Chapter 1a Pre-Practical Class Submission

1. Explain what are lists, stacks and queues.

Lists: Shopping list app

- List is a collection of items in which the items have a position **(No order constraint)**
- List is a linear collection of entries in which entries may be added, removed and searched for without restriction

Stacks:

- Stack is a collection of objects in which the objects are inserted and removed according to the last-in first out **(LIFO)**

Queue:

- Queue is a collection of objects that are inserted and removed according to the first in first out **(FIFO)** principle. That is, the element that has been in the queue the longest will be the next one to be removed. Elements enter a queue at the rear or back; elements are removed from the front

2. Consider that you are developing software for karaoke centers to host karaoke shows and playing karaoke songs. Besides displaying the latest new songs, the songs in the system are grouped by the gender of the singers, the name of the singer, language, region, musical genre, etc. The system can sort the songs according to the customer's selection. For example, when a customer is selecting the songs based on title, the titles are sorted ascendingly; when a customer is selecting the songs based on singer, the singers are sorted according to their surnames; and so on. The system allows multiple customers to add songs to their own playlists and then they have to wait for their turn. For every customer, it will allow each person to sing their chosen songs for around 10 minutes. After that, the system will select the next user in line to sing. This will repeat until all songs are played, or the time is up. Any customer can remove any of their selected songs from the playlist at any time, and they can move the song to the front of the playlist as well. The system is able to sync the first 10 songs in the playlist to the customers' smartphones in real time, so that they know when it is their turn even when they are outside the karaoke room.

   Based on the case study above, analyze how list, stack and queue can be applied in the system. Explain with relevant examples.

   a. Propose ONE (1) application of **list** in the karaoke system with an example.

   b. Propose ONE (1) application of **stack(LIFO)** in the karaoke system with an example.

c. Propose ONE (1) application of **queue(FIFO)** in the karaoke system with an example.

a. **List** Application Example
   - *Create a list of songs based on customers preference.*
   - For example, it generates a list of songs based on musical genres namely rock, hip hop music, jazz and electronic music.

b. **Stack** Application Example: **[LIFO]**
   - *The stack can be applied when the system wants to display the latest new songs added.*
   - For example, the system stores the songs based on the newest date and the latest date will be displayed at the top of the customer's playlist.

c. **Queue** Application Example: **[FIFO]**
   - *The queue can be applied to this system when syncing the song*.
   - For example, the system is able to sync the first 10 songs in the playlist to the customers' smartphones in real time, so that they know when it is their turn even when they are outside the karaoke room (*FIFO*).

Chapter 1b Pre-Practical Class Submission

1. Describe how a stack may be used to check whether the parentheses in the following mathematical expression are balanced and match.

   [(6 + {9 - 7}) / {2 * (5 + 3)}]

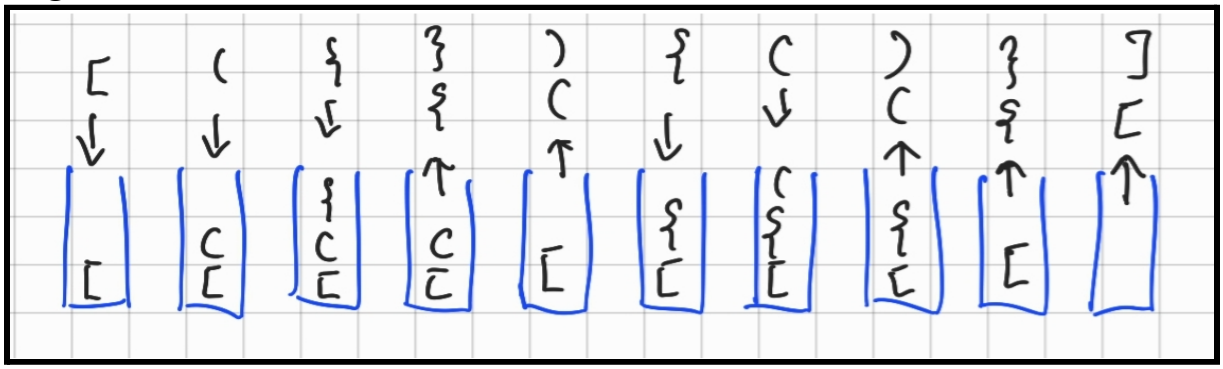
General Explanation:

- First, traverse the expression to process each character
  - ❖ Push it onto the stack if it is a left/open bracket
  - ❖ Pop a bracket from the stack and compare whether the two bracket types match

Explanation based on mathematical expression:

- The first parentheses scanned is [which is open bracket, so push it onto the stack
- The next parentheses is (which is open bracket, so push it onto the stack
- The next parentheses is { which is open bracket, so push it onto the stack
- The next parentheses is } which is close bracket, so pop it onto the stack and it matches with the {
- Next, the parentheses is) which is close bracket, so pop it onto the stack and it matches with the (
- The next parentheses is { which is open bracket, so push it onto the stack
- The next parentheses is (which is open bracket, so push it onto the stack
- Next, the parentheses is) which is close bracket, so pop it onto the stack and it matches with the (
- Next, the parentheses is } which is close bracket, so pop it onto the stack and it matches with the {
- Last, the parentheses is] which is close bracket, so pop it onto the stack and it matches with the [
- In conclusion, the contents of a stack during the scan of an expression that contains the balanced delimiters [({ }) { ( ) }]

Diagram:



2. Describe how a stack may be used to evaluate the following postfix expression:

6 2 + 5 3 * /

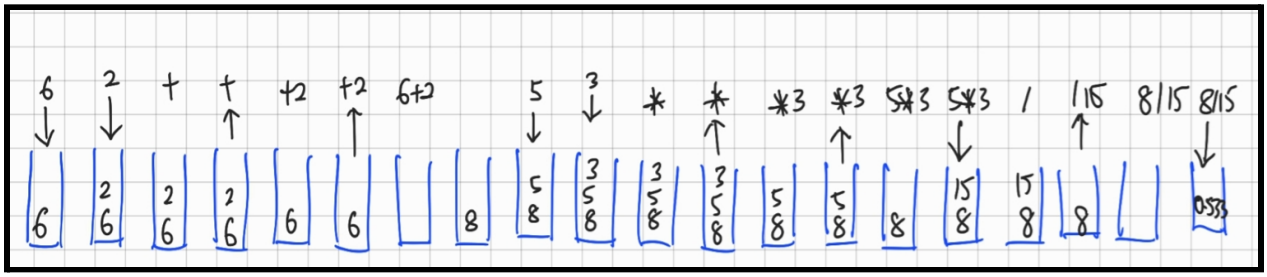
General Explanation:

- Postfix expression evaluation uses a stack of operands.
- It traverses the expression to process each character
  - ❖ If it is an operand, push it onto the stack
  - ❖ If it is an operator,
    - ➔ Pop the top two elements from the stack *(Note: the first pop execution returns the right operand while the second pop returns the left operand)*
    - ➔ Perform the operation on the two elements
    - ➔ Push the result of the operation onto the stack

Explanation based on postfix expression:

- The first character is "6" which is a operand, so push it onto the stack
- The second character is "2" which is also an operand, so push it onto the stack
- The third character is "+" which is an operator, so pop the two elements from the stack.
  - ➔ The first pop which is "2" returns the right operand
  - ➔ The second pop which is "6" returns the left operand
- Next, push the result "6+2" (8) i=onto the stack
- The fourth character is "5" which is an operand, so push it onto the stack
- The fifth character is "3" which is an operand, so also push it onto the stack
- While for the next (6th) character is "*" which is an operator, so pop the two elements from the stack.
  - ➔ The first pop which is 3 returns the right operand
  - ➔ The second pop which is 5 returns the left operand
- Next, push the result of 5*3 (15) onto the stack
- The next character is "/" which is an operator, so pop the two elements from the stack.
  - ➔ The first pop which is 15 returns the right operand
  - ➔ The second pop which is 8 returns the left operand
- All the characters are done scanning, the remaining element in the stack will be the results of this postfix evaluation which is 0.533

Diagram:



Chapter 1c Pre-Practical Class Submission

1. Describe the basic operations that are typically provided for a queue ADT?

Basic operations of Queue ADT

Operation	Description
enqueue	Add an entry to the rear of the queue
dequeue	Remove the entry at the front of the queue
isEmpty	Check if the queue is empty
getFront	Retrieve (but do not remove) the front object in the queue
size	Return the number of entries in the queue

2. Explain the use of the queue data structure in the computation of profits in the sale of shares.

- In the computation of profits in the sale of shares, we will need to sell them in which we purchase them by applying the queue data structure (FIFO).

- For example:
  - ➔ Last year, you bought 15 shares of coca cola at RM40 per share
  - ➔ Last month, you bought another 20 shares at RM70 per share
  - ➔ Today, you sold 30 shares at MRM 60 share
  - ➔ Profit of shares: **[(30*60) - [(15*40) + (15*70)]]** = 1800 - 1650 = **RM 150**

Chapter 2 Pre-Practical

1. What is an **ADT specification**?

   - It written in natural language, mostly in english, and are independent of any programming language
   - It was used as specifications for concrete data types (i.e. the actual data types used in programs)

2. Describe the following parts of an operation in an ADT specification:

   a. Operation header

   b. Operation description

   c. Pre-condition

   d. Post-condition

   e. Returns

   Are each of the above parts compulsory for every operation in the ADT specification? Explain your answer.

Answer :

a. Specify the return type (if any), operation name and parameters (if any)
b. A brief description of what the operation does
c. A statement which the condition(s) must be true before the operation invoked
d. A statement that specify what is true after invoke the operation
e. The value return by the operation (if any)

No, it's not compulsory to have all these parts for every operation in the ADT specification. First, it's not necessary to specify if an operation doesn't have any return type or parameter. Next, some operation does not necessarily have a precondition in order to invoke the operation, same goal as the "returns" part.

Chapter 3 Pre-Practical

1. What is the **Big-O notation**?

- Time complexity of algorithms - estimate the times for the best case, average case and worst case
- We express time efficiency as a factor of the problem size (e.g. when searching a collection of data, the problem size is the number of items in collection)

2. Describe the steps to derive the Big-O notation for an algorithm.

Steps to derive the Big O notation for an algorithm
– Use the given identities
– Ignore smaller terms in a growth-rate function

- E.g., if the growth-rate function is 4n^2 + 50n -10, O(4n^2 + 50n -10)
  = O(4n^2) by ignoring the smaller terms
  = O(n^2) by ignoring the constant multiplier / coefficient

Chapter 4 Pre-Practical Class

1. Describe the main steps needed to create and use an abstract data type (ADT)?

Step 1 Write the ADT specification

- Write an ADT specification which describes the characteristics of that data type and the set of operations for manipulating the data. Should not include any implementation or usage details.

Step 2 Implement the ADT

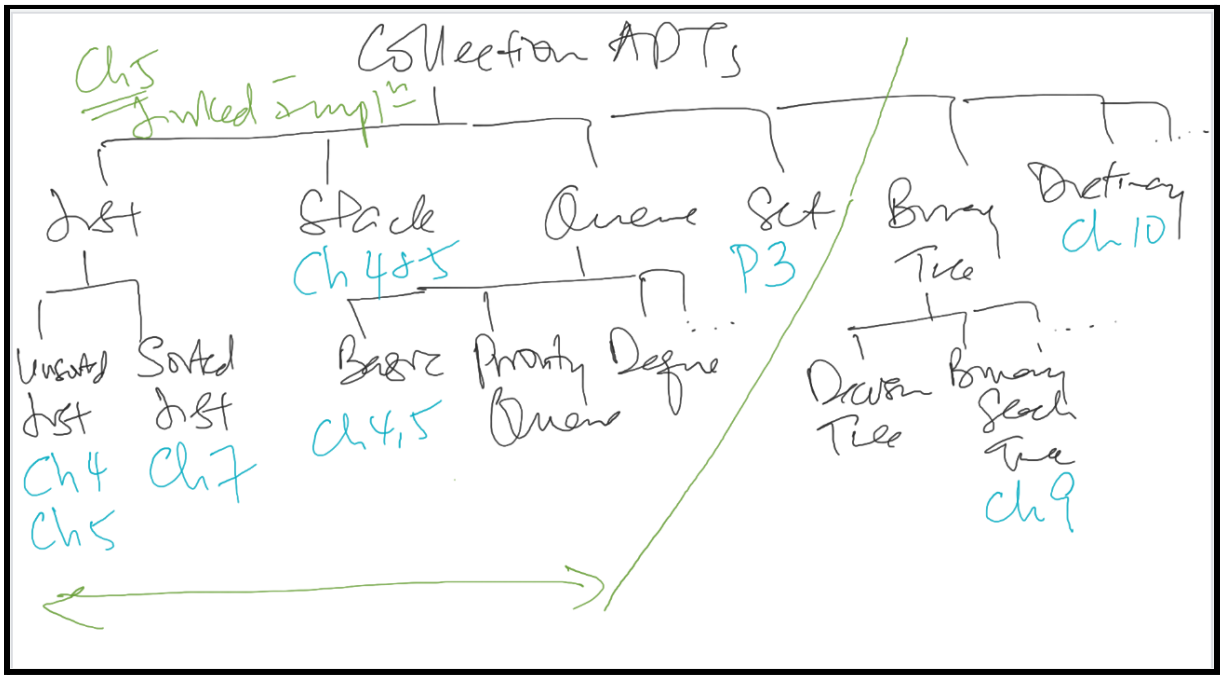
a. Write a Java interface
   - Include all the operations from the ADT specification

b. Write a Java class
   - This class implements the Java interface from a.
   - Determine how to represent the data
   - Implement all the operations from the interface

Step 3 Use the ADT in a client program or application

2. What is a collection ADT? Give FIVE (5) examples of collection ADTs.

- An ADT that can store a collection of objects
- A linear collection that stores its entries in a linear sequence, e.g.
  - List, Stack, Queue



Client program:

```
ListInterface<Item> itemList = new List<>();
customerList.add(___)
```

Interface class <T>
Boolean Add(T target)

3. State why each of the following are considered as bad practices.

   a. If you plan to implement the List using arrays, you should name the ADT as *ADT ArrayList*.
      - List

   b. If you plan to use the ADT list to store a list of customer objects, you should name the ADT as *ADT Customer List*.
      - **EntryList:** specific as Customer object cannot be reused in another context / application

   c. If you plan to use the ADT list to store a list of customer objects, your element type for the add operation's parameter should be Customer.
      - Should use **Generic type T**
      - Concept of reusability : abstraction
      - Client invoke method as: customer.add(target)

4. Explain what is a generic type. Describe how it is used in collection ADT implementations.

   - Used in Java Interface and classes which implement collection ADTs to specify and constrain the type of objects being stored in the collection.
   - ListInterface<T>, List<T>, T as Generic Type
   - Client program: ListInterface<Customer> customerList = new ListInterface<>();
   - The object stored is not generic but specific, for example a Customer object, again this ADT cannot be reused in another context
   - public interface ListInterface<T>
   - When you use the class, you specify an actual type argument to replace T. e.g.
   - ListInterface<Customer> customerList;

5. What is an iterator? How should it be implemented in the implementation class of a collection ADT?
   - Iterator is an object that enables you to traverse a collection of data, beginning with the first entry. It acts as a cursor or pointer, moving and locating individual elements
   - The iterator class is defined as an inner class of the collection ADT. Thus, it has direct access to the ADT's data fields

java.util.Iterator

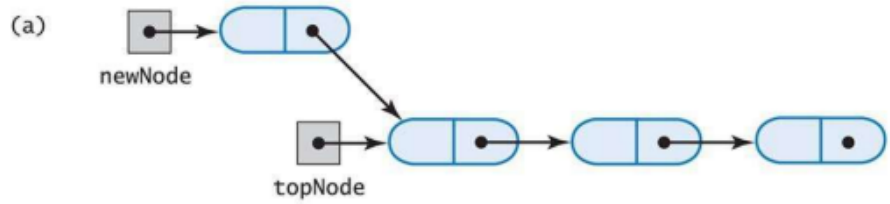
- This interface specifies a generic type for entries
- Includes 3 method headers:

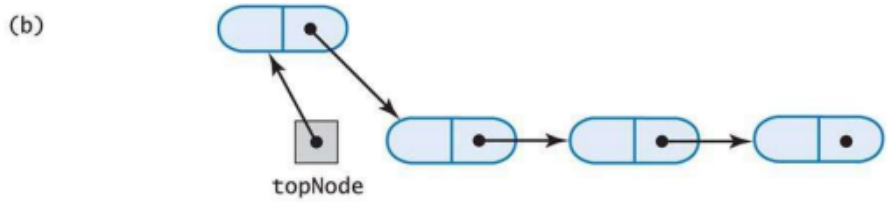
hasNext	Checks if next entry exists .
next	Returns next entry and advances iterator to the next entry. Throws `NoSuchElementException` if there are no more elements.
remove	Removes the entry that was returned by the last call to **`next()`.**

Chapter 5 Pre-Practical Class Submission

1. Explain the steps that need to be carried out in the push and pop operations when a stack is implemented using linked implementation with a reference to the top entry.

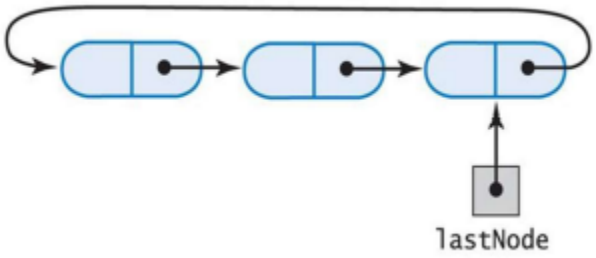
Maintain 1 reference topNode

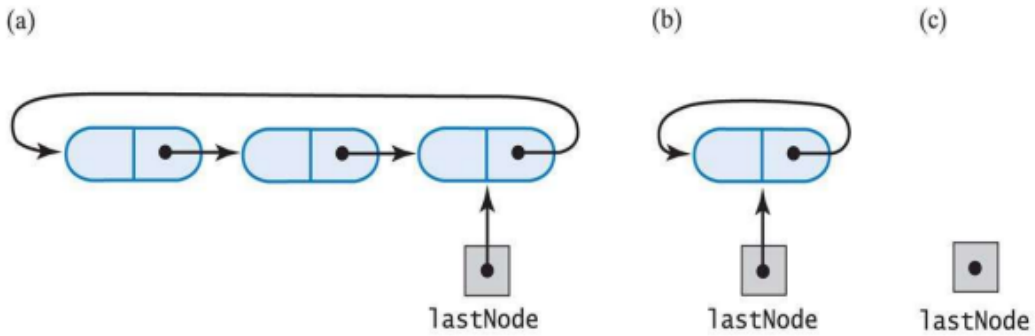




2. Explain how a queue may be implemented using a linked implementation with only one external reference to the last node.

Circular Linked Implementation of Queue





Chapter 6 Pre-Practical Class Submission

1. Explain the steps that need to be carried out in the **push** and **pop** operations when a stack is implemented using linked implementation with a reference to the top entry.

push() method

- Create a new node
- Make the new node point to the current top node
- Make the head reference **topNode** point to the new node

```
Node newNode = new Node(newEntry);
newNode.next = topNode;
topNode = newNode;
```

pop() method

If the stack is not empty

- Assign current top node to a reference to be returned
- Make the head reference **topNode** point to the next node

```
T topEntry = topNode.data;
if(topNode!=null)
      topNode = topNode.next;
return topEntry;
```

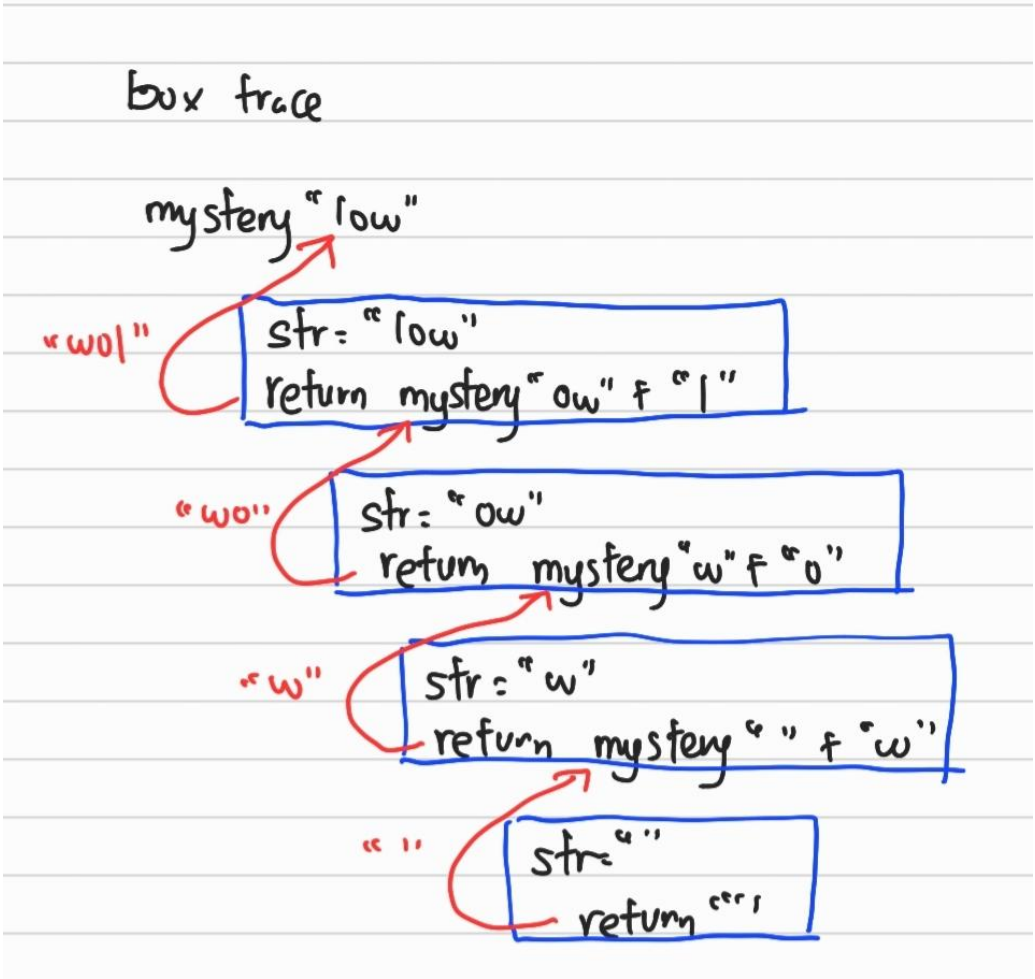
2. Consider the following recursive method:

```
public static String mystery(String str) {
   if (str.length() == 0) {
      return "";
   } else {
      return mystery(str.substring(1)) + " " +
str.charAt(0);
   }
}
```

Do a box trace for the call to the recursive method **mystery** when the value for the parameter str is the first 3 letters of your name in uppercase.

For each box, clearly indicate

- the argument value,
- the statements that are executed, and
- the return value.



Chapter 7 Pre-Practical Class Submission

1. Describe the difference between the generic type declarations for ListInterface and SortedListInterface. Explain the reason for this difference.

    - SortedListInterface, use `Comparable` interface
    - The objects in the sorted list must be Comparable i.e.must implement the method compare To
    - To enforce this requirements, we write
    - <T extends Comparable<T>>

2. Describe the difference between the constructors for the ArrayList class and the SortedArrayList class. Explain the reason for this difference.

    - new Object in the ArrayList
    - Note that new statement to construct the array in the constructor;
    - List = `(T[]) new Comparable[initialCapacity]`
    - Because the generic type enforces the requirements that the entries are Comparable

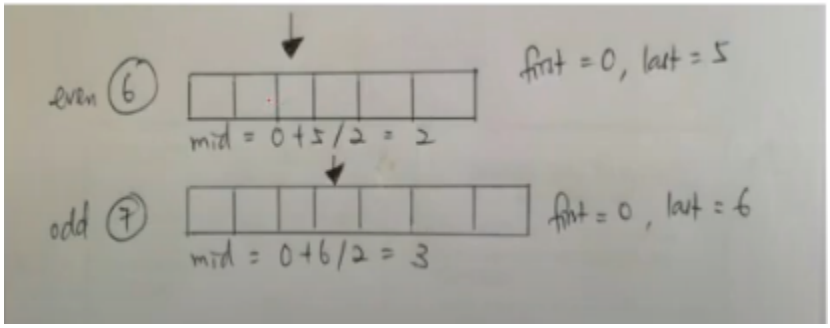
Chapter 8a Pre-Practical Class Submission

1. Describe how sequential search is carried out on a sorted array.

Index	0	1	2	3
Sorted Array	E	H	O	Y

Linear searching for each entry using compareTo(), when Target entry is compareTo array[i] current entry >0, STOP searching.

2. Describe how binary search is carried out.



- Divide the array in half and compare the target with the element at the array's mid-point
- If they match, the target is found
- Otherwise, if the target is less than the middle element, search the left subarray
- Otherwise, search the right subarray

Chapter 8b Pre-Practical Class Submission

Consider the following array of programme codes:

RSD2	RIT2	RSF2	REI2	RST2	RIS2	RMM2

1. Demonstrate the steps of *Selection Sort* when sorting the programme codes ascendingly. You are required to show the contents after each pass.

Selection Sort in PrePracQue: Search for smallest , Swapping needed

Passes	RSD2	RIT2	RSF2	REI2	RST2	RIS2	RMM2	
End P1		RIT2	RSF2		RST2	RIS2	RMM2	Swap REI RSD, i0
End P2	REI2	RIS2	RSF2	RSD2	RST2	RIT2	RMM2	Swap RIT RIS i1
End P3	REI2	RIS2	RIT2	RSD2	RST2	RSF2	RMM2	Swap RSF RIT i2
End P4	REI2	RIS2	RIT2	RMM2	RST2	RSF2	RSD2	Swap RMM RSD i3
End P5	REI2	RIS2	RIT2	RMM2	RSD2	RSF2	RST2	Swap RST RSD i4
End P6	REI2	RIS2	RIT2	RMM2	RSD2	RSF2	RST2	No swapping i5

2. Demonstrate the steps of *Insertion Sort* when sorting the programme codes ascendingly. You are required to show the contents after each pass.

Insertion Sort in PrePracQue: Shifting needed

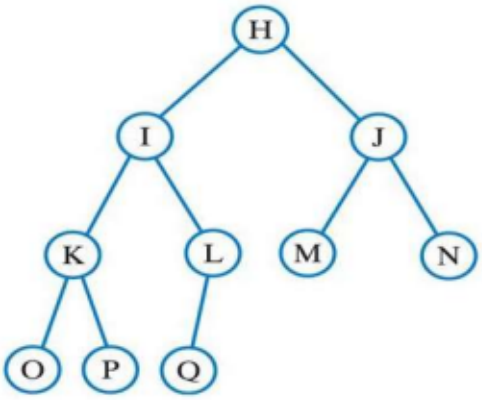
Passes	RSD2	RIT2	RSF2	REI2	RST2	RIS2	RMM2	
End P1	RIT2	RSD2	RSF2	REI2	RST2	RIS2	RMM2	Insert RIT in sorted
End P2	RIT2	RSD2	RSF2	REI2	RST2	RIS2	RMM2	Insert RSD in sorted
End P3	REI2	RIT2	RSD2	RSF2	RST2	RIS2	RMM2	Insert REI
End P4	REI2	RIT2	RSD2	RSF2	RST2	RIS2	RMM2	Insert RSF in sorted
End P5	REI2	RIS2	RIT2	RSD2	RSF2	RST2	RMM2	Insert RIS
End P6	REI2	RIS2	RIT2	RMM2	RSD2	RSF2	RST2	Insert RMM

Chapter 9 Pre-Practical Class Submission

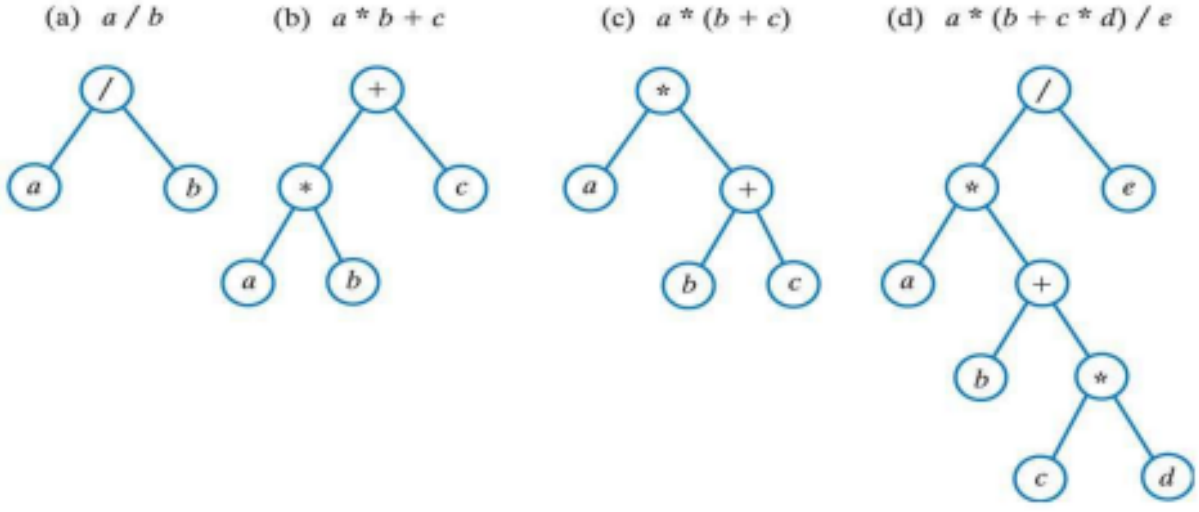
1. What is a binary tree?

Binary Trees

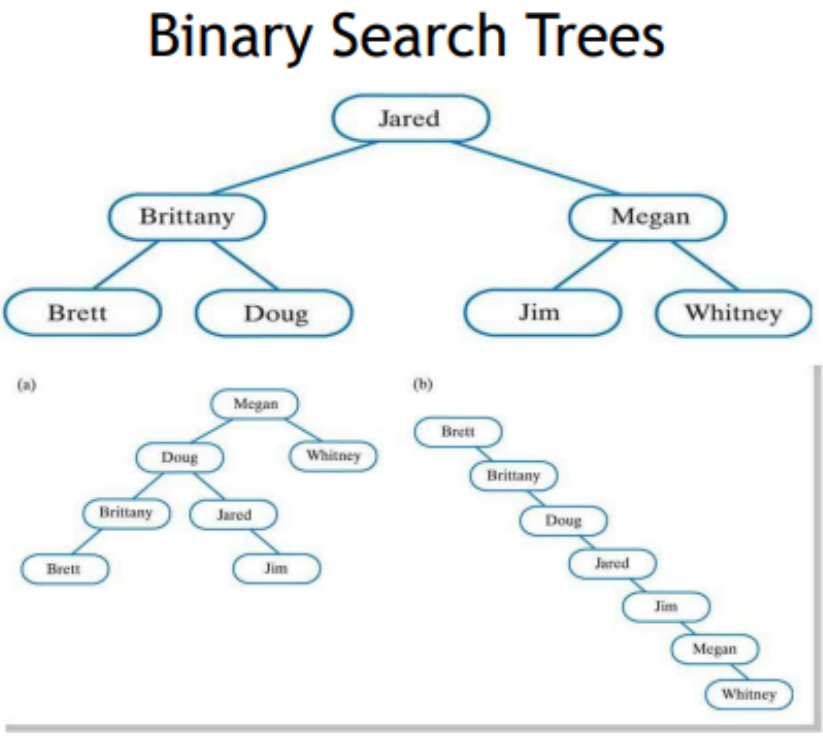
- A tree in which each node has *at most two children*.

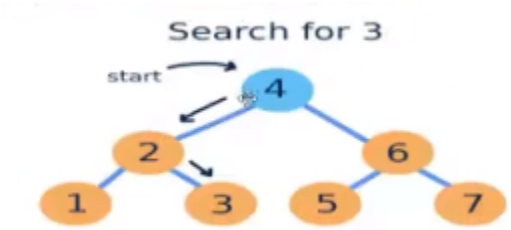


2. What is an expression tree?
    - An expression tree represents an algebraic expression whose operators are binary



3. What is a binary search tree?
    - Nodes contains Comparable objects (compare To())
    - A node's data is greater than the data in the node's left subtree
    - A node's data is less than the data in the node's right subtree





Chapter 10 Pre-Practical Class Submission

1. You are intending to store the details of 75 students of the Vegan Society into a hash table with table size as 83. Design an appropriate **hash function** using the Student ID (e.g., 20PMD1250, 20PMD1251, etc) as input. Express your answer in coding or algorithm.

```
getHashIndex(studentID)
     i = last 4 digits of StudentID // Step 1
     return i % 83                  // Step 2
```

2. What is the *folding* technique and how can it be used to produce the hash index of a 16-digit key?
   - It break the key into group of digits and then combine the group by using either addition or bitwise operator such as exclusive OR
   - The number of digit in a group should correspond to the size of the array

For example:
16 digits: **1234 5678 2355 6543** need to fold inside a hash table size of 25

1234 + 5678 + 2355 + 6543 = 15720
Compress them into the range of the table size:
15720 % 25 = 20

Hence, **1234 5678 2355 6543** is inserted into hash table at address 20