Introduction

Week 1

Required Activities

- Check Announcements regularly (every 2-3 days)
- Review Syllabus
- Read “Data Structures And Algorithms.: Made Easy.” (DSA): Chapter 1
- Introduce yourself in Discussions forum
- Read Toold.pdf:
  - Install Compiler/IDE of your choice (Java, Python, C#, or C++)
  - Install and/or setup tool for screen recording (to make assignment videos)

Evaluation Overview

- Participation (14%)
  - Attending or watching the weekly session is worth 1% per week
  - Recording must be watched by Sunday midnight EST of the same calendar week as live session to earn credit

- Bi-weekly Assignments (64%)
  - Each student must do his/her own work
  - Programming can be completed in Java, Python, C#, or C++
  - Need to submit source code (as zip file) and short video explaining implementation and showing running program (another zip file) and any additional file needed for a specific assignment
  - Always check instructions what to submit and grading rubric before submitting work

- Project (22%)
  - Design, implement, and test a system that uses data structures (linked list, queue, stack, tree, or graph) and algorithms learned in the class (at minimum sort and search). You will implement two different algorithms (e.g. search) and compare them
  - Can be done individually or as a group up to 3 members
  - Need to submit source code, analysis report, and individual video explaining system, code, and analysis
  - Check grading rubric before submitting work

Highlights

- Weeks 1-5 concentrate on basic data structures and algorithms
- Weeks 6-14 concentrate on problem solving techniques, algorithm design, and analyzing algorithms
- Weekly meeting is not mandatory but you must watch the recording (participation credit)
- Questions about the topics can be posted in the discussions forum and sent as email (I will usually respond within 48 hours)
- You are responsible for reviewing all weekly materials and completing programming assignments (and final project) on time. Check Syllabus for details
- It is your responsibility to make sure that submitted files are readable and virus-free and have all the required files. Double check after submitting to make sure.

Assignment submission & grading

Submission

- Each solution needs to be a separate class in a separate file
- Programs need to be submitted as source code and not pasted into word document
- Programs needs to be submitted as single zip/rar file with source code only. Do not submit your whole IDE project
- Data should be populated by the program (hardcoded in main method) and not interactive unless assignment instructs otherwise
- Video should be submitted as a separate zip/rar file. Keep the compressed video about 50MB max.

Grading

- You will not earn any points for the programs unless there is a video explaining them. Video is not optional.
- Assignments are graded on correctness of the solution and explanation in the video – if you are unable to explain the solution, I assume you did not do the work.

Terminology

- **data –** a value or a set of values (e.g. number 5)
- **data structure** – collection of data items stored in memory with some operations to manipulate that data (e.g. array)
- **data type** – classification of data which determines what operations can be performed on that data (e.g. integer, List)
  - **primitive data types** – predefined by the programming language (e.g. int, long, double)
  - **built in classes** – complex data types that the programming language provides in a library (e.g. Java String)
  - **user defined data types** – programmer creates a data type such as enum, struct, or class (e.g. Account)
  - **abstract data type (ADT)** – definition of a logical data type with specific operations (e.g. Queue, Stack)
- **algorithm** – finite set of instructions (list of steps) that accomplishes some task (solves a problem)

Analyzing algorithms

- **time complexity** – measures (or estimates) the running time of an algorithm, counting the number of elementary operations performed by the algorithm, based on the size of input data. So expressed as a function of the size of the input. Most interested in value as the input size increases. Uses big 'O' notation.
    - **worst-case time complexity -** maximum amount of time for inputs of a given size
    - **average-case time complexity** - average of the time taken for inputs of a given size

- **space complexity** – how much space the algorithm needs to execute. Most interested in most space needed at any given point (worst case) relative to size of input data. Uses big 'O' notation

- **selecting input considerations –** random, sorted, partially sorted, size of input

- **Asymptotic analysis**:
    - **best case** – input where algorithm is quickest (shortest time with least amount of work)
    - **worst case** – input where algorithm is slowest.
    - **average case** – input where performance is average. Input is random.

Big Oh notation

- **O(1)** - constant time – where time does not depend on the size of the input (the upper bound is independent of input size)
- **O(n)** – linear time – as input size increases the time increases linearly
- **O(log n)** – logarithmic time - the ratio of the number of operations relative to the size of the input decreases and tends to zero when n increases so gets close to constant time; considered highly efficient
- **O(n^2)** – quadratic time
- **O(n^3)** – cubic time
- **O(2^n)** - exponential
- …

Calculate time complexity

- **loops –** at most equals to running time of statements times number of iterations

```
repeat n times:
      i++  // constant c
```

time =c * n = O(n)

- **nested loops** – product of the sizes of all the loops

```
repeat n times:
      repeat n times:
              i++  // constant c
```

time = c * n * n = $O(n^2)$

- **If-else statement –** test condition plus the largest of the true or false branches

```
if (func() == 1) // constant
      i++
else  // constant * n
      repeat n times:
           i++  // constant c
```

time = c0 + c1 * n = O(n)

time complexity cont.

- **consecutive statements** – add time complexity of each statement

```
i++  // constant
m = m * 2  // constant
```

time = c0 + c1 = O(1)

```
i++  // constant
repeat n times:  // nested loop c * n * n
        repeat n times:
              i++  // constant c
```

time = c0 + c1 * n * n = $O(n^2)$

Questions ?

- Post in the discussions
- Send email to RMcFadden@HarrisburgU.edu
- Respond usually within 48hours