**Module 1: ADTs and Recursion (August 16, 2022)**

Abstract Data Types

* **Data** are the ingredients to cook a menu / components/ properties.
  + Data information to state knowledge about a world
* **Operations** are the goods that allow you to make food/ process that we intend to perform against the data.
  + Operations manipulate data to reason about the world.
* **ADT** is a collection/container that includes data and operations under one entity. Entity that contains data and operation
  + It is composed of data components and properties of a typeof data, and the relationship among these data.

ADT Stages

1. **Specification**
   1. Putting ourselves in the shoes of the end user
   2. Identify the data and operations
   3. We do not know yet how to do this
      1. What are the data?
      2. What are the operations?
      3. From the point of view of the user.
2. **Representation** 
   1. How can the data be efficiently taken down by the computer
   2. How can the computer remember the data/store
      1. Implement how the data can be stored
   3. Identify appropriate data structure
   4. Identify the operation
      1. How is the data stored?
         1. Node (a container of related data)
         2. Array
      2. Decided by someone implementing the ADT
3. **Implementation** 
   1. How to do something
   2. Set iteration to 1
   3. We compare the grade of ana to the grade of the student being visited
   4. Identify the procedure on how to do the operation
      1. How are the operations performed
      2. Decided by someone implementing the ADT
      3. OOP languages implement ADTs naturally
         1. Fields (data)
         2. Methods (implementation/operation)

Sets of Data

* May be
  + **Static** (changes infrequently)
  + Dynamic (items are added and deleted over time)
* Factors for modeling data sets as ADTs:
  + Data relationships
  + Operations on data
  + Static or dynamic?
* Some ADTs
  + Stacks, queues, linked lists, trees, graphs, hash tables
* Parallel arrays, elements with the same index number are related.
* Organizing data so that you can efficiently extract the needed information.
* When you create your implementation, mag iiba depends on the person.
* Among the choices, which is the most efficient way.
* When we have identified the ADT, it is most simple to use OOP after.
* OOP are a very natural choice because it’s a good fit for ADT
  + Method is the implementation of the abstract operation.
  + Field
* Data can be static (rarely changes) or dynamic (changes a lot)
* Factors for modeling data sets as ADTs:
  + Data relationships
  + Operations on data
  + Static or dynamic?
* The right choice will dictate your operations

Data Structures

* Data type includes a range of values and operations.
  + Data structures are representations that show relationships among data in a computer environment.
    - bit/boolean
    - Bit string/byte/int/char
    - Strings of char/floating point numbers
    - Ordered tuple
    - Arrays, structs, classes/objects
    - Etc.
  + Uses primitive or composite data structures
  + Affect how operations are carried out
* Int x = means we are declaring a particular portion of our memory labeled by x
* Operations have methods that are valid
* Tuple - array that does not change
* The **correct choice of data structure** will enable you to have an easier and simpler data gathering procedure as it affects the operations.
* Primitive (by default nasa java na) or composite (combo such as user defined).

Algorithms

* Whatever technique we choose to follow, we must clarify the procedure.
  + Algorithm is a well defined procedure that processes input values to generate desired output values to solve a problem.
  + Algorithms implement operations using data structures
  + Abu Ja’Far Muhammed Ibn Musa Al-Khwarizmi
    - Wrote procedures to perform math operations
* Algorithms are methods in java
* Aspects of good algorithm
  + Has input
  + Has output
  + Definite
  + Effective
  + Finite
  + Correct
  + General
  + **Efficient (can be measured by space and time) faster, more efficient, less amount of memory being used is efficient.**
* Algorithms can be written using pseudocode.
  + Describe process precisely and concisely
    - Natural language (english)
    - Programming language (java)
    - Pseudocode (mix)
  + Combined to form more compex algorithms
* If you have designed a good algorithm, it can be used for further use
* Complex algorithms are a mixture of several simple and small problems. Then provide individual solutions for complex problems.
* Programs = Data Structures + Algorithms
* Procedure or Method: Algorithm module
  + Functions - with return value
  + Subroutines - has side effects
* We utilize what is available and address the present problem.
* No matter how simple or complex something is, it is composed of data structures and algo.
* Any program is composed of data structures and algorithms.

Aspects of Good Algorithm

* Has input
* Has output
* Definitive
* Effective
* Finite
* Correct
* General
* Efficient

Case Study - Stage 1

* Character Sequence ADT
* Stage 1: Specification
  + Data:
    - Specification of characters - the length is from 0 up to 255 characters.
  + Operations:
    - toString - function that returns the conversion of the sequence to string
      * Providing the implementation without
    - isFull - function that returns true if the character sequence is full
      * is means returning a boolean
    - append - subroutine that adds a character at the end of the sequence, unless it is already full.
* Stage 2: Representation
  + class CharSeq {

private static final int maxLength = 255;

Private char [ ] cArray = new char (maxLength + 1);

Private int cLength = 0;

* Stage 3: Implementation
  + Class CharSeq

Public String toString (){

String s = “”;

for (int i=1; i<cLength; i++)

S = s + cArray [i];

return s;

}

Public boolean isFull() {

if (cLength == maxLength) return true;

else return false;

}

public void append (char c) {

if (isFull()) System.out.println(“CharSeq is full.”);

else cArray [++cLength] = c;

}

public static void main(String[] argos){

CharSeq ca = new CharSeq ();

ca.append(‘d’);

ca.append(‘s’);

ca.append(‘a’);

System.out.println(ca.toString());

}

* Iteration vs Recursion
* Iteration
  + An execution of a loop’s body
  + Iterating a block of statements (looping once)
* Recursion
  + Solves problems in terms of smaller instances or parts
  + Through procedures - functions or subroutines
    - A procedure calling itself
    - A cycle of procedure calls
  + Similar
    - Mathematical induction
    - Recursive algorithms
    - Recurrence relations
    - Recursive definitions
    - Elements of Recursion
* Basis: direct solution to smallest parts
* Recursive step: solves a big problem in terms of solutions to smaller parts

Recursive Definition

* Set N of all natural numbers
* Factorial of n
* The nth term of Fibonacci sequence

Block contains code of algorithm

**Module 2: Computational Complexity (August 30, 2022 and September 6, 2022)**

Algorithm

* A step-by-step procedure to solve a problem
  + Start from an initial state and input
  + Proceed through a finite number of successive states
  + Stop when reaching a final state and producing output

Algorithm Analysis

1. Correctness and Generality Output (should produce accurate results)
2. Efficiency (must be measurable)
   1. Time efficiency
      1. Mabilis
      2. Main criteria but don't neglect space
   2. Space efficiency
      1. RAM
      2. Bandwidth
      3. Send the smallest amount of data as possible
   3. Comparing against
      1. A standard
      2. Another algorithm that solves the same problem

Algorithm Performance

* Algorithm performance is measured by the amount of computer memory and running time required to run an algorithm
* Performance Measurement
  + Empirical/Experimentational Analysis
    - * It changes because realtime ayos and your environment plays a role sa running time
    - Computes memory space in use and running time
    - Results are not general - valid only for tested inputs
    - Same machine environment must be used to compare two algorithms
  + Theoretical analysis
    - Compute **asymptotic bound** of space and running time
      * Asymptotic bound is the bases
    - Sometimes, it is difficult to measure average cases; **worst case analysis are often used** 
      * When you are processing all the elements in an array
      * We are preparing for the worst case analysis
    - Machine independent

Empirical Analysis

* Measurement Criteria
  + Actual memory space in use
  + Running time of algorithm (how much time you need to finish the algorithm)

Theoretical Analysis

* Measurement Criteria
  + Space - amount of memory that algorithm utilizes
  + Time
    - Running time
    - Typically measured by the number of primitive operations
    - However, different primitive operations may take different running time
* Basis
  + Uses a pseudocode of the algorithm
  + Needs to take all possible inputs into account
  + Evaluate the algorithm speed independent of the machine environment
* Measure the primitive annotations

Pseudocode

* High-level description of an algorithm
  + Independent of any programming language
  + More structured than English proses
  + Less detailed than program codes
  + Hiding program details issues
  + Easy to understand

Space Complexity

* Amount of necessary memory for an algorithm
  + The space complexity may define an upper bound on the data that the algorithm uses
* Reasons for space complexity analysis
  + We may not have a sufficient memory space in our computer
  + When we solve a large-scale problem, memory space is often a critical bottleneck

Time Complexity

* Amount of time required to run an algorithm
* Reasons for time complexity analysis
  + Some applications require real-time responses
  + If there are many solutions for a problem, we typically prefer the fastest one
* Measurement
  + Count a particular operation
  + Count the number of steps
  + Asymptotic complexity

Asymptotic Complexity

* Asymptotic
  + Approaching a given value as an expression containing a variable tends to infinity
* Asymptotic complexity
  + Describes the growth rate of the time or space complexity with respect to input size
  + Is not directly related to the exact running time
* Three notations
  + Big O notation provides an upper bound for the growth rate of the given function. (worst case)
  + Big Omega notation provides a lower bound. (best case)
  + Big Theta notation is used when an algorithm is bounded both above and below by the same kind of function. (average case)
  + To get the asymptotic eme disregard the constant
    - Only get the variable with the highest component

Caveat

* The relative behavior
* Our analysis is only estimates not the actual performance of eme !
* The asymptotic complexity may make no sense of small n.
  + Main concern is worst case or large N
  + Algorithms with the same asymptotic complexity still have different time complexity
  + Habang tumataas yung degree, bumabagal siya

Trade-Off

* Space complexity and time complexity may not be independent.
  + There is a trade-off between the two complexities.
    - Sparse matrix are separated

Sorts:

* Selection sort
  + Select one small number and swap with the current smallest
* Insertion sort
* Bubble sort
* Merge sor
* **Quick sort**
* **Merge sort**
* **Linear search**
* **Binary search**

**Given an array of scores, return true if each score is equal or greater than the one before. The array will be length 2 or more.**

***Given an array of scores sorted in increasing order, return true if the array contains 3 adjacent scores that differ from each other by at most 2, such as with {3, 4, 5} or {3, 5, 5}.***

**What is the specification of the ADT?**

***Given a string and a non-empty* word *string, return a version of the original String where all chars have been replaced by pluses ("+"), except for appearances of the word string which are preserved unchanged.***

**What is the data representation of your ADT?**

**Stack - September 20, 2022**

**Stack Operations**

* **New - creates a new empty stack**
* **Clear - clears the stack of its items**
* **isEmpty - boolean** 
  + - **Checks if the stack has no items**
* **isFull - boolean** 
  + - **Checks if the stack is filled to its capacity**
* **Push - adds an item as the new top item of the stack**
* **Pop - removes and returns the top item from the stack**
* **Peek - returns top items without removing it from stack. Also called the top () open**

**t**