2. Describe the Array Implementation of Queue with one example. You need to provide a sample data and walk through the enqueue and dequeue, and other operations as necessary and manage the head and tail pointers. Note: I have already provided you with one example.

Ans:pasted-image.tiff

3. Consider String “It is a sunny day today”

a) Generate a binary Huffman Tree

b) Show binary data both before and after compression. Analyze difference.

c) Consider Java code:

https://www.geeksforgeeks.org/huffman-coding-greedy-algo-3/

Write Pseudo-Code for the Huffman algorithm, SHOW as to why/how

the algorithm uses PriorityQueue

d) Compile Java code and run it with the input string provided above.

Ans:

pasted-image.tiff

Code Before Huffman Encoding:

100100100010111000000100100101101100111000000100100001100000010011001

110101011100111011001110110100111100000010000100110100001101001111000

0001000010111011110110001001101000011010011110

Code After Huffman Encoding:

111100000111111111001100011110110110010011010111001001010100011010110

0100101

Before Encoding: 184 bits

After Encoding: 76 bits

Pseudo-Code:

Begin

Create a node for each character and add to priority queue

While There is more than 1 node in the priority queue

Remove two nodes with least frequency from the priority queue

Create a Node with two node fetched as children and frequency of new node equals sum of two children node frequency

Add new Node to the priority queue

End While

Last Remaining node is the Root Node

End

Why Priority Queue:

We make use of priority queue here to add nodes with different frequency levels and

later remove them based on their frequency and priority. In this case, lowest frequency

node has the highest priority.

4. Write a Java program that generates random text string with a length of 500 bytes for 200,000 iterations. For each iteration, reverse the string using: a) String operations, and b) StringBuilder operations. Then c) What is the running time complexity of (a) and (b) after all iterations? d) Present your results in (c) as a graph showing the running times.

Ans: The time complexity for String is **O(n^3)** because we are creating new instance of the String in Java.

The time complexity of String Builder is **O(n^2)** because we are looping through the string, Java copies the string in each loop to create instance of the sentence.

5. Consider the following code for User class.

A) Discuss code in details

B) Write Java code to test User class with multiple test cases to test quals,

hashCode and CompareTo methods.

public class User implements Comparable<User> {

private String name;

private int id;

private Date birth;

public User (String name, int id, Date birth)

{ this.name = name; this.id = id; this.birth = birth; }

@Override

public boolean equals(Object other) {

if (this == other) return true;

if (other == null || (this.getClass() != other.getClass()))

{ return false; }

User guest = (User) other;

return(this.id == guest.id) &&

(this.name = null && name.equals(guest.name)) &&

(this.dob != null && dob.equals(guest.birth));

}

@Override public int hashCode() {

int result = 0;

result = 31\*result + id;

result = 31\*result + (name !=null ? name.hashCode() : 0);

result = 31\*result + (birth !=null ? dob.hashCode() : 0);

return result;

}

@Override

public int compareTo(User o) {

return this.id - o.id; }

}

}

Ans: This program implements a custom equal, compareTo and hasCode functional.

The equal method compares the reference of the 2 objects. If they have same

references, it returns else it compares the classes of each objects and the

attributes ID, Date of Birth and the name.

The hasCode method generates hash code for each object by considering the

attributes of the class.

The compareTo method compare the ids and returns an integer value based on it.

6. Write the Algorithms for the following Two Queues using Fixed Array Size.

times of best the was it null null null null

0 1 2 3 4 5 6 7 8 9

it was the best of times null null null nu

ll 0 1 2 3 4 5 6 7 8 9

Ans: **Algorithm 1:**

Initialise the array size, tail and head indices

Enqueue

If array not full, tail is not equal to size then

Add element at array[tail]

Increment tail

Dequeue

If array not empty, tail is not equal to head then

Remove element at array[head]

Move all elements to the left

End

**Algorithm 2:**

Initialise the array size, tail and head indices

Enqueue

If array not full rear & it’s size then add element array[tail]

Increment front

On each enqueue move elements to the right

Dequeue

If array not empty, head & get 0

Remove element at array[head]

Decrement head

End

8. B) Write Java code using MinPQ operations.

C) Compile and Run the program with above Input

8. Why Java is Pass-by-Value?

Consider the following two programs,

program-1:

public static void main(String[] args) {

Dog aDog = new Dog(“Bella");

Dog oldDog = aDog;

changeName(aDog);

aDog.getName().equals("Bella"); True/False, why?

aDog.getName().equals("Molly"); True/False, why?

aDog == oldDog; True/False, why?

}

public static void changeName(Dog d) {

d.getName().equals("Bella"); True/False, why?

d = new Dog("Molly");

d.getName().equals("Molly");

}

program-2:

public static void main(String[] args) {

Dog aDog = new Dog("Bella");

Dog oldDog = aDog;

changeName(aDog);

aDog.getName().equals("Molly"); True/False, why?

aDog == oldDog; True/False, why?

}

public static void changeName(Dog d) {

d.getName().equals(“Bella"); True/False, why?

d.setName(“Molly"); }

Ans: Whenever a function call is made in java, for primitive type, function is called by passing variable.

1. aDog.getName().equals(&quot;Bella&quot;); True/False, why?**True.** Because aDog object has name attribute value as Bella.
2. aDog.getName().equals(&quot;Molly&quot;); True/False, why? **False**. Because aDog object has name attribute value as Bella and Not Molly.
3. aDog == oldDog; True/False, why? **True.** Because both variables hold same reference to an object.
4. d.getName().equals(&quot;Bella&quot;); True/False, why? **True.** Because variable “d” is referencing the same object as aDog.
5. aDog.getName().equals(&quot;Molly&quot;); True/False, why? **True.** Because name attribute was changed the function.
6. aDog == oldDog; True/False, why?**True.** Because both variables hold same reference to an object.
7. d.getName().equals(“Bella&quot;); True/False, why? **True.** Because variable “d” is referencing the same object as aDog.