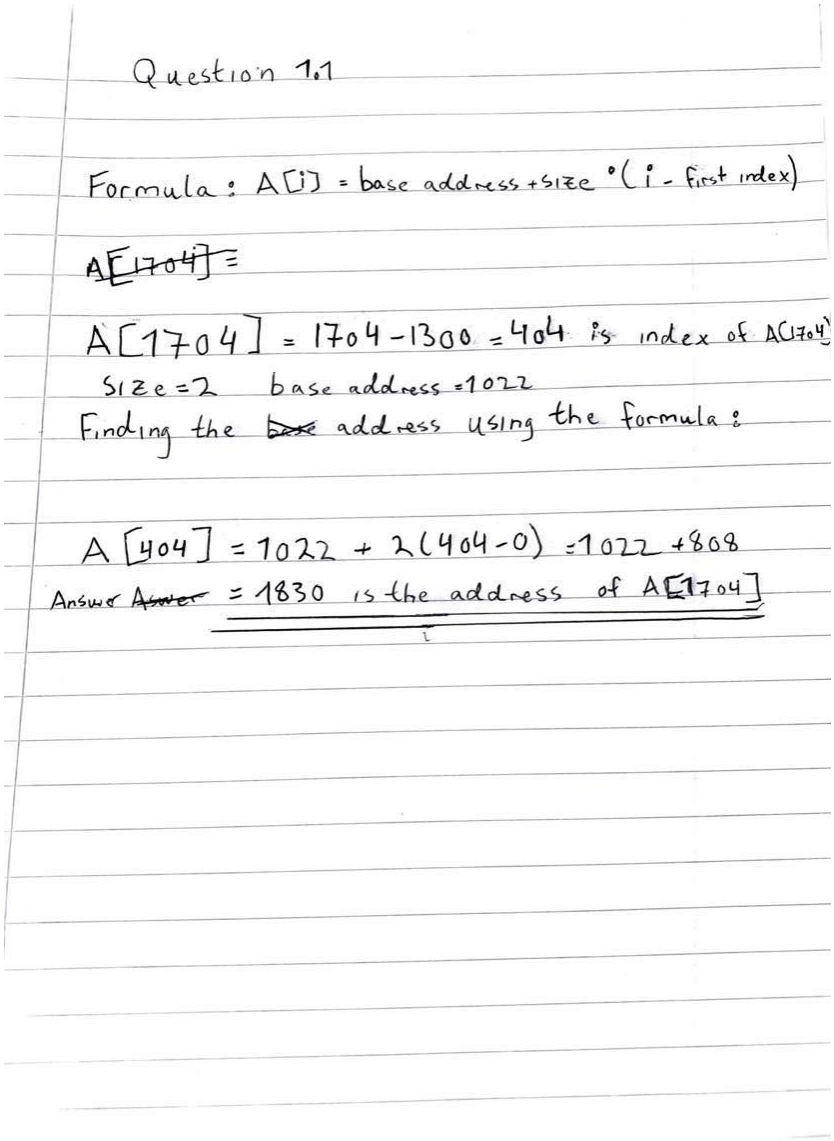
Question 1.1:



Question 1.1

Formula: A[i] = base address + size * (i - first index)

~~A[1704] =~~

A[1704] = 1704 - 1300 = 404 is index of A[1704]

size = 2 base address = 1022

Finding the ~~base~~ address using the formula:

A[404] = 1022 + 2(404 - 0) = 1022 + 808

Answer ~~Answer~~ = 1830 is the address of A[1704]

Question 1.2:

Step by Step algorithm to find the largest number in array:

1) Initialize a variable in the type of an int and call it: “maxNumber” and set the max variable equal to the first element in array as default value.
2) Create a loop that iterates through each element in the array starting from the second element in the array.
3) Create an if statement inside the loop, if element is greater than the largest value, set the largest value equal to the element.
4) Print the largest value outside the loop and the largest value should be displayed in terminal.

Question 2.1:

Stack push algorithm:

1) When we insert elements in stack, we check whether its full or not.
2) When we try inserting element in a full stack it can cause an overflow.
3) Set the top value equal to -1, which checks whether the stack is empty when its initialized.
4) When pushing a new element in the stack, set the top value equal to top+1 so that the stack increments when a new element is added. When an element is pushed, it is added on the top of the stack.
5) The pushed elements are added until we reach an overflow condition in the stack.

Stack pop algorithm:

1)When deleting an element from the stack, we first check If its empty or not.

2) If we try deleting an element from an empty stack, it can cause an underflow condition.

3)if it's not empty, then access the top element in the stack.

4) The top is decremented by 1 after the performing the pop operation to the stack

Question 2.2:

Explanation:

1) First create a stack in the type of int and make it public
2) Initial a variable called minElement and make it also public
3) After that create a method called push and set int = x as parameter. Inside the method create a if statement, if stack is empty minElement = x. push x to the stack and return -1. Set return x outside the loop and display "the element is pushed to stack" every time an element is pushed.

4) Create a method called “pop”. Inside the method create a if statement, if stack is empty, return -1 , and display the message element is removed from stack. Return the minElement
5) Lastly create a getMin() method that displays the minimum value from the stack. Use a if statement, if stack is empty return -1 else return the minElement.

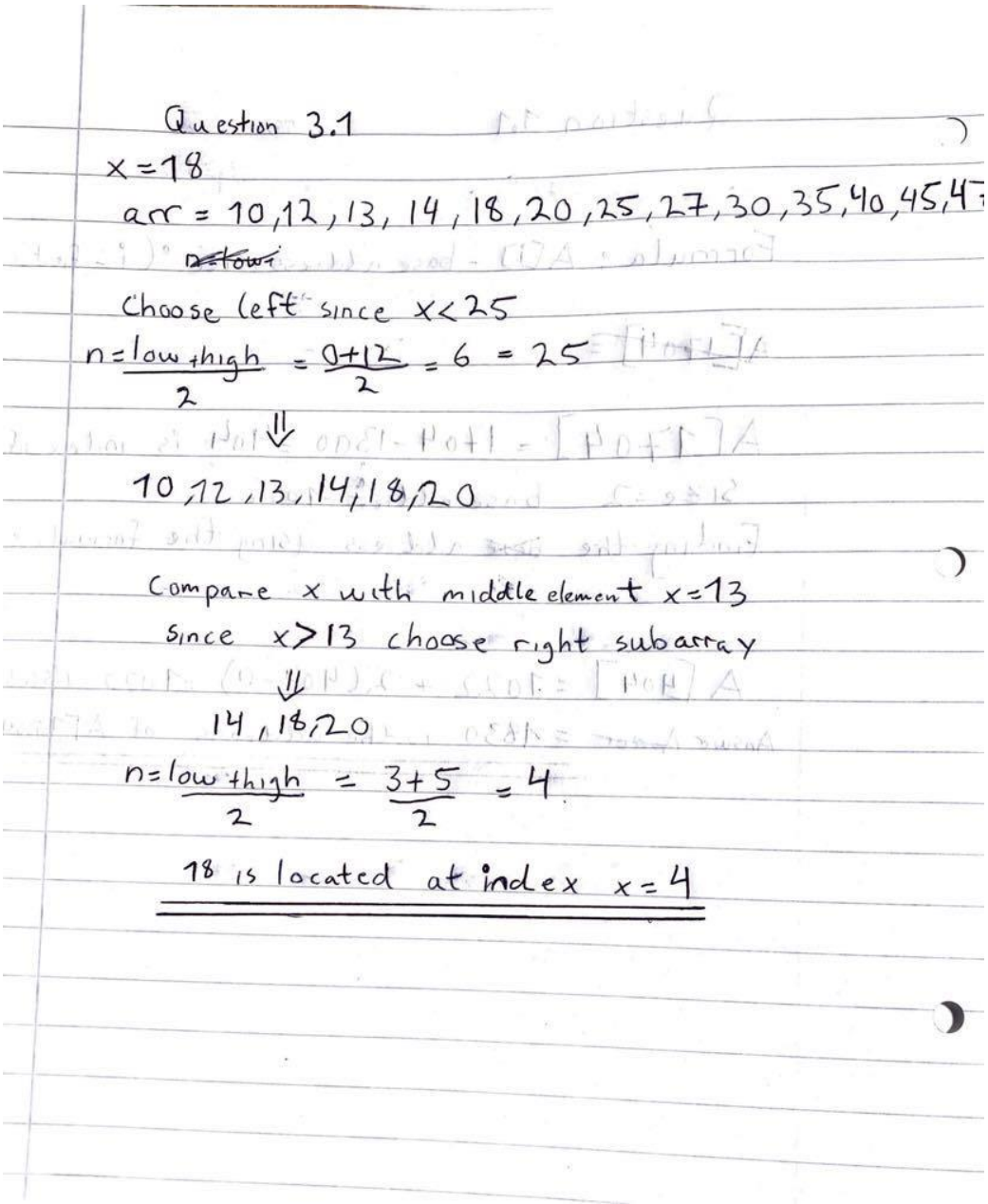
See my code in the java folder Question2.java

Question 3.1)

Step by Step Algorithm of Binary Search Recursive version:

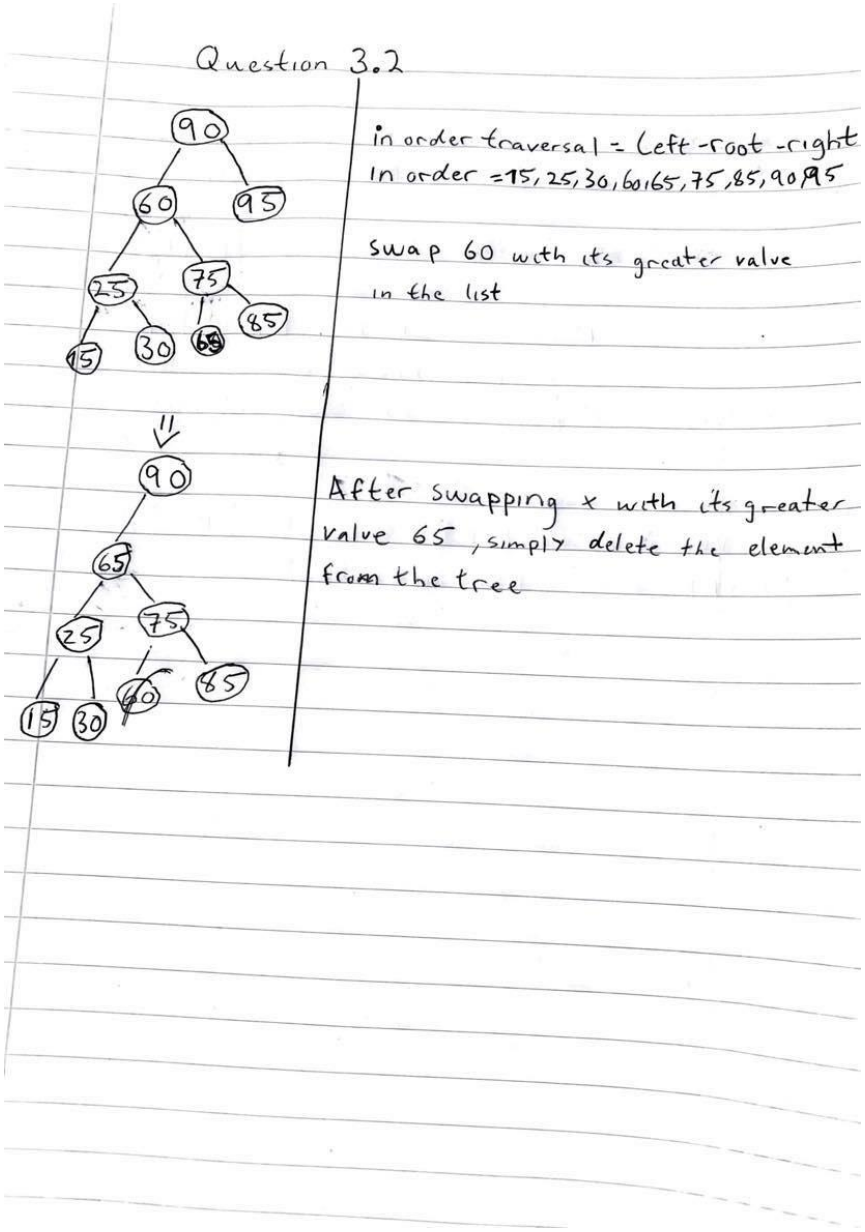
1) Initialize the variables: x , arr , lowIndex , highIndex in the type of int.
2) Create a variable named midValue set this equal to (lowIndex+highIndex/2) which calculates the middle value.
3) If x is equal to the midValue, the operation is done and the key is found
4) Else the array splits into two same sized subarrays.
5) If x is greater than the midValue , choose right subarray and set the highIndex equal to midValue -1
6) Else choose left and set the lowIndex equal to midValue +1
7) The method uses recursion to call itself until the subarray is small and the key is found!

Human representation:



Question 3.2)

Human representation:



Step by Step algorithm Deletion of Node having two children:

1) When deleting a node having two children, we simply use in order traversal in this case, after this we simply replace the node, we want to delete (x=60) with its successor.
2) The in-order traversal of this binary tree is: (15,25,30,60,65,75,85,90,95)
3) Replace node 60 with its successor (65). The deleted node is now replaced by 65 and is now a leaf node (node without children) which is to be deleted.

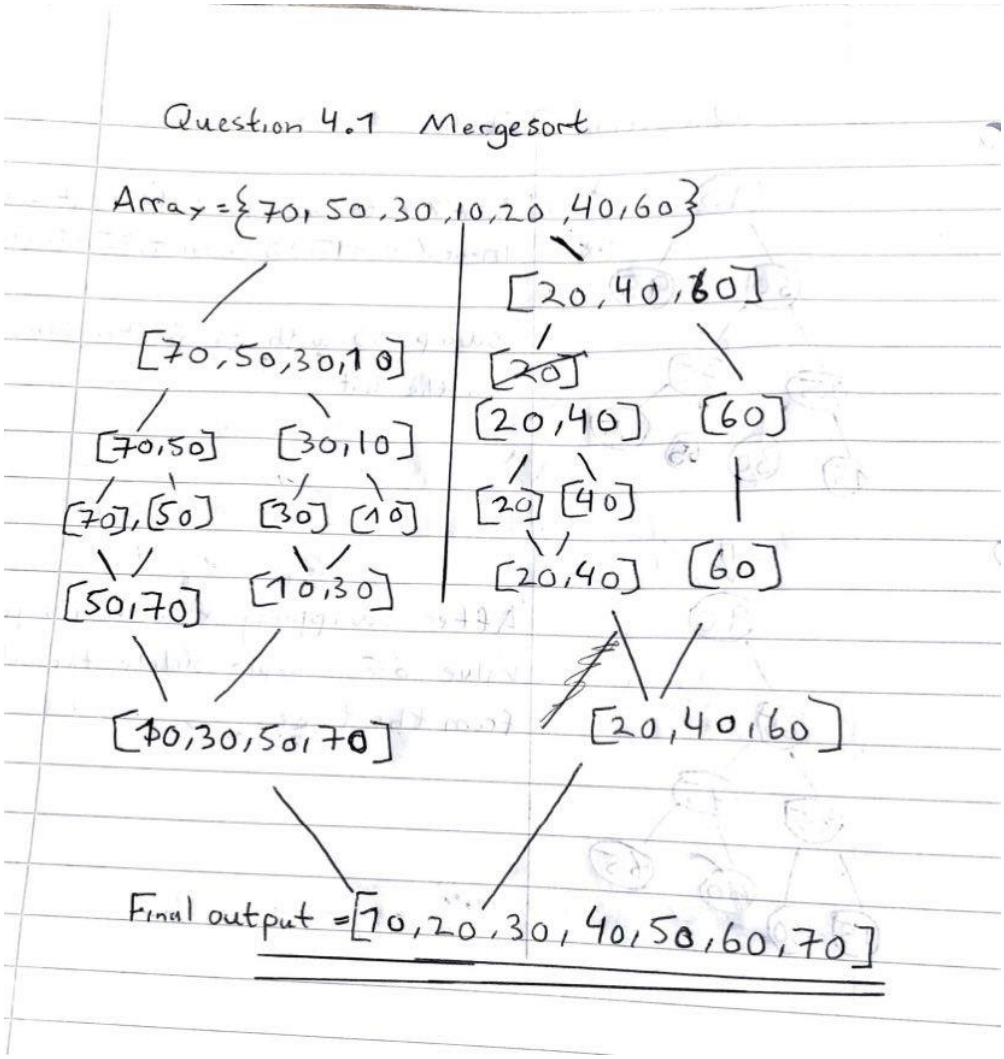
Question 4.1)

Step by Step Algorithm Merge Sort:

1) First, we start the operation by dividing the array (70,50,30,10,20,40,60) into two subarrays where the first has four element and the other has three

2) After that we conquer each element in the subarray until the array gets smaller
3) Lastly, we merge the elements in the subarrays together and the sorting is done.

Graphical Representation below:



Merge Sort Table format:

Arr = [70,50,30,10,20,40,60]

k	U	V	Sorted (Result)
1	**70,50,30,10**	**20,40,60**	**10**
2	**70,50,30,10**	**20,40,60**	**10,20**
3	**70,50,30,10**	**20,40,60**	**10,20,30**

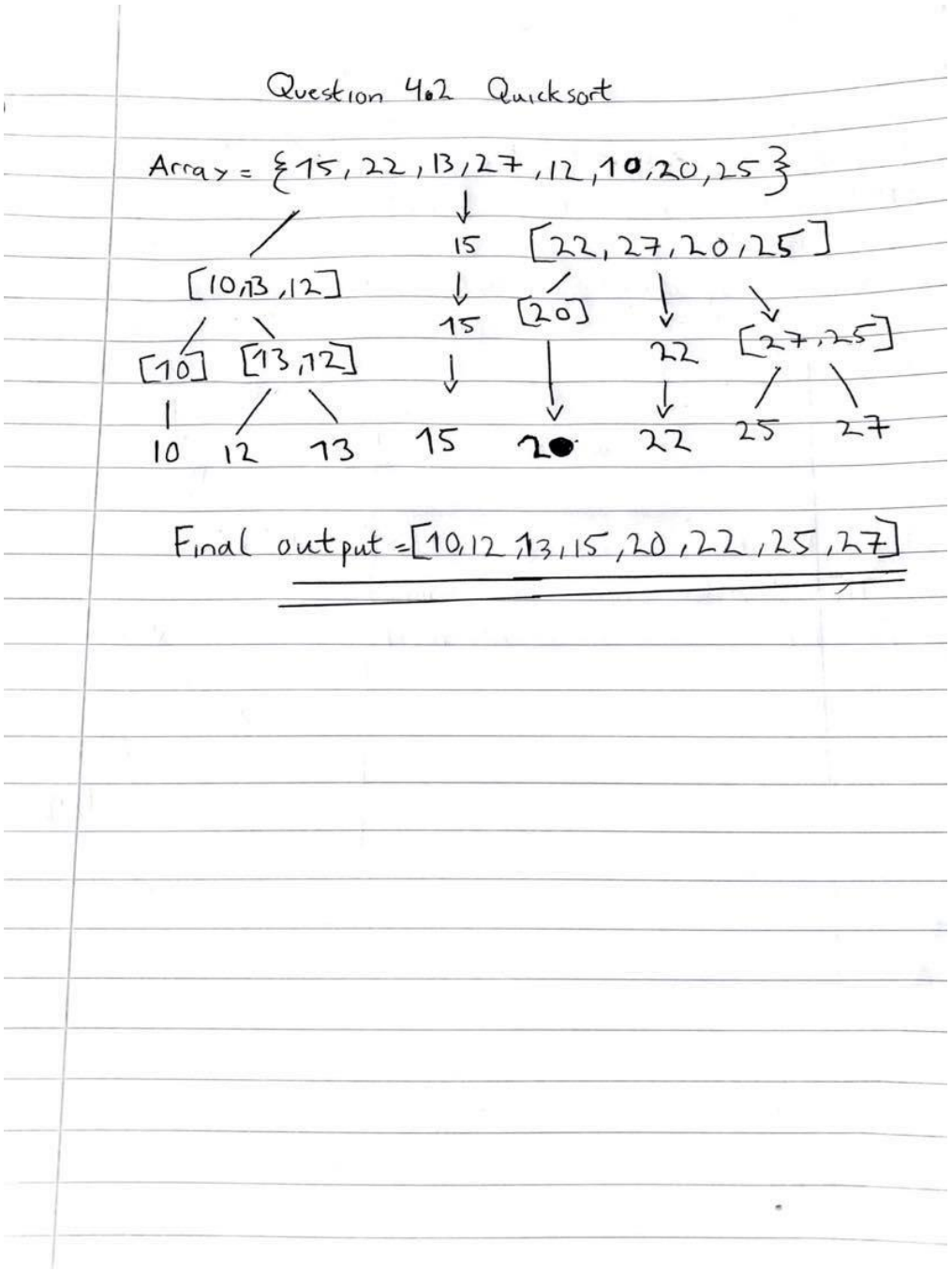
4	**70,50,30,10**	**20,40,60**	**10,20,30,40**
5	**70,50,30,10**	**20,40,60**	**10,20,30,40,50**
6	**70,50,30,10**	**20,40,60**	**10,20,30,40,50,60**
7	**70,50,30,10**	**20,40,60**	**10,20,30,40,50,60,70**
----------	**70,50,30,10**	**20,40,60**	**10,20,30,40,50,60,70 (Final values)**

Question 4.2:

Step by Step Algorithm to Quicksort:

1) Divide the array into two subarrays
2) Choose a pivot element before starting the partition of the element, its optional which element you choose as pivot. The first element was chosen as the pivot element for this task.
3) The pivot element splits the array into two separate subarrays.
4) Elements that are lower than the pivot is placed in the left subarray.
5) Elements that are greater than the pivot element is placed on the right subarray.
6) After recursively swapping the elements the subarrays the array is now sorted.

Graphical Representation:



Question 5.1

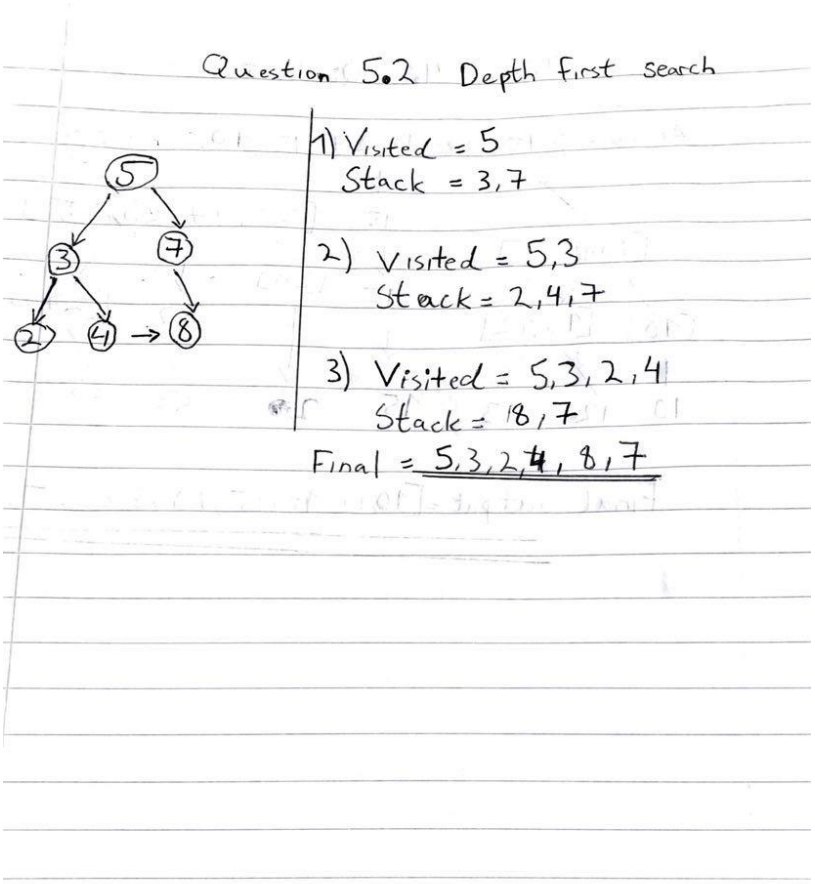
See code

Question 5.2:

Depth First Search Algorithm:

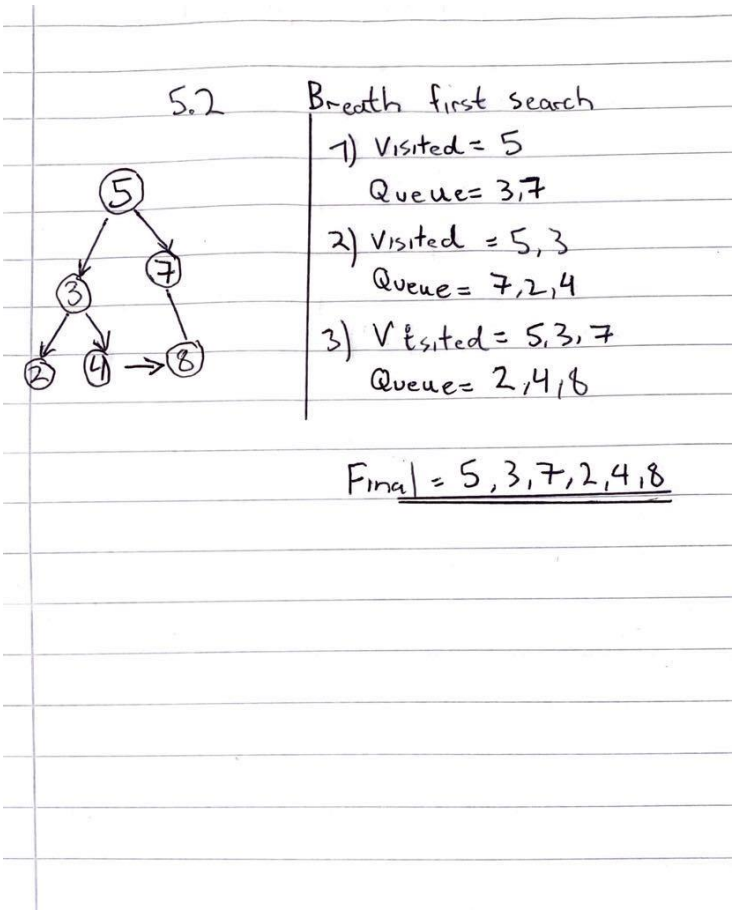
1) We first by putting the vertices from graph on top of the stack.
2) We take out the first element in the stack and put it to the visited list.
3) We put the non-visited elements on the top of the stack.
4) Repeat the steps 2 and 3 until the stack gets empty.

5) After the operation is finished you should have the final output = 5,3,2,4,8,7.



Breath First Search Algorithm:

1) We first start by putting the vertices on the rear end of a queue
2) Secondly, we take the front element and add it to the visited list.
3) We create a list for non-visited vertices/ elements and add them to the rear end side in the queue
4) Repeat the steps 2 and 3 until the queue gets empty.
5) After the operation is finished you should have the final output = 5,3,7,2,4,8.



Question 6:

6.1)

Complexity classes is the classes that describes the efficiency of different algorithms. In other words, it can analyze the time complexity (how muck time) and how much space (space complexity is required) to solve the problem. Problems for example that are unsolved can be divided into complexity classes, examples of classes are P, NP, CO-NP and NP-hard classes.

6.2)

Big O Notation is a mathematical expression that is used to analyze the time and space complexity of an algorithm to classify which algorithm is the best. it is also used to find the efficiency of different algorithm. The notation is also used to analyze the performance in the algorithm as the input grows.

6.3)

P – Class is a type of a Complexity Class and stands for Polynomial class, the solutions to P problems can be found easily the answers to this type of problems has either “yes” or “no” answers. P problems are often tractable (can be solved in practice and theory) and solvable. real life example of P problems is merge Sort.

6.4)

NP – Class stands for Non - deterministic Polynomial Time Class is a set of problems that is solvable in non-deterministic polynomial time. Solutions to this type of class is easy to verify, but hard to find. Example of a NP problem is Tower of Hanoi puzzle

6.5)

NP Complete Class is a combination of both NP and NP hard, NP complete solutions is very difficult to find in the NP class. If a problem is NP and NP hard it is NP- complete. Since he NP complete can be solvable in polynomial time, then it is also possible to solve any problem in polynomial time of NP an example is Traveling salesman problem.