

2022/2023

Lab 7: Binary Search Tree & Priority Queue

FAKULTI TEKNOLOGI KEJURUTERAAN KELAUTAN DAN INFORMATIK

**DATA STRUCTURE & ALGORITHM**



**VERSION 1**

STUDENT INFORMATION

**Please fill in your details:**

**Name: arun mugilan a/l sargunan**

**MATRIC NUMBER:s63746**

**GROUP:K2**

**LAB:MP3**

**DATE:3/1/2023**

Table of Contents

[INSTRUCTIONS 1](#_Toc24877445)

[TASK 1: Implementing Binary Search Tree (BST) 2](#_Toc24877446)

[TASK 2: Implementing Priority Queue 17](#_Toc24877447)

# INSTRUCTIONS

Manual makmal ini adalah untuk kegunaan pelajar-pelajar Fakulti Teknologi Kejuruteraan Kelautan dan Informatik, Universiti Malaysia Terengganu (UMT) sahaja. Tidak dibenarkan mencetak dan mengedar manual ini tanpa kebenaran rasmi daripada penulis.

Sila ikuti langkah demi langkah sebagaimana yang dinyatakan di dalam manual.

*This laboratory manual is for use by the students of the Faculty of Ocean Engineering Technology and Informatics, Universiti Malaysia Terengganu (UMT) only. It is not permissible to print and distribute this manual without the official authorisation of the author.*

*Please follow step by step, as described in the manual.*

# TASK 1: Implementing binary search tree (bst)

## Objective

In this task, students must be able to:

* Understand the concept of Binary Search Tree (BST).
* Implement BST.

## Estimated Time

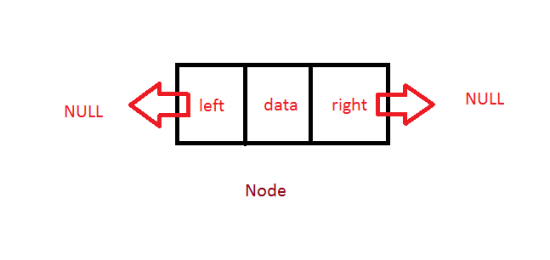
[60 Minutes]

### definition of binary search tree

**Binary Tree :** A data structure in which we have nodes containing data and two references to other nodes, one on the left and one on the right.

Binary Tree consist of Nodes

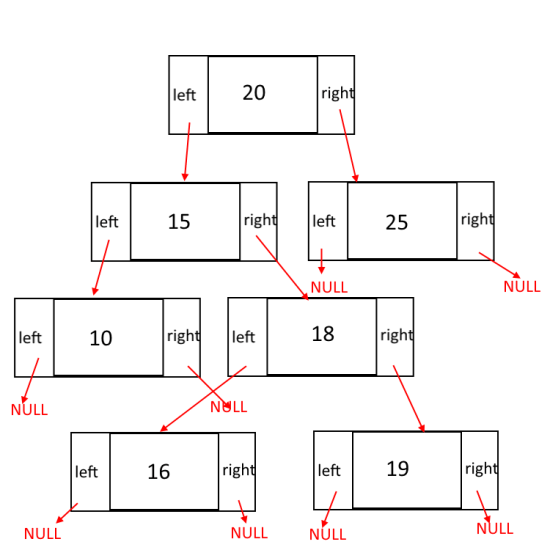
* Nodes are nothing but objects of a class and each node has data and a link to the left node and right node.
* Usually we call the starting node of a tree as *root*.
* Left and right node of a Leaf node points to NULL so you will know that you have reached to the end of the tree.

[](https://i0.wp.com/algorithms.tutorialhorizon.com/files/2014/09/Binary-Tree-Node.png)

**Binary Search Tree:**

Often we call it as BST, is a type of Binary tree which has a special property.

*Nodes smaller than root goes to the left of the root and Nodes greater than root goes to the right of the root*.

[](https://i2.wp.com/algorithms.tutorialhorizon.com/files/2014/09/Binary-Search-Tree.png)

**Operations:**

**Insert(int n) :** Add a node the tree with value n. Its O(lgn)

**Find(int n) :** Find a node the tree with value n. Its O(lgn)

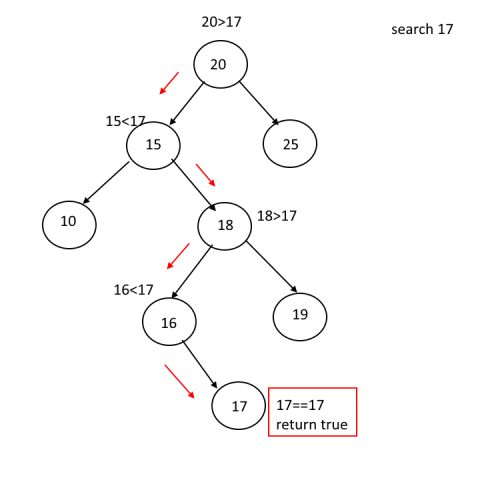
**Delete (int n)**: Delete a node the tree with value n. Its O(lgn)

**Display**(): Prints the entire tree in increasing order. O(n).

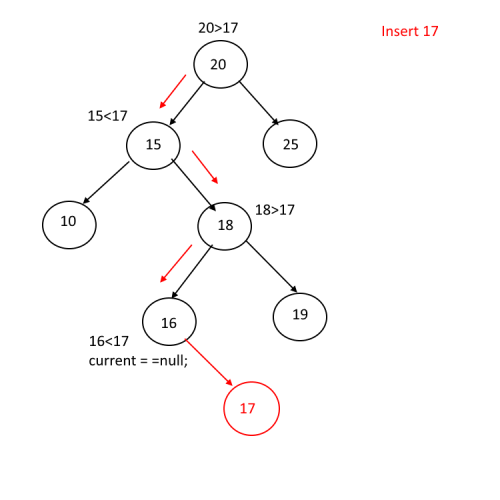
Detail Explanations for the Operations:

**Find(int n):**

* Its very simple operation to perform.
* start from the root and compare root.data with n
* if root.data is greater than n that means we need to go to the left of the root.
* if root.data is smaller than n that means we need to go to the right of the root.
* if any point of time root.data is equal to the n then we have found the node, return true.
* if we reach to the leaves (end of the tree) return false, we didn’t find the element

[](https://i1.wp.com/algorithms.tutorialhorizon.com/files/2014/11/BST-Find.png)**Insert(int n):**

* Very much similar to find() operation.
* To insert a node our first task is to find the place to insert the node.
* Take current = root .
* start from the current and compare root.data with n
* if current.data is greater than n that means we need to go to the left of the root.
* if current.data is smaller than n that means we need to go to the right of the root.
* if any point of time current is null that means we have reached to the leaf node, insert your node here with the help of parent node. (See code)

[](https://i0.wp.com/algorithms.tutorialhorizon.com/files/2014/09/BST-Insert.png)

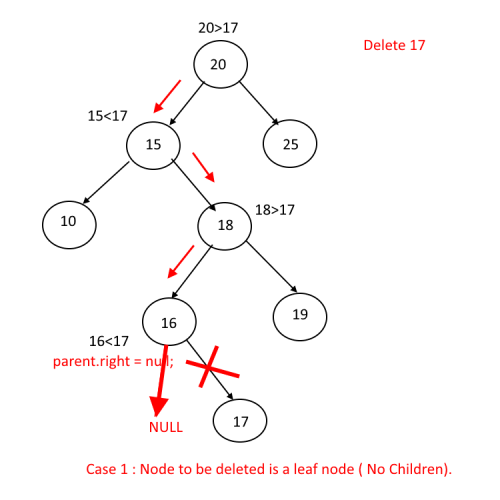
**Delete(int n):**

Complicated than Find() and Insert() operations. Here we have to deal with 3 cases.

* Node to be deleted is a leaf node ( No Children).
* Node to be deleted has only one child.
* Node to be deleted has two childrens.

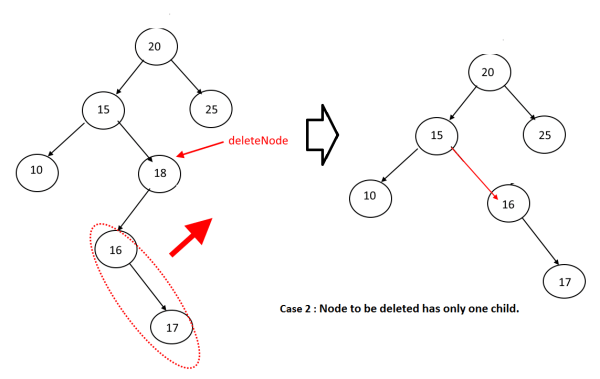
**Node to be deleted is a leaf node ( No Children).**

its a very simple case, if a node to be deleted has no children then just traverse to that node, keep track of parent node and the side in which the node exist(left or right) and set ***parent.left = null or parent.right = null;***

[](https://i0.wp.com/algorithms.tutorialhorizon.com/files/2014/09/BST-Node-to-be-deleted-is-a-leaf-node-No-Children.1.png)

**Node to be deleted has only one child.**

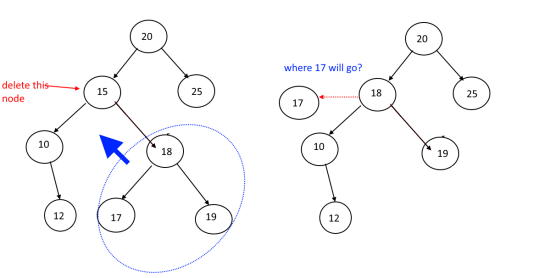
1. its a slightly complex case. if a node to be deleted(deleteNode) has only one child then just traverse to that node, keep track of parent node and the side in which the node exist(left or right).
2. check which side child is null (since it has only one child).
3. Say node to be deleted has child on its left side . Then take the entire sub tree from the left side and add it to the parent and the side on which deleteNode exist, see step 1 and example.

[](https://i2.wp.com/algorithms.tutorialhorizon.com/files/2014/09/BST-Node-to-be-deleted-has-only-one-child.1.png)

**Node to be deleted has two children.**

Now this is quite exciting

You just cannot replace the deleteNode with any of its child, Why? Lets try out a example.

[](https://i2.wp.com/algorithms.tutorialhorizon.com/files/2014/09/BST-Node-to-be-deleted-has-2-children-Example-1.png)

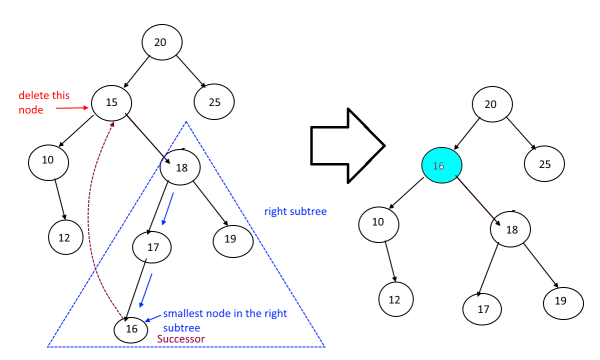
**What to do now?????**

Dont worry we have solution for this

**Find The Successor:**

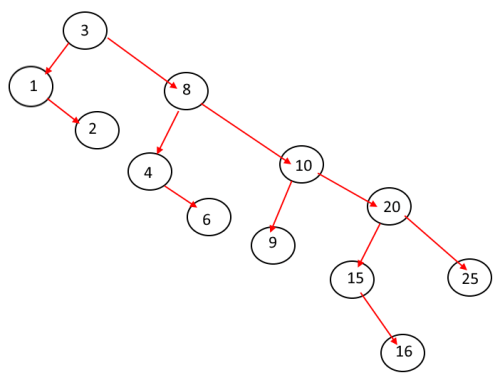
Successor is the node which will replace the deleted node. Now the question is to how to find it and where to find it.

*Successor is the smaller node in the right sub tree of the node to be deleted.*

[](https://i1.wp.com/algorithms.tutorialhorizon.com/files/2014/09/BST-Node-to-be-deleted-has-2-children-Example-2.png)

**Display()** : To know about how we are displaying nodes in increasing order, Click Here

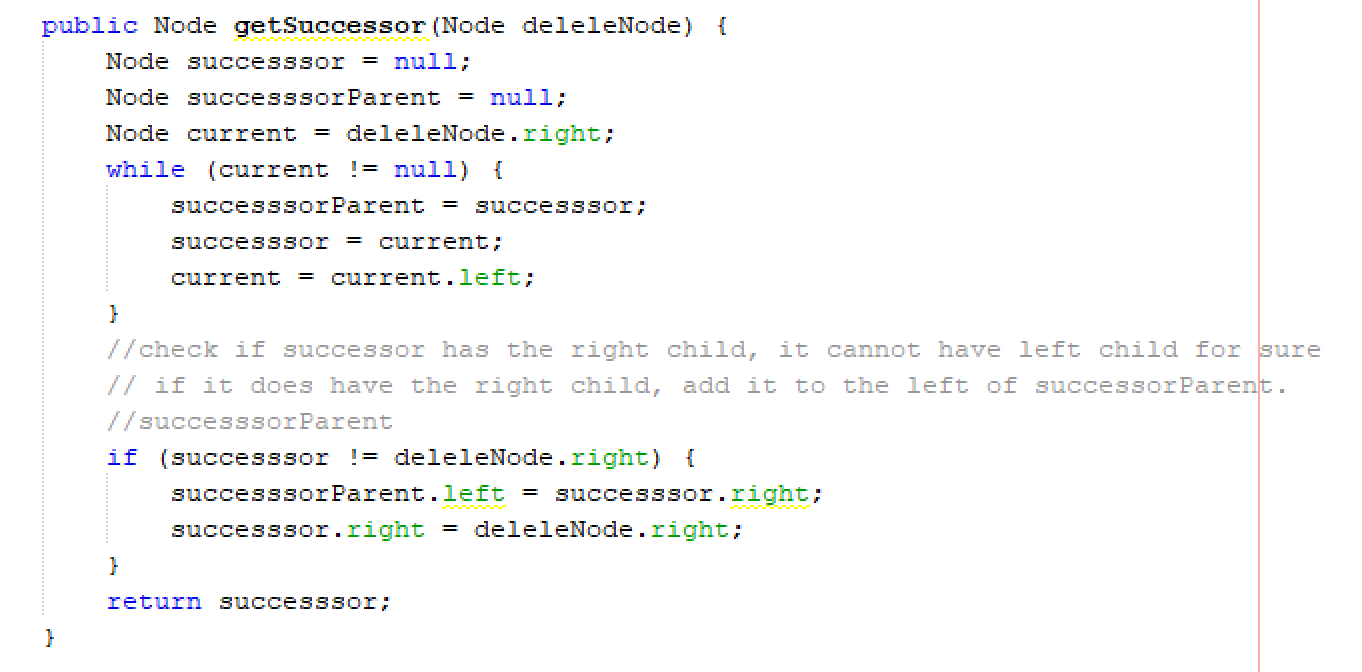
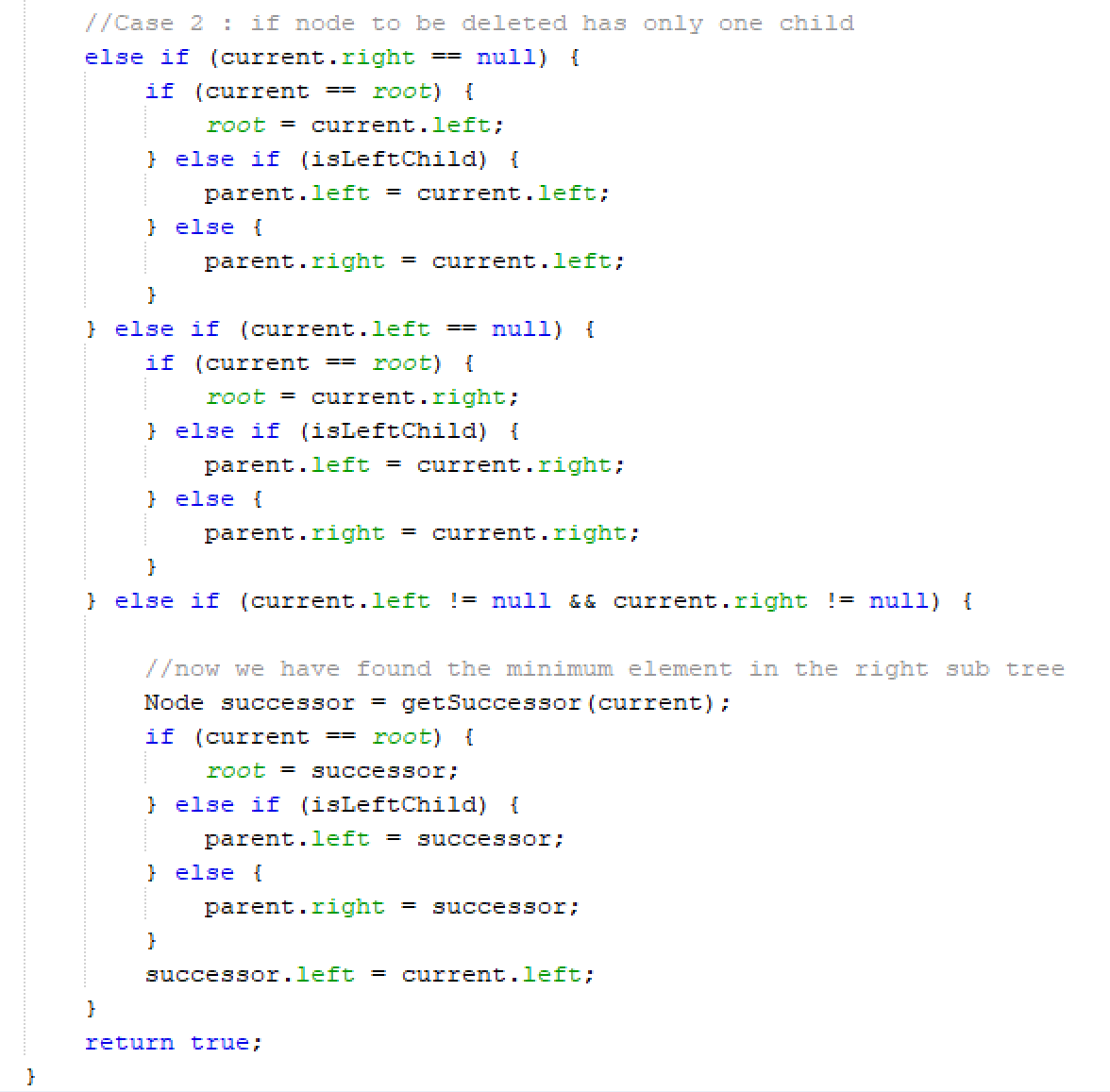
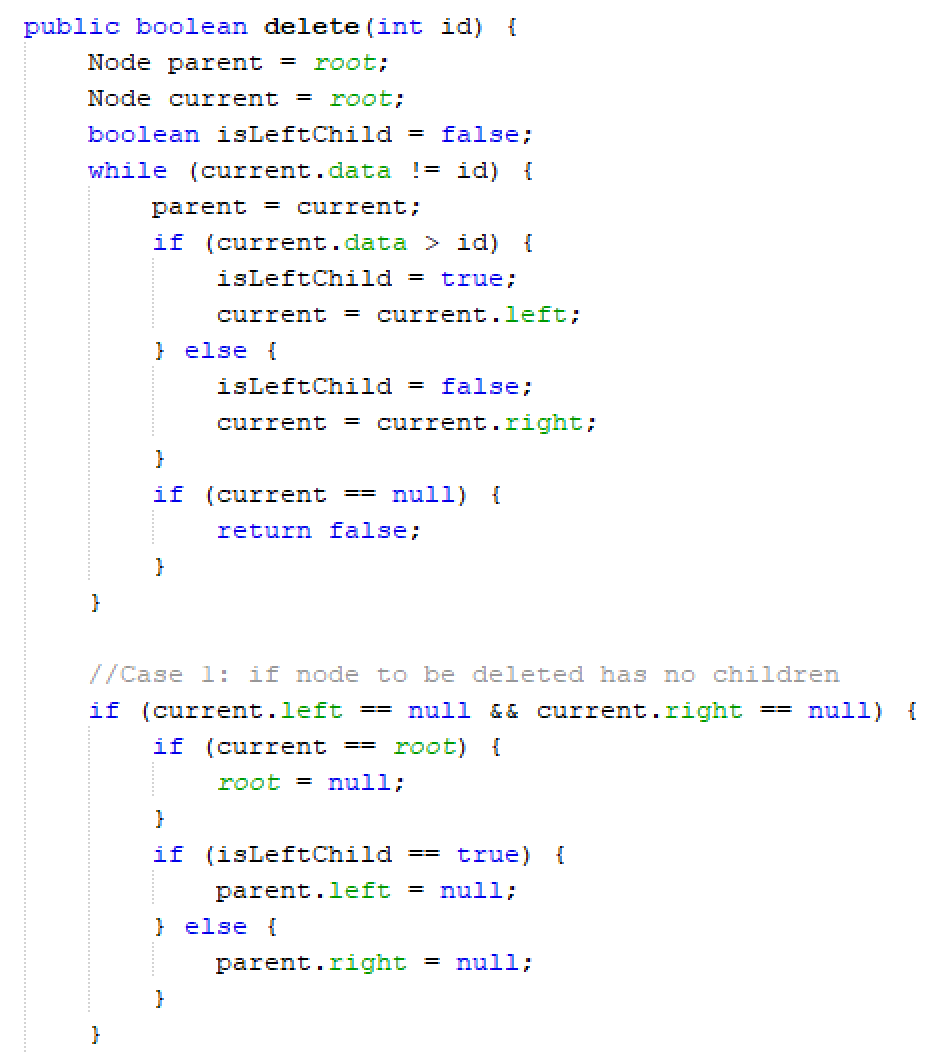
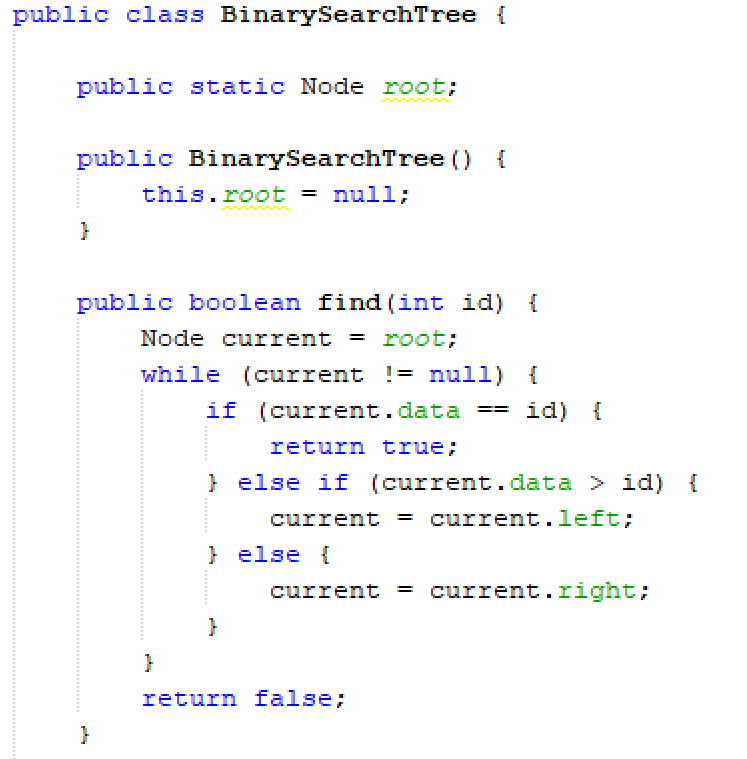
Complete Example :

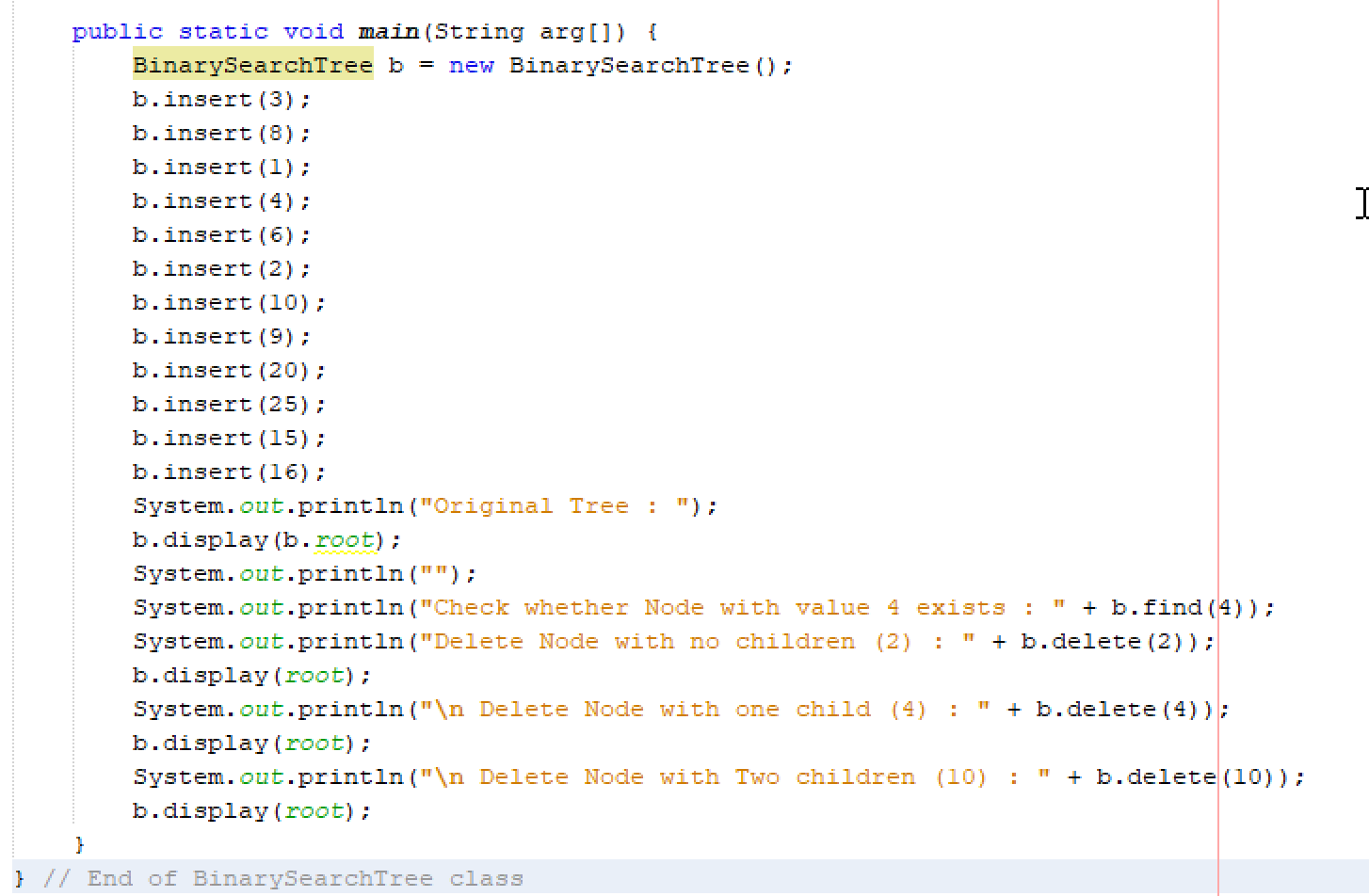
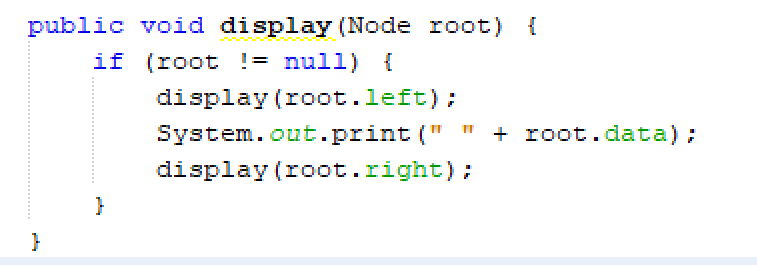
[](https://i1.wp.com/algorithms.tutorialhorizon.com/files/2014/09/Complete-Example.png)

### steps:

1. After you understand the explanation in the above section, open NetBeans and create a new java application project.
2. Name your project as BinarySearchTreeExperiment and click finish.
3. Change author profiles to :
   1. Name :
   2. Program: <put your program. E.g., SMSK(SE) or SMSK with IM
   3. Course : CSF3104
   4. Lab : <enter lab number>
   5. Date : <enter lab date>
4. In the same BinarySearchTreeEXperiment project’s package, create a new file named BinarySearchTree.java.
5. Add the following codes to the file:

*Note: The codes below are quite long. Take your time to understand the codes and write your own comment where applicable. This will help you to study the codes later.*

A screenshot of a cell phone

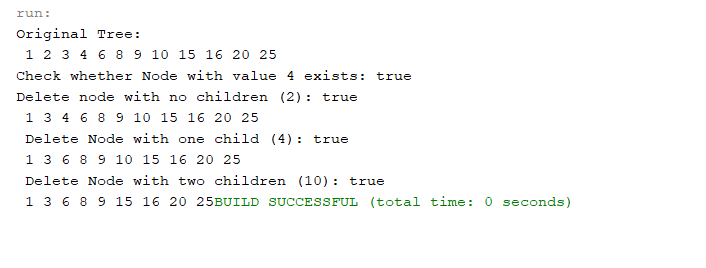
Description automatically generated

1. Just after the end of BinarySearchTree class, add the Node class below it:

A screenshot of a cell phone

Description automatically generated

1. Save, compile and run your codes. Observe the output.
2. Upload your output using the control box below:



1. Copy and paste your Java codes into the text box below:

**Answer:**

/\*

\* Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license

\* Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Main.java to edit this template

\*/

package binarysearchtreeexperiment;

public class BinarySearchTreeExperiment {

public static Node root;

public BinarySearchTreeExperiment(){

this.root=null;

}

public boolean find(int id){

Node current = root;

while (current != null){

if (current.data==id){

return true;

} else if (current.data > id) {

current = current.left;

} else{

current = current.right;

}

}

return false;

}

public boolean delete(int id){

Node parent = root;

Node current = root;

boolean isLeftChild = false;

while(current.data != id){

parent = current;

if (current.data>id){

isLeftChild = true;

current = current.left;

} else {

isLeftChild = false;

current = current.right;

}

if (current == null){

return false;

}

}

if (current.left == null && current.right == null){

if(current == root){

root= null;

}

if (isLeftChild == true){

parent.left = null;

} else {

parent.right = null;

}

}

else if (current.right == null){

if (current == root){

root=current.left;

} else if (isLeftChild){

parent.left=current.left;

} else {

parent.right=current.left;

}

} else if(current.left==null){

if(current==root){

root=current.right;

}else if (isLeftChild){

parent.left=current.right;

} else {

parent.right=current.right;

}

} else if (current.left !=null && current.right != null){

Node successor = getSuccessor(current);

if(current==root){

root=successor;

}else if (isLeftChild){

parent.left=successor;

} else {

parent.right=successor;

}

successor.left = current.left;

}

return true;

}

public Node getSuccessor (Node deleteNode){

Node successor = null;

Node successorParent = null;

Node current = deleteNode.right;

while (current != null){

successorParent = successor;

successor = current;

current = current.left;

}

if (successor != deleteNode.right){

successorParent.left = successor.right;

successor.right = deleteNode.right;

}

return successor;

}

public void insert(int id){

Node newNode = new Node(id);

if(root==null){

root = newNode;

return;

}

Node current = root;

Node parent = null;

while(true){

parent = current;

if (id < current.data){

current= current.left;

if(current == null){

parent.left=newNode;

return;

}

}else {

current = current.right;

if (current == null){

parent.right = newNode;

return;

}

}

}

}

public void display(Node root){

if (root != null ){

display(root.left);

System.out.print(" "+ root.data);

display(root.right);

}

}

public static void main(String[] args){

BinarySearchTreeExperiment b = new BinarySearchTreeExperiment();

b.insert(3);

b.insert(8);

b.insert(1);

b.insert(4);

b.insert(6);

b.insert(2);

b.insert(10);

b.insert(9);

b.insert(20);

b.insert(25);

b.insert(15);

b.insert(16);

System.out.println("Original Tree: ");

b.display(b.root);

System.out.println("");

System.out.println("Check whether Node with value 4 exists: " + b.find(4));

System.out.println("Delete node with no children (2): " + b.delete(2));

b.display(root);

System.out.println("\n Delete Node with one child (4): " + b.delete(4));

b.display(root);

System.out.println("\n Delete Node with two children (10): " + b.delete(10));

b.display(root);

}

}

class Node{

int data;

Node left;

Node right;

public Node (int data){

this.data=data;

left=null;

right=null;

}

}

### questions

1. Discuss the differences between General Tree and Binary Search Tree.

**Answer:**

**The general tree is a tree in which each node can have many children or nodes.** **Whereas in a binary tree, each node can have at most two nodes**.

1. What are the advantages of BST?

**Answer:**

BST **makes it easier to store data and to move from one storage space to another whenever needed**. This makes data storage and movement easy from one system to another.

# TASK 2: Implementing priority queue

## objective

During this activity, students will learn and apply the concept of priority queue data structure.

## estimated time

[60 Minutes]

### Introduction

The priority queue is a somewhat similar data structure to the queue. The difference lies in how the elements are being processed:

* A standard queue strictly follows the FIFO (First-In-Last-Out) principle.
* A priority queue does not follow the FIFO principle.

In a priority queue, the elements are being removed from the queue based on their *priority.*This translates to the requirement that:

**Every element in a priority queue must have a priority associated with it.**

As you might have guessed, the element with the highest priority is removed from the queue (dequeued).

But how do should you define the priority of the elements in the queue?

Basically, you have two alternatives for defining priorities to the elements in the queue. You can either:

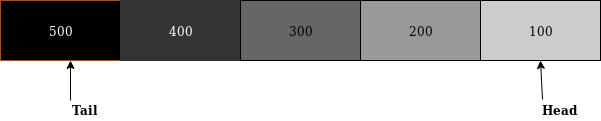
* Order elements based on their natural ordering.
* Order elements with a custom Comparator.

In this article, we'll focus on how to implement a priority queue. So for simplicity's sake, we'll order the elements based on their natural ordering.

In our example, the priority queue will be limited to int, so natural ordering is perfectly fine. However, keep in mind that this is for demonstration purposes only.

If you were to implement a priority queue in real life, you probably want to make use of generics — or just do like any other developer and use the built-in java.util.PriorityQueue.

To keep our example implementation compliant with the Java specification, the least element is defined as the element with the highest priority.



### Priority Queue Operations

The most basic set of operations for any implementation of a queue is:

enqueue — Inserting an element into the queue.

dequeue — Removing an element from the queue.

isEmpty — Returning true if the queue is empty.

size — Returning the size/number of elements in the queue.

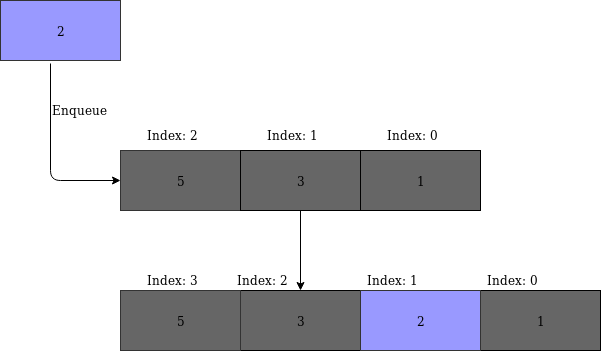
contains — Returning true if the queue contains the given element.

peek — Returning the front element of the queue, without removing it.

Please note that the Java implementation of the priority queue uses different names for the methods. In a production environment, I highly suggest that you use the default implementation of the priority queue, instead of "home-growing" it.

**Implementing Enqueue and Dequeue**

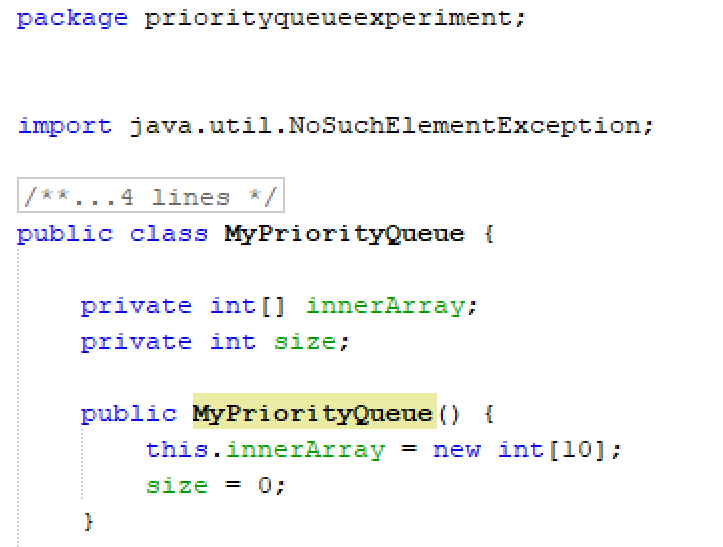
First, let's think about what we want to happen if we insert/enqueue an element. Note that we ignore the doubleArraymethod for now. We want inserted elements to be placed in the queue, in the correct position, according to its priority.



This visualization of the enqueueoperation tells us that we'll have to shift all elements of lower priority one position up in the array.

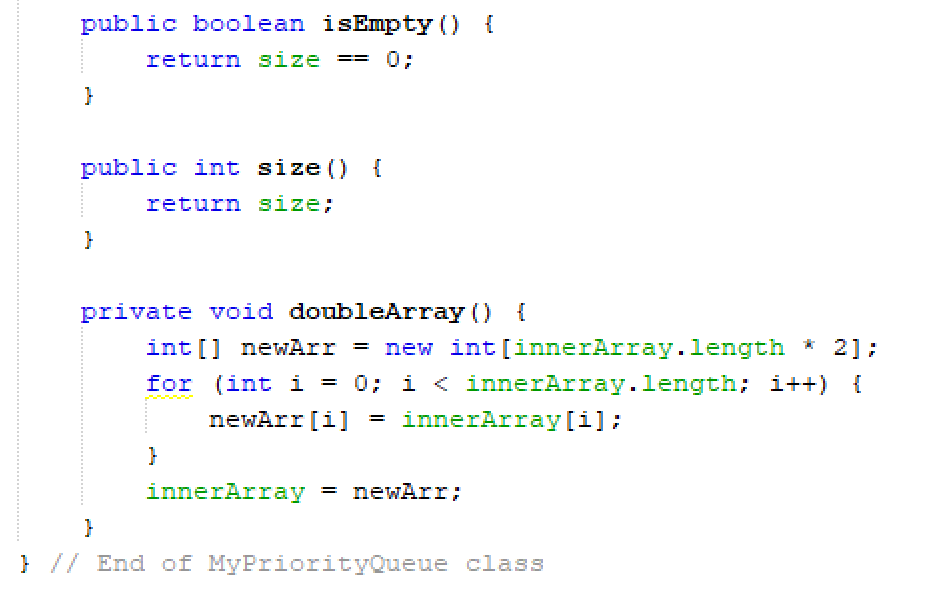
### steps:

1. Create a new Netbeans project. Name the project as PriorityQueueExperiment.
2. Create a new class named MyPriorityQueue.
3. In MyPriorityQueue.java, insert the following codes, again, try to understand the codes and write your own comment:

A screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generatedA screenshot of a social media post

Description automatically generated

1. Write a test class by completing the codes section below for MyPriorityQueue class:

A screenshot of a cell phone

Description automatically generated

1. Save, compile and execute your codes.
2. Upload your output using the control box below:

Table

Description automatically generated with medium confidence

1. Copy and paste your Java codes from NetBeans into text box below:

package priorityqueueexperiment;

import java.util.NoSuchElementException;

public class MyPriorityQueue {

private int[] innerArray;

private int size;

public MyPriorityQueue(){

this.innerArray = new int[10];

size =0;

}

public void enqueue(int x){

if (size == 0){

size++;

innerArray[0]=x;

return;

}

if (size()== innerArray.length){

doubleArray();

}

int temp = x;

for (int i =0; i<size; i++){

if (x <= innerArray[i]){

int next;

temp = innerArray[i];

innerArray[i]=x;

while (i< size-1){

next = innerArray[i+1];

innerArray[i+1] = temp;

temp = next;

i++;

}break;

}

}

innerArray[size] = temp;

size++;

}

public int dequeue(){

if (isEmpty()){

throw new NoSuchElementException("The queue is empty");

}

int retValue = innerArray[0];

for (int i=1; i< size; i++){

innerArray[i-1]= innerArray[i];

}

innerArray[size - 1] = 0;

size--;

return retValue;

}

public int peek(){

if (isEmpty()){

throw new NoSuchElementException("The queue is empty");

}

return innerArray[0];

}

public boolean contains(int x){

if(isEmpty()){

return false;

}

for (int i=0; i< size; i++){

if (innerArray[i]== x){

return true;

}

}

return false;

}

public boolean isEmpty(){

return size == 0;

}

public int size(){

return size;

}

public void doubleArray(){

int[] newArr = new int [innerArray.length \*2];

for(int i =0; i< innerArray.length; i++){

newArr[i]= innerArray[i];

}

innerArray = newArr;

}

}

public class MPQtest {

public static void main (String[]args){

MyPriorityQueue c = new MyPriorityQueue();

c.enqueue(21);

c.enqueue(11);

c.enqueue(7);

c.enqueue(15);

c.enqueue(5);

c.enqueue(9);

c.enqueue(3);

while(!c.isEmpty()){

System.out.println(c.dequeue());

}

}

}

### questions

1. List the differences between common Queue and Priority Queue.

**Answer:**

In a queue, the first-in-first-out rule is implemented whereas, in a priority queue, the values are removed on the basis of priority. The element with the highest priority is removed first.

1. In Java, there is a built-in class for PriorityQueue, how can you use it?

**Answer:**

A priority queue is used when the objects are supposed to be processed based on priority. It is known that a [Queue](https://www.geeksforgeeks.org/queue-interface-java/) follows the First-In-First-Out algorithm, but sometimes the elements of the queue are needed to be processed according to the priority, that’s when the priority queue comes into play.

1. What are the ADTs for PriorityQueue?

**Answer:**

Priority Queue is an Abstract Data Type (ADT) that holds a collection of elements, it is similar to a normal Queue, the difference is that the elements will be dequeued following a priority order.

Finally, read the instruction regarding submission carefully. Submit your answer using the link provided in Oceania UMT. Please ensure your codes are submitted to the correct group.