiple graphs in the same figure).

- Determine the most likely asymptotic complexity from each scatter plot in Θ notation.
- Comment on your understanding of the differences you observe between the different algorithms in each task. In addition, provide any additional information that may be necessary to understand the plots. For example, as you were plotting the growth rates A(k), D(n), M(n), C(n), how did you choose the largest value of k? How did your implementation fare for large values of n?

Extra Credit

Extra credit (+5) will be provided if you get feedback regarding your report from any group and incorporate them into your report.

- Please indicate in your README file the group to whom you gave feedback and the group from which you received feedback.
- Include the feedback that you received, the original report and modified report (that incorporated the feedback) to get full credit

---

Deliverables (Put these in a single tar / zip file)

1. Code relevant to each task.
2. A README file that has the following:
    - Names of members in your group.
    - Instructions to run code (I will follow these exactly to run your code - you will lose points if instructions are missing/insufficient).
    - If attempting Extra Credit, the group to which feedback was given and the group from which feedback was obtained.
3. Report summarizing the outcomes of the three tasks.
4. Feedback files (if you are attempting Extra Credit) comprising of
    - Original report
    - The feedback that you got
    - Modified report