# CS2223: Algorithms | Assignment 1

## *General Instructions*

• Code vs. Pseudocode: Each question will state whether the deliverable is pseudocode or an executable program and code that the TA will run to give you a grade.

• You should make clear in your report:
– How to compile your program
– How to execute it, along with any arguments

• Submissions: Submit your work as a .Zip file through Canvas. All programs plus your written questions (which should be PDF files to ensure we can see them) should be zipped into a single archive when submitted. Please organize folders by question, e.g. Q1, Q2, etc.

# Q1: Stacks and Stackulators (20 pts)

In this question, you are to implement a version of the Stack calculator on p. 129 of Sedgewick, with several modifications.

**Tasks:**

1) Implement the Stack on p. 129 of Sedgewick. Make your Stack Calculator it's own Class that you will call from your main program.
2) In your main program, call functions from your Stack Calculator class to demonstrate the following functionality:
    a) Show at least three expressions that demonstrate it working, e.g. "( 4 + 1 + 3 )"
    b) Show at least one expression where it does not work. Make it believable-- not just random characters.
3) Modify the Stack Calculator to add at least two new mathematical operations of your choosing.
    a) In your main program, show at least one expression per new operation to validate their functionality.

While working on this problem, be sure to inspect the **ops** and **vals** stacks to gain an understanding of how Stacks work. This will be important in future algorithms.

**Output:**
The output of this should include Java code TAs can inspect and run.

**Challenge:**
For an additional challenge, build more sophisticated calculations and parsing.

# Q2: Managing Management (30 pts)

In this question, assume the role of a software engineer working on search functions, who needs to prove their worth to management.

Management is questioning the amount of time you're spending on search functions, which, according to them, "already work fine".
But you know how to convince them: reports and charts.

Your plan is to demonstrate, empirically, the difference between the old search functions and the new algorithms you've been working on.

**Tasks:**

1) Implement at least three search algorithms: Linear, Binary, and some other variant of your choosing (this could be Binary recursive, or others).
    a) Write your own code, but referring to your book or online sources is permitted.
2) Prepare for data collection: add "step" counters, like we did in the second class, at key operations in the search functions.
    a) Placement is key! Be sure you can justify where you place the step counters.
    b) Points can be lost for overcounting (inflating by an order of magnitude) or undercounting (missing an order of magnitude).
    c) Ensure you are making a fair comparison between your algorithms.
3) Run a series of experiments and collect step counts:
    a) Run single search keys with multiple sizes of input data e.g. 100, 1,000, 10,000, 100,000, 250,000
    b) Run the above at least 3 times for each algorithm, to give you a measure of how different inputs contribute to variance in algorithm performance.
4) Generate plots for each algorithm, with the y-axis showing steps and the x-axis showing input size.

Hints: Your assigned readings thus far includes discussion of key operations and how to identify them. This question puts that into practice, while giving you additional exposure to search algorithms and their variants. You will also gain an understanding of how time-complexity can be measured empirically, through experiments and plotting.

**Output**

Provide a concise report (PDF) describing the three algorithms, and plots demonstrating their relative effectiveness. Provide the code as well. Place these in an organized folder for evaluation.

# Q3: Orders of Growth, Empirically (10 pts)

To observe the importance of algorithm efficiency and the order of growth, perform the following experiment.

**Tasks:**

Assume an input of size 100,000 (denoted as N)

> **(a) Linear function (O(N))**: Write a function that loops over the input items (a single loop) and print the elapsed time taken by the function in milliseconds.
>
> **(b) Quadratic function (O(N2))**: Write a function that loops over the input items with two-levels of nesting and print the elapsed time taken by the function in milliseconds. The function should have two nested loops (outer and inner) where each one starts from 0 to N.
>
> **(c) Cubic function (O(N3)):** Write a function that loops over the input items with three-levels of nesting and print the elapsed time taken by the function in milliseconds. The function should have three nested loops where each one starts from 0 to N.

Hint: The actual code inside the loop is not important. So make it simple and define a variable in the beginning of your program, and then increment this variable inside the inner most loop (The pseudocode below is for two-level nesting).

```
int sum = 0;
for (i=0;  i<=n ; i++)
   for(j=0; j<=n; j++)
       sum += j;
   End Loop;
End Loop;
```

**Output:**

This should include a simple Java program with all three functions. In addition, include a single PDF showing a plot of the three algorithms' execution time. You can make the x-axis the type of function (e.g. "linear"), and the y-axis can be the observed execution time. A bar chart may work well, but other clear forms of visual representation are permitted. Use a log scale if your bars are too long to fit reasonably in a chart.

Extra Hint: Refer to the Stopwatch example in the book for code on collecting time measurements.

# Q4: Orders of Growth, Mathematically (10 pts)

**Tasks:**

Put these functions in order. Put the following functions in a list by increasing order of growth. If they have the same general order of complexity, then either one can come first, but mark them!

| | | | |
|---|---|---|---|
| $n \log_2 n$ | $12\sqrt{n}$ | $1/n$ | $n^{\log_2 n}$ |
| $100n^2 + 6n$ | $n^{0.51}$ | $n^2 - 324$ | $50n^{0.5}$ |
| $2n^3$ | $3^n$ | $2^{32}n$ | $\log_2 n$ |

**Output:**
A PDF listing these in order. Mark visually, in some way, those that are of the same order of complexity.

# Q5: Sorting Comparison (30 pts)

Management is at it again, this time pestering you about the time you're taking on developing new sorting algorithms.

Repeat the analysis in Question 2, with the following changes.

**Tasks:**

1) Implement **Bubblesort**, **Mergesort**, and **Quicksort**, as separate classes.
2) You will not need search keys. Instead, simply generate unsorted data of varying sizes, e.g. 100, 1,000, 10,000, 100,000, 250,000
    a) Run at least 3 repetitions at each size to account for variance.

**Output:**
Provide a concise report (PDF) describing the three algorithms, and plots demonstrating their relative effectiveness. Provide the code as well. Place these in an organized folder for evaluation.