

DATA STRUCTURES AND ALGORITHMS

ASSIGNMENT

SEPTEMBER 2017

Sydney Twigg
M8C3XRSN8

CONTENTS

| | |
|-------------------------|----|
| QUESTION 1 | 2 |
| QUESTION 1.1.A. | 2 |
| QUESTION 1.1.B.I. | 3 |
| QUESTION 1.1.B.II. | 5 |
| QUESTION 1.2.A. | 6 |
| QUESTION 1.2.B. | 9 |
| QUESTION 2 | 11 |
| QUESTION 2.1.A. | 11 |
| QUESTION 2.1.B. | 14 |
| QUESTION 2.2.A. | 15 |
| QUESTION 2.2.B. | 16 |
| QUESTION 3 | 17 |
| QUESTION 3.1.A. | 17 |
| QUESTION 3.1.B. | 17 |
| QUESTION 3.1.C. | 18 |
| QUESTION 3.2.A. | 19 |
| QUESTION 3.2.B. | 19 |
| QUESTION 3.2.C. | 19 |
| QUESTION 3.2.D. | 19 |
| QUESTION 3.3.A. | 19 |
| QUESTION 3.3.B. | 20 |
| QUESTION 3.3.C. | 20 |
| QUESTION 4 | 21 |
| References | 24 |

QUESTION 1

QUESTION 1.1.A.

Bubble sort is a sorting algorithm that repetitively compares adjacent pairs of elements and swaps them if necessary (theoryapp, 2012). Bubble sort swaps the pairs of elements to be in the correct order until the entire list to be compared is sorted, repeatedly cycling through the list until it is sorted (Indika, 2011). Bubble sort is a simple algorithm, as seen in Figure 1:

```
1 public static void bubbleSort (int[] data)
2 {
3     for (int i = data.length - 1; i >= 0; i--)
4     {
5         // bubble up
6         for (int j = 0; j <= i - 1; j++)
7         {
8             if (data[j] > data[j + 1])
9                 swap(data, j, j + 1);
10        }
11    }
12 }
```

Figure 1 - Bubble Sort algorithm. Source: (theoryapp, 2012).

The time to run bubble sort is:

$$t(n) = n * (n/2) = O(n^2)$$

Where n = the array length (theoryapp, 2012). Due to the average time complexity of the bubble sort - $O(n^2)$ - this is not suitable for lists with many elements (Indika, 2011).

Selection sort is a sorting algorithm that repetitively picks up the smallest element and puts them into the correct position. The process to do this is as follows:

- 1) Find the smallest element in the array and put it in the first position - sortArray[0] - of the array
- 2) Find the next smallest element in the array and put it in the second position - sortArray[1] - of the array
- 3) Repeat until all elements of the array are in the correct order (theoryapp, 2012).

As the selection sort algorithm runs, the sorted items grow from left to right, repeatedly cycling through the array until all items are sorted (Indika, 2011). The selection sort algorithm is more complicated than the bubble sort, as shown in Figure 2:

```
1 public static void selectionSort(int[] arr)
2 {
3     // find the smallest element starting from position i
4     for (int i = 0; i < arr.length - 1; i++)
5     {
6         int min = i; // record the position of the smallest
7         for (int j = i + 1; j < arr.length; j++)
8         {
9             // update min when finding a smaller element
10            if (arr[j] < arr[min])
11                min = j;
12        }
13        // put the smallest element at position i
14        swap(arr, i, min);
15    }
16 }
17 public static void swap (int[] arr, int i, int j)
18 {
19     int temp = arr[i];
20     arr[i] = arr[j];
21     arr[j] = temp;
22 }
23 }
```

Figure 2 - Selection Sort algorithm. Source: (theoryapp, 2012).

However, selection sort has the same average time complexity as bubble sort - $O(n^2)$ - selection sort is more efficient as bubble sort needs to perform more swaps (Indika, 2011). Selection sort is not suited for lists with many elements. The time to run selection sort is:

$$t(n) = (n - 1) * (n/2) * k = O(n^2)$$

Where n = the number of elements in the array, and k = the number of basic operations inside each inner loop (theoryapp, 2012).

Comparatively, bubble sort is a simpler, smaller, stable algorithm while selection sort is larger, and unstable - however selection sort outperforms bubble sort in requiring less swaps to sort the array (Indika, 2011).

QUESTION 1.1.B.I.

Bubble Sort:

array[] = {5, 1, 6, 2, 4, 3}

Loop 1:

Compare the first two values, array[0] and array[1], and swap them around if array[1] is smaller than array[0]:

{1, 5, 6, 2, 4, 3}

Loop 2:

Compare the next two values, array[1] and array[2], and swap them around if array[2] is smaller than array[1]:

{1, 5, 6, 2, 4, 3}

Loop 3:

Compare the next two values, array[2] and array[3], and swap them around if array[3] is smaller than array[2]:

{1, 5, 2, 6, 4, 3}

Loop 4:

Compare the next two values, array[3] and array[4], and swap them around if array[4] is smaller than array[3]:

{1, 5, 2, 4, 6, 3}

Loop 5:

Compare the next two values, array[4] and array[5], and swap them around if array[5] is smaller than array[4]:

{1, 5, 2, 4, 3, 6}

Loop 6:

Start from array[0] again as the array is not yet sorted. Compare the values of array[0] and array[1], and swap them around if array[1] is smaller than array[0]:

{1, 5, 2, 4, 3, 6}

Loop 7:

Compare the next two values, array[1] and array[2], and swap them around if array[2] is smaller than array[1]:

{1, 2, 5, 4, 3, 6}

Loop 8:

Compare the next two values, array[2] and array[3], and swap them around if array[3] is smaller than array[2]:

{1, 2, 4, 5, 3, 6}

Loop 9:

Compare the next two values, array[3] and array[4], and swap them around if array[4] is smaller than array[3]:

{1, 2, 4, 3, 5, 6}

Loop 10:

Compare the next two values, array[4] and array[5], and swap them around if array[5] is smaller than array[4]:

{1, 2, 4, 3, 5, 6}

Loop 11:

Start from array[0] again as the array is not yet sorted. Compare the values of array[0] and array[1], and swap them around if array[1] is smaller than array[0]:

{1, 2, 4, 3, 5, 6}

Loop 12:

Compare the next two values, array[1] and array[2], and swap them around if array[2] is smaller than array[1]:

{1, 2, 4, 3, 5, 6}

Loop 13:

Compare the next two values, array[2] and array[3], and swap them around if array[3] is smaller than array[2]:

{1, 2, 3, 4, 5, 6}

The array is now sorted.

Summary of Bubble Sort:

| Bubble Sort | | | | | | | |
|-------------|--------------------|-----------|-----------|-----------|-----------|-----------|-----------|
| Loop | Array | Element 0 | Element 1 | Element 2 | Element 3 | Element 4 | Element 5 |
| 0 | {5, 1, 6, 2, 4, 3} | 5 | 1 | 6 | 2 | 4 | 3 |
| 1 | {1, 5, 6, 2, 4, 3} | 1 | 5 | 6 | 2 | 4 | 3 |
| 2 | {1, 5, 6, 2, 4, 3} | 1 | 5 | 6 | 2 | 4 | 3 |
| 3 | {1, 5, 2, 6, 4, 3} | 1 | 5 | 2 | 6 | 4 | 3 |
| 4 | {1, 5, 2, 4, 6, 3} | 1 | 5 | 2 | 4 | 6 | 3 |
| 5 | {1, 5, 2, 4, 3, 6} | 1 | 5 | 2 | 4 | 3 | 6 |
| 6 | {1, 5, 2, 4, 3, 6} | 1 | 5 | 2 | 4 | 3 | 6 |
| 7 | {1, 2, 5, 4, 3, 6} | 1 | 2 | 5 | 4 | 3 | 6 |
| 8 | {1, 2, 4, 5, 3, 6} | 1 | 2 | 4 | 5 | 3 | 6 |
| 9 | {1, 2, 4, 3, 5, 6} | 1 | 2 | 4 | 3 | 5 | 6 |
| 10 | {1, 2, 4, 3, 5, 6} | 1 | 2 | 4 | 3 | 5 | 6 |
| 11 | {1, 2, 4, 3, 5, 6} | 1 | 2 | 4 | 3 | 5 | 6 |
| 12 | {1, 2, 4, 3, 5, 6} | 1 | 2 | 4 | 3 | 5 | 6 |
| 13 | {1, 2, 3, 4, 5, 6} | 1 | 2 | 3 | 4 | 5 | 6 |

QUESTION 1.1.B.II.

array[] = {5, 1, 6, 2, 4, 3}

Loop 1:

{1, 5, 6, 2, 4, 3}

Loop 2:

{1, 2, 6, 5, 4, 3}

Loop 3:

{1, 2, 5, 6, 4, 3}

Loop 4:

{1, 2, 5, 4, 6, 3}

Loop 5:

{1, 2, 5, 4, 3, 6}

Loop 6:

{1, 2, 5, 4, 3, 6}

Loop 7:

{1, 2, 5, 4, 3, 6}

Loop 8:

{1, 2, 5, 4, 3, 6}

Loop 9:

{1, 2, 5, 4, 3, 6}

Loop 10:

{1, 2, 4, 3, 5, 6}

Loop 11:

{1, 2, 4, 3, 5, 6}

Loop 12:

{1, 2, 4, 3, 5, 6}

Loop 13:

{1, 2, 4, 3, 5, 6}

Loop 14:

{1, 2, 4, 3, 5, 6}

Loop 15:

{1, 2, 3, 4, 5, 6}

The array is now sorted.

Summary of Selection Sort:

| Selection Sort | | | | | | | |
|----------------|--------------------|-----------|-----------|-----------|-----------|-----------|-----------|
| Loop | Array | Element 0 | Element 1 | Element 2 | Element 3 | Element 4 | Element 5 |
| 0 | {5, 1, 6, 2, 4, 3} | 5 | 1 | 6 | 2 | 4 | 3 |
| 1 | {1, 5, 6, 2, 4, 3} | 1 | 5 | 6 | 2 | 4 | 3 |
| 2 | {1, 2, 6, 5, 4, 3} | 1 | 2 | 6 | 5 | 4 | 3 |
| 3 | {1, 2, 5, 6, 4, 3} | 1 | 2 | 5 | 6 | 4 | 3 |
| 4 | {1, 2, 5, 4, 6, 3} | 1 | 2 | 5 | 4 | 6 | 3 |
| 5 | {1, 2, 5, 4, 3, 6} | 1 | 2 | 5 | 4 | 3 | 6 |
| 6 | {1, 2, 5, 4, 3, 6} | 1 | 2 | 5 | 4 | 3 | 6 |
| 7 | {1, 2, 5, 4, 3, 6} | 1 | 2 | 5 | 4 | 3 | 6 |
| 8 | {1, 2, 5, 4, 3, 6} | 1 | 2 | 5 | 4 | 3 | 6 |
| 9 | {1, 2, 4, 5, 3, 6} | 1 | 2 | 4 | 5 | 3 | 6 |
| 10 | {1, 2, 4, 3, 5, 6} | 1 | 2 | 4 | 3 | 5 | 6 |
| 11 | {1, 2, 4, 3, 5, 6} | 1 | 2 | 4 | 3 | 5 | 6 |
| 12 | {1, 2, 4, 3, 5, 6} | 1 | 2 | 4 | 3 | 5 | 6 |
| 13 | {1, 2, 4, 3, 5, 6} | 1 | 2 | 4 | 3 | 5 | 6 |
| 14 | {1, 2, 4, 3, 5, 6} | 1 | 2 | 4 | 3 | 5 | 6 |
| 15 | {1, 2, 3, 4, 5, 6} | 1 | 2 | 3 | 4 | 5 | 6 |

QUESTION 1.2.A.

AVL TREE

{5, 1, 6, 2, 4, 3}

Balance factor = height of left subtree - height of right subtree

Root node:

Level 0: insert 5, height = 1



Insert 1:

Level 0: 1 < 5 ∴ left, height = 2

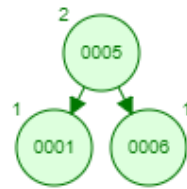
Level 1: insert 1, height = 1



Insert 6:

Level 0: $6 \geq 5 \therefore \text{right, height} = 2$

Level 1: insert 6, height = 1

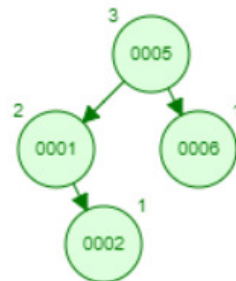


Insert 2:

Level 0: $2 < 5 \therefore \text{left, height} = 3$

Level 1: $2 \geq 1 \therefore \text{right, height} = 2$

Level 2: insert 2, height = 1



Insert 4

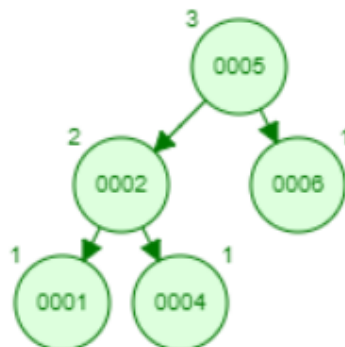
Level 0: $4 < 5 \therefore \text{left}$

Level 1: $4 \geq 1 \therefore \text{right}$

Level 2: $4 \geq 2 \therefore \text{right}$

Level 3: insert 4

To balance AVL tree: $2 > 1 \therefore \text{single rotation left}$



Insert 3

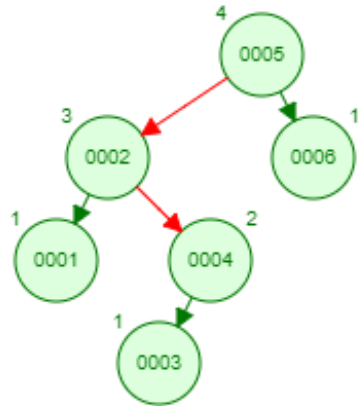
Level 0: $3 < 5 \therefore$ left

Level 1: $3 \geq 2 \therefore$ right

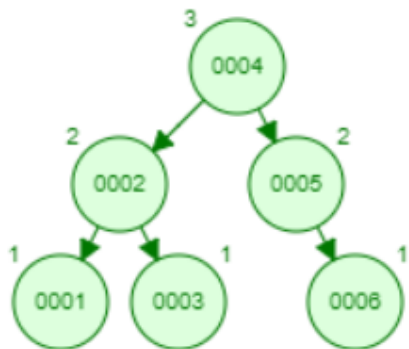
Level 2: $3 < 4 \therefore$ left

Level 3: insert 3

To balance AVL tree: perform a double right rotation



Perform a double right rotation to balance the AVL tree



Balanced AVL tree

QUESTION 1.2.B.

BINARY SEARCH TREE

{5, 1, 6, 2, 4, 3}

Root node:

Level 0: insert 5



Insert 1:

1 is less than 5, therefore will be placed on the left subtree

Level 0: $1 < 5 \therefore \text{left}$

Level 1: insert 1



Insert 6:

6 is greater than 5, therefore it will be placed on the right subtree

Level 0: $6 \geq 5 \therefore \text{right}$

Level 1: insert 6



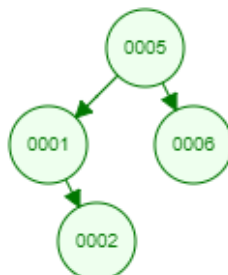
Insert 2:

2 is less than 5, therefore will be placed on the left subtree; 2 is greater than 1, therefore will be placed on the right subtree of 1.

Level 0: $2 < 5 \therefore \text{left}$

Level 1: $2 \geq 1 \therefore \text{right}$

Level 2: insert 2



Insert 4:

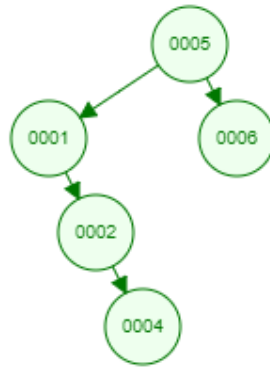
4 is less than 5, therefore will be placed on the left subtree; 4 is greater than 1, therefore will be placed on the right subtree of 1; 4 is greater than 2, therefore will be placed on the right subtree of 2.

Level 0: $4 < 5 \therefore \text{left}$

Level 1: $4 \geq 1 \therefore \text{right}$

Level 2: $4 \geq 2 \therefore \text{right}$

Level 3: insert 4

**Insert 3:**

3 is less than 5, therefore will be placed on the left subtree; 3 is greater than 1, therefore will be placed on the right subtree of 1; 3 is greater than 2, therefore will be placed on the right subtree of 2; 3 is less than 4, therefore will be placed on the left subtree of 4.

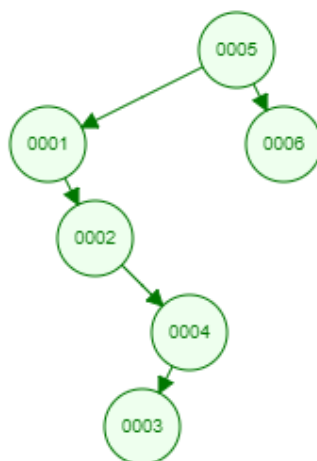
Level 0: $3 < 5 \therefore \text{left}$

Level 1: $3 \geq 1 \therefore \text{right}$

Level 2: $3 \geq 2 \therefore \text{right}$

Level 3: $3 < 4 \therefore \text{left}$

Level 4: insert 3



QUESTION 2

QUESTION 2.1.A

```
1. //Sydney Twigg
2. //M8C3XRSN8
3. //ITDS221 Assignment Question 2.1
4. package question2;
5.
6. public class Client {
7.     private String name, contactDetails;
8.
9.     public Client(String name, String contactDetails) {
10.         this.name = name;
11.         this.contactDetails = contactDetails;
12.     }
13.
14.     //getter
15.     public String getName() {
16.         return name;
17.     }
18.
19.     //getter
20.     public String getContactDetails() {
21.         return contactDetails;
22.     }
23.
24.     //toString class to display
25.     public String toString() {
26.         return "Name: " + getName() + " , Contact Details: " + getContactDetails();
27.     }
28. }
```

```
1. //Sydney Twigg
2. //M8C3XRSN8
3. //ITDS221 Assignment Question 2.1
4. package question2;
5.
6. public class Queue {
7.     private int front, rear, count, size;
8.     private Client[] data;
9.
10.    //create empty queue
11.    public Queue(int n) {
12.        data = new Client[n];
13.        front = 0;
14.        rear = 0;
15.        count = 0;
16.    }
17.
18.    //test whether or not the queue is empty
19.    public boolean isEmpty() {
20.        return (count == 0);
21.    }
22.
23.    //test whether or not the queue is full
24.    public boolean isFull() {
25.        return (count == data.length);
26.    }
27.
28.    //add item - client - to the end of the queue
29.    public void enqueue(Client val) {
30.        data[rear] = val;
31.        rear = (rear + 1) % data.length;
32.        count++;
33.    }
34.
35.
36. }
```

```

37. //remove item - client - from the front of the queue
38. public Client dequeue() {
39.     Client value = data[front + 1];
40.     front = (front + 1) % data.length;
41.     count--;
42.     return value;
43. }
44.
45. //returns the client in the front of the queue
46. public Client front() {
47.     return data[front];
48. }
49.
50. //returns the size of the queue
51. public int size() {
52.     return size;
53. }
54.
55. //display the queue
56. public void print() {
57.     for (int i = 0; i < data.length; i++) {
58.         if (i == front) {
59.             System.out.println(data[i] + "(f), ");
60.         } else if (i == rear) {
61.             System.out.println(data[i] + "(r), ");
62.         } else {
63.             System.out.println(data[i] + ", ");
64.         }
65.     }
66. }
67. }

```

```

1. //Sydney Twigg
2. //M8C3XRSN8
3. //ITDS221 Assignment Question 2.1
4.
5. package question2;
6. import java.util.Scanner;
7.
8. public class QueueOfClients {
9.
10.     public static void main(String[] args) {
11.         //set queue size to 10, as per the client limit per day
12.         QueueOfClients queue = new QueueOfClients();
13.
14.         //call test class
15.         queue.testQueue();
16.     }
17.
18.     public void testQueue() {
19.         //create an object of stack class
20.         Queue queue = new Queue(10);
21.         Scanner sc = new Scanner(System.in);
22.         //initialise variables
23.         char ch;
24.
25.         String clientName, contactDetails;
26.         //create menu to test
27.         do {
28.             System.out.println("\nStack Operations");
29.             System.out.println("1. enqueue");
30.             System.out.println("2. dequeue");
31.             System.out.println("3. size");
32.             System.out.println("4. view front");
33.             System.out.println("5. check full");
34.             System.out.println("6. check empty");
35.             System.out.println("7. view queue");
36.             int choice = sc.nextInt();
37.

```

```

38.         switch (choice) {
39.             case 1: //enqueue
40.                 try {
41.                     System.out.println("Enter client name: ");
42.                     clientName = sc.next();
43.                     System.out.println("Enter contact details: ");
44.                     contactDetails = sc.next();
45.                     queue.enqueue(new Client(clientName, contactDetails));
46.                 } catch (Exception e) {
47.                     System.out.println("Error: " + e.getMessage());
48.                 }
49.                 break;
50.             case 2: //dequeue
51.                 try {
52.                     System.out.println("Served client: " + queue.dequeue());
53.                 } catch (Exception e) {
54.                     System.out.println("Error: " + e.getMessage());
55.                 }
56.                 break;
57.             case 3: //size
58.                 System.out.println("Client's booked for today: " + queue.size());
59.                 break;
60.             case 4: //front
61.                 try {
62.                     System.out.println("Next client booking: " + queue.front());
63.                 } catch (Exception e) {
64.                     System.out.println("Error: " + e.getMessage());
65.                 }
66.                 break;
67.             case 5: //isFull
68.                 System.out.println("Full status: " + queue.isFull());
69.                 break;
70.             case 6: //isEmpty
71.                 System.out.println("Empty status: " + queue.isEmpty());
72.                 break;
73.             case 7: //view queue
74.                 System.out.println("Salon Queue:");
75.                 queue.print();
76.             default:
77.                 System.out.println("Invalid entry");
78.                 break;
79.         }
80.         System.out.println("Do you want to continue (Type y or n) \n");
81.         ch = sc.next().charAt(0);
82.     } while (ch == 'Y' || ch == 'y');
83. }
84. }

```

QUESTION 2.1.B.

A queue has the following operations:

Queue

Queue is an operation that creates an empty queue; this is related to the scenario in that a new queue is created for each day's salon bookings (Leung, 2017).

Enqueue

Enqueue is an operation that adds an item to the end of a queue; this is related to the scenario in that each new client who books an appointment is added to the end of the booking queue, in the order that they book - first come, first serve (Leung, 2017).

Dequeue

Dequeue is an operation that removes an item from the front of a queue; this is related to the scenario in that as each client is served by the salon, they will be removed from the queue as they have received their services (Leung, 2017).

isFull

isFull is an operation that tests whether the queue is full; this is related to the scenario in that a client cannot book an appointment if the queue is full (Leung, 2017).

isEmpty

isEmpty is an operation that tests whether the queue is empty; this is related to the scenario in that each day needs to begin with an empty queue for clients to book services (Leung, 2017).

Size

Size is an operation that returns the number of items that are in a queue; this is related to the scenario in that Mrs Mosuhli needs to see the amount of people who have booked appointments per day in order to prepare for that day (Leung, 2017).

Front

Front is an operation that returns the item at the front of the queue without removing it; this is related to the scenario in that the salon staff can check who is to be served next without having to remove the client from the queue (Leung, 2017).

QUESTION 2.2.A.

The following table contrasts and compares stack and queue operations:

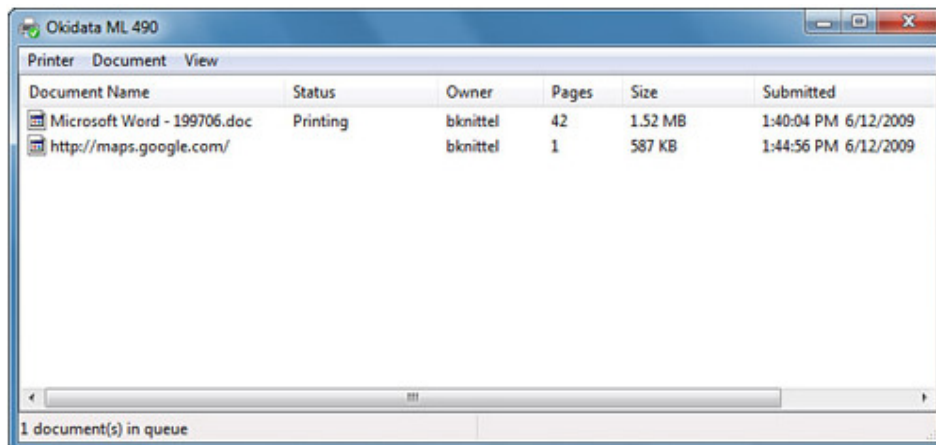
| Operation | Stack | Queue | Conclusion |
|--------------------------------------|--|--|--|
| Create | Stack() <ul style="list-style-type: none"> Create a new empty stack Needs no parameters Returns an empty stack | Queue() <ul style="list-style-type: none"> Create a new empty queue Needs no parameters Returns an empty queue | Operations are similar |
| Is empty | isEmpty() <ul style="list-style-type: none"> Checks whether a stack is empty Needs no parameters Returns Boolean | isEmpty() <ul style="list-style-type: none"> Checks whether a queue is empty Needs no parameters Returns Boolean | Operations are similar |
| Is full | isFull() <ul style="list-style-type: none"> Check whether a stack is full Needs no parameters Returns Boolean | isFull() <ul style="list-style-type: none"> Checks whether a queue is full Needs no parameters Returns Boolean | Operations are similar |
| Add new item | push(item) <ul style="list-style-type: none"> Adds a new item to the top of the stack Item is the parameter Returns nothing | enqueue(item) <ul style="list-style-type: none"> Adds a new item to the end of a queue Item is the parameter Returns nothing | Operations are not similar - a stack adds to the top of the list, while a queue adds to the back of a list |
| Remove item | pop() <ul style="list-style-type: none"> Remove the top item of a stack Needs no parameters Returns the top item Modifies stack | dequeue() <ul style="list-style-type: none"> Removes the front item of the queue Needs no parameters Returns the front item Modifies queue | Operations are similar |
| Show most recently added item | peek() <ul style="list-style-type: none"> Return the top item of a stack without removing it Needs no parameters Returns the top item Does not modify stack | front() <ul style="list-style-type: none"> Returns the front item of the queue without removing it Needs no parameters Returns the front item Does not modify queue | Operations are similar |
| Size | size() <ul style="list-style-type: none"> Return the number of items in the stack Needs no parameters Returns integer | size() <ul style="list-style-type: none"> Return the number of items in the queue Needs no parameters Returns integer | Operations are similar |

References: (Sedgewick & Wayne, 2017) (Leung, 2017) (Leung, 2017).

QUESTION 2.2.B.

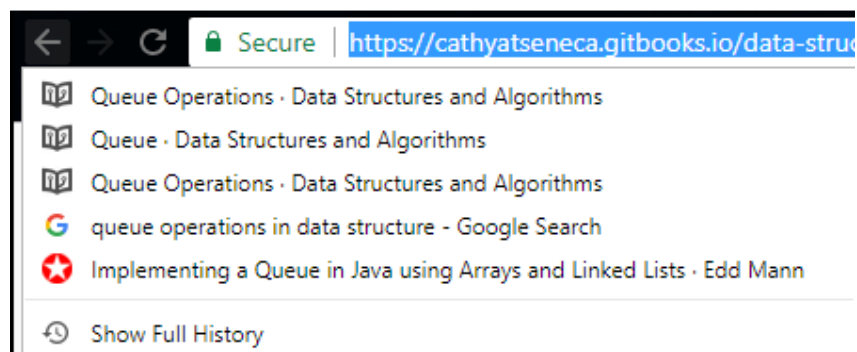
An example of a queue in a computer environment is a print queue; each document that is sent to the printer joins a print queue and waits for the items sent before it to finish printing until it can print.

The following screenshot shows a print queue in Windows:



Source: (Lasertek Services)

An example of a stack in a computer environment is a browser's back button; as you switch from webpage to webpage, each website's URL is placed on a stack, from oldest to most recent webpage that you have viewed (Miller & Ranum, 2017).



Screenshot showing a stack of previously viewed webpages in a browser

QUESTION 3

QUESTION 3.1.A.

Recursion is a problem-solving method in which the solution to a problem relies on the solutions to smaller instances of the same problem, a recursive function therefore calls itself to solve a given problem (Legodi & Shangirai, 2012, p. 7). To prevent a recursive function looping infinitely, there are fundamental properties of recursion, these are: base criteria and progressive approach. Base criteria means that there must always be one base criteria or condition that stops the function calling itself once the condition is met; progressive approach means that each recursive call made should bring the method closer to the base criteria (TutorialsPoint, 2017).

QUESTION 3.1.B.

The following algorithm shows a recursive method of the factorial method:

```
1 //Sydney Twigg
2 //M8C3XRSN8
3 //ITDS221 Assignment Question 3.1.b.
4
5 /* Factorial method:
6 Formula: factorial(n) = n * factorial(n-1)
7 */
8
9 private static long factorial(int n){
10     if (n == 1){
11         return 1;
12     }
13     else{
14         return n * factorial(n-1);
15     }
16 }
```

Screenshot showing a recursive method

QUESTION 3.1.C.

The value of magic(2, 25) is 50.

Calculations:

| Call | a | b | if (b==0) | if (b%2 ==0) | Return | Return Value: |
|------|----|----|-----------|--------------|-------------------|---------------|
| 1 | 2 | 25 | FALSE | FALSE | (2+2, 25/2) + 2 | 50 |
| 2 | 4 | 12 | FALSE | TRUE | (4+4, 12/2) | 48 |
| 3 | 8 | 6 | FALSE | TRUE | (8+8, 6/2) | 48 |
| 4 | 16 | 3 | FALSE | FALSE | (16+16, 3/2) + 16 | 48 |
| 5 | 32 | 1 | FALSE | FALSE | (32+32, 1/2) + 32 | 32 |
| 6 | 64 | 0 | TRUE | TRUE | 0 | 0 |

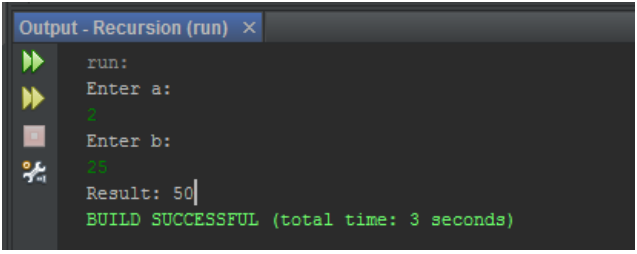
The value of magic(3, 11) is 33.

Calculations:

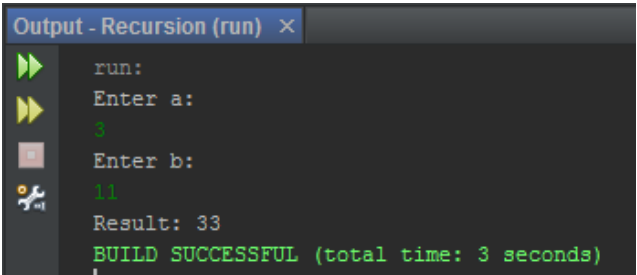
| Call | a | b | if (b==0) | if (b%2 ==0) | Return | Return Value: |
|------|----|----|-----------|--------------|-------------------|---------------|
| 1 | 3 | 11 | FALSE | FALSE | (3+3, 11/2) + 3 | 33 |
| 2 | 6 | 5 | FALSE | FALSE | (6+6, 5/2) + 6 | 30 |
| 3 | 12 | 2 | FALSE | TRUE | (12+12, 2/2) + 12 | 24 |
| 4 | 24 | 1 | FALSE | FALSE | (24+24, 1/2) + 24 | 24 |
| 5 | 48 | 0 | TRUE | TRUE | 0 | 0 |

The following java code shows this being tested:

```
1 //Sydney Twigg
2 //M8C3XRSN8
3 //ITDS221 Assignment Question 3.1.c.
4 package recursion;
5
6 import java.util.Scanner;
7
8 public class Recursion {
9
10     public static void main(String[] args) {
11         Scanner sc = new Scanner(System.in);
12
13         System.out.println("Enter a: ");
14         int a = sc.nextInt();
15
16         System.out.println("Enter b: ");
17         int b = sc.nextInt();
18
19         System.out.println("Result: " + magic(a, b));
20     }
21
22     public static int magic(int a, int b){
23         if (b==0){
24             return 0;
25         }
26         else if (b%2 == 0){
27             return magic(a+a, b/2);
28         }
29         else{
30             return magic(a+a, b/2) + a;
31         }
32     }
33 }
```



```
Output - Recursion (run) X
run:
Enter a:
2
Enter b:
25
Result: 50
BUILD SUCCESSFUL (total time: 3 seconds)
```



```
Output - Recursion (run) X
run:
Enter a:
3
Enter b:
11
Result: 33
BUILD SUCCESSFUL (total time: 3 seconds)
```

Java code and output to test the magic() method

QUESTION 3.2.A.

$O(n)$

This is the time complexity algorithm for **linear time**, this means that the running time - particularly with large input values of n - of the algorithm increases linearly with the size of the input (Adamchik, 2009). This can be used for algorithms such as finding the smallest or largest item in an unsorted array.

QUESTION 3.2.B.

$O(n^2)$

This is the time complexity algorithm for **quadratic time**, this means that the running time is proportional to the square of its input size (Adamchik, 2009). This can be used for algorithms such as bubble sort, insertion sort, and selection sort.

QUESTION 3.2.C.

$O(n^3)$

This is the time complexity algorithm for **cubic time**, this means that the running time is proportional to the cube of its input size (Tel, 2011). This is used for algorithms such as partial correlation.

QUESTION 3.2.D.

$O(b^n)$

This is the time complexity algorithm for **exponential time**, this means that the running time increases exponentially proportional to its input size, exponential algorithms run slower as the input size increases, thus they are only useful for very small input sizes (Tel, 2011). Exponential time algorithms are used for brute-force search algorithms.

QUESTION 3.3.A.

```
1. for (int number: arrayOfNumbers) {  
2.     System.out.println(number);  
3. }
```

The above statement within the loop is $O(1)$.

The above loop executes N times, therefore the total time of the loop is $O(1) * N = O(N)$.

```
1. for (int firstNumber: arrayOfNumbers) {  
2.     for (int secondNumber: arrayOfNumbers) {  
3.         System.out.println(firstNumber + secondNumber);  
4.     }  
5. }
```

The above statement within the loop is $O(1)$.

The above outer loop executes N times, each time the outer loop executes, the inner loop will execute N times, therefore the total time of the loop is $O(1) * N^2 = O(N^2)$.

The runtime of `printAllNumbersThenAllPairSums(int[] arrayOfNumbers)` should be $O(N + N^2)$, but due to the nature of polynomial Big O calculations, you will only keep the highest order term, $O(N^2)$ (Cantos, 2011).

Therefore, the running time of the algorithm is $O(N^2)$.

QUESTION 3.3.B.

```
1. for (int i = 0; i < n; i++) {  
2.     System.out.println("hi");  
3. }
```

The above statement within the loop is $O(1)$.

The above loop executes N times, therefore the total running time of loop is $O(1) * N = O(N)$.

Therefore, the running time of the algorithm is $O(N)$.

QUESTION 3.3.C.

```
1. for (int i = 0; i < n; i++) {  
2.     hiArray[i] = "hi";  
3. }
```

The above statement within the loop is $O(1)$.

The above loop executes N times, therefore the total time of loop is $O(1) * N = O(N)$.

Therefore, the running time of the algorithm is $O(N)$.

Reference used for all calculations: (Vernon, 2005).

QUESTION 4

```
1. //Sydney Twigg
2. //M8C3XRSN8
3. //ITDS221 Assignment Question 4
4. package question4;
5.
6. //class to create car instance of each car
7. public class Car {
8.     private String carModel;
9.     private String plateNum;
10.    public Car(String model, String plate) {
11.        this.carModel = model;
12.        this.plateNum = plate;
13.    }
14.
15.    //getter
16.    public String getCarModel() {
17.        return carModel;
18.    }
19.
20.    //setter
21.    public void setCarModel(String carModel) {
22.        this.carModel = carModel;
23.    }
24.
25.    //getter
26.    public String getPlateNum() {
27.        return plateNum;
28.    }
29.
30.    //setter
31.    public void setPlateNum(String plateNum) {
32.        this.plateNum = plateNum;
33.    }
34.
35.    //toString method to display
36.    @Override
37.    public String toString() {
38.        return "Car Model: " + carModel + ", Plate Number: " + plateNum;
39.    }
40. }
```

```
1. //Sydney Twigg
2. //M8C3XRSN8
3. //ITDS221 Assignment Question 4
4. package question4;
5.
6. public class Stack {
7.     private int size;
8.     private Car[] stackArray;
9.     private int top;
10.
11.    //create empty stack
12.    public Stack(int size) {
13.        size = size;
14.        stackArray = new Car[size];
15.        top = -1;
16.    }
17.
18.    //add item - car - to the top of the stack
19.    public void push(Car c) {
20.        stackArray[++top] = c;
21.    }
22.
23.    //remove the top item from the stack
24.    public Car pop() {
25.        return stackArray[top--];
26.    } /
```

```

27. //View the top item of the stack
28.     public Car peek() {
29.         return stackArray[top];
30.     }
31.
32. //Return the size of the stack
33.     public int size() {
34.         return size;
35.     }
36.
37. //test if the stack is empty
38.     public boolean isEmpty() {
39.         return top == -1;
40.     }
41. }

```

```

1. //Sydney Twigg
2. //M8C3XRSN8
3. //ITDS221 Assignment Question 4
4. package question4;
5.
6. import java.util.Scanner;
7.
8. public class StackOfCars {
9.     public static void main(String[] args) {
10.         StackOfCars stack = new StackOfCars();
11.         stack.testStack();
12.     }
13.
14.     public void testStack() {
15.         //create an object of stack class
16.         Stack stack = new Stack(6);
17.         Scanner sc = new Scanner(System.in);
18.
19.         //initialise variables
20.         char ch;
21.         String model, plateNum;
22.
23.         //create menu to test
24.         do {
25.             System.out.println("\nStack Operations");
26.             System.out.println("1. push");
27.             System.out.println("2. pop");
28.             System.out.println("3. size");
29.             System.out.println("4. peek");
30.             int choice = sc.nextInt();
31.             switch (choice) {
32.                 case 1:
33.                     try {
34.                         System.out.println("Enter model: ");
35.                         model = sc.next();
36.                         System.out.println("Enter plate number: ");
37.                         plateNum = sc.next();
38.                         stack.push(new Car(model, plateNum));
39.                     } catch (Exception e) {
40.                         System.out.println("Error: " + e.getMessage());
41.                     }
42.                     break;
43.                 case 2:
44.                     try {
45.                         System.out.println("Removed: " + stack.pop());
46.                     } catch (Exception e) {
47.                         System.out.println("Error: " + e.getMessage());
48.                     }
49.                     break;
50.                 case 3:
51.                     System.out.println("Cars available: " + stack.size());
52.                     break;
53.

```

```
54.         case 4:
55.             try {
56.                 System.out.println("Next car available: " + stack.peek());
57.             } catch (Exception e) {
58.                 System.out.println("Error: " + e.getMessage());
59.             }
60.             break;
61.         default:
62.             System.out.println("Invalid entry");
63.             break;
64.     }
65.     System.out.println("Do you want to continue (Type y or n) \n");
66.     ch = sc.next().charAt(0);
67.     } while (ch == 'Y' || ch == 'y');
68. }
69. }
```


References

- Adamchik, V. S., 2009. *Algorithmic Complexity*. [Online]
Available at: <https://www.cs.cmu.edu/~adamchik/15-121/lectures/Algorithmic%20Complexity/complexity.html>
[Accessed September 2017].
- BetterExplained, 2007. *Sorting Algorithms*. [Online]
Available at: https://betterexplained.com/articles/sorting-algorithms/#Bubble_Sort_Best_On_WorstON2
[Accessed September 2017].
- Cantos, M., 2011. *Big O, how do you calculate/approximate it?*. [Online]
Available at: <https://stackoverflow.com/questions/3255/big-o-how-do-you-calculate-approximate-it>
[Accessed September 2017].
- Indika, 2011. *Difference Between Bubble Sort and Selection Sort*. [Online]
Available at: <http://www.differencebetween.com/difference-between-bubble-sort-and-vs-selection-sort/>
[Accessed September 2017].
- Lasertek Services, 2017. *How To Clear a Printer Queue*. [Online]
Available at: <http://resources.lasertekservices.com/how-to-clear-a-printer-queue/>
[Accessed September 2017].
- Legodi, M. & Shangirai, R., 2012. *Data Structures and Algorithms*. Johannesburg: CTI Education Group/London School of Business and Management..
- Leung, C., 2017. *Queue Operations*. [Online]
Available at: https://cathyatseneca.gitbooks.io/data-structures-and-algorithms/queue/queue_operations.html
[Accessed September 2017].
- Leung, C., 2017. *Stack Operations*. [Online]
Available at: https://cathyatseneca.gitbooks.io/data-structures-and-algorithms/the_stack/stack_operations.html
[Accessed September 2017].
- Miller, B. & Ranum, D., 2017. *What is a Stack?*. [Online]
Available at: <http://interactivepython.org/runestone/static/pythonds/BasicDS/WhatisaStack.html>
[Accessed September 2017].
- Sedgewick, R. & Wayne, K., 2017. *Stacks and Queues*. [Online]
Available at: <http://introcs.cs.princeton.edu/java/43stack/>
[Accessed September 2017].
- Tel, P., 2011. *Determining The Complexity Of Algorithm (The Basic Part)*. [Online]
Available at: <https://philipstel.wordpress.com/2011/03/07/determining-the-complexity-of-an-algorithm-the-basic-part/>
[Accessed September 2017].
- theoryapp, 2012. *Selection, Insertion and Bubble Sort*. [Online]
Available at: <http://theoryapp.com/selection-insertion-and-bubble-sort/>
[Accessed August 2017].
- TutorialsPoint, 2017. *Data Structure - Recursion Basics*. [Online]
Available at: https://www.tutorialspoint.com/data_structures_algorithms/recursion_basics.htm
[Accessed September 2017].
- Vernon, M. K., 2005. *Complexity and Big-O Notation*. [Online]
Available at: <http://pages.cs.wisc.edu/~vernon/cs367/notes/3.COMPLEXITY.html>
[Accessed September 2017].