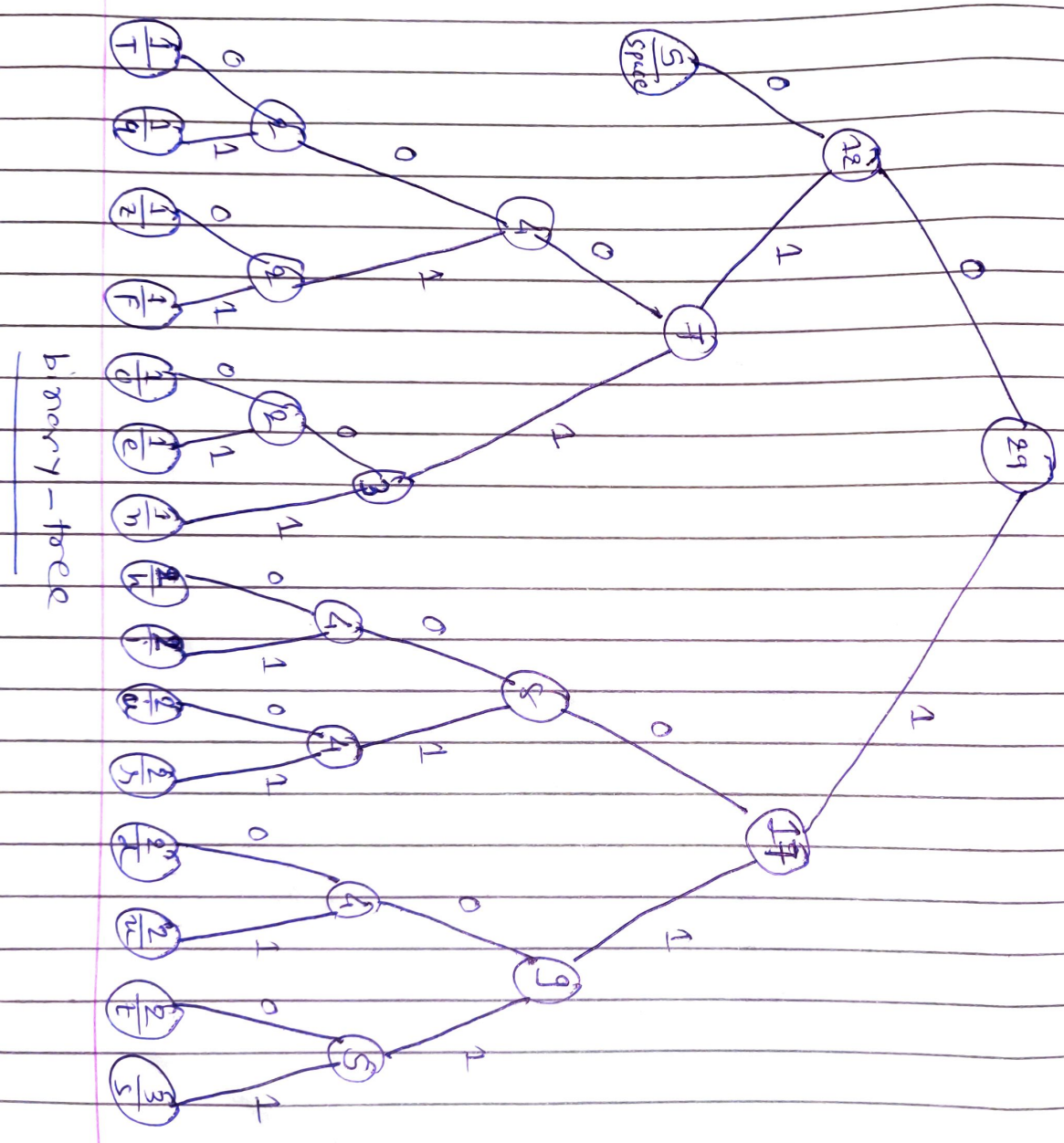
Assignment 2:

Ans: 1 String = "This a hard quiz for students"

(a)

T q z f o e m 2 2
1 1 1 1 1 1 1 1 k i

T q z f o e m h i a r d u t s space
1 1 1 1 1 1 1 2 2 2 2 2 2 2 3 5



(b)

character	count	Before compress	After compression	Binary ~~Before~~ compress
T	1	01010100	01000	0000
q	1	01110001	01001	0001
z	1	01111010	01010	0010
F	1	01100110	01011	0011
O	1	01101111	01100	0100
e	1	01100101	01101	0101
n	1	01101110	01111	0110
h	2	01101000	1000	0111
i	2	01101001	1001	1000
a	2	01100001	1010	1001
r	2	01111010	1011	1010
d	2	01100100	1100	1011
u	2	01110101	1101	1100
t	2	01110100	1110	1101
s	3	01110011	1111	1110
space	5	00100000	00	1111
	29			

Before compression = ~~29 x 8~~ 29 x 4

(binary) = ~~232 bit~~ 116 bit

After compression = 10 + 12 + 5 + 5 + 5 + 5 + 5 + 5 + 5 + 5 + (7 x 5)

= 22 + 56 + 35

= 113 bit

~~Before~~

Before compression = 29 x 8

(ASCII) = 232 bit

(c) pseudo-code:

- Node to store count of character, character and left/right node reference.
- comparator class to compare nodes in priority queue.

- main program
  - char Array to hold string
  - ~~char~~ int Array to hold counts of character in string
  - n = 16 (length of char Array)

  - q = priorityQueue(n, comparator object)

  - for i ⟶ n
    - create new Node
    - node.c = charArray[i]
    - node.data = ~~character~~ intArray[i]
    - ~~root~~
    - node.left = null
    - node.right = null
  - end for
  - priorityQueue.add(node)

  - rootNode = null

  - While q.size() is not equal to 1
    z = new Node()
    z.left = x = q.pop
    z.right = y = q.pop
    z.frequency = x.frequency + y.frequency
    q.push(z)

end while
Return q.

- printCode (root, "") - function to traversing the tree

end main

- printcode function (Node root, String s)

- if (root.left == null and root.right == null)
  return;
  printCode (root.left, s+"0"); } Recursion calling
  printCode (root.right, s+"1");

end printCode

- priority queue is used to store all node which represent each character into String along with occurance count in the String.

- priority queue is used because it is easy to fetch highest priority node, which will add and saved as new node in the same queue.

- This operation is repeated for every pair of nodes which are stored in proper order because of priority queue

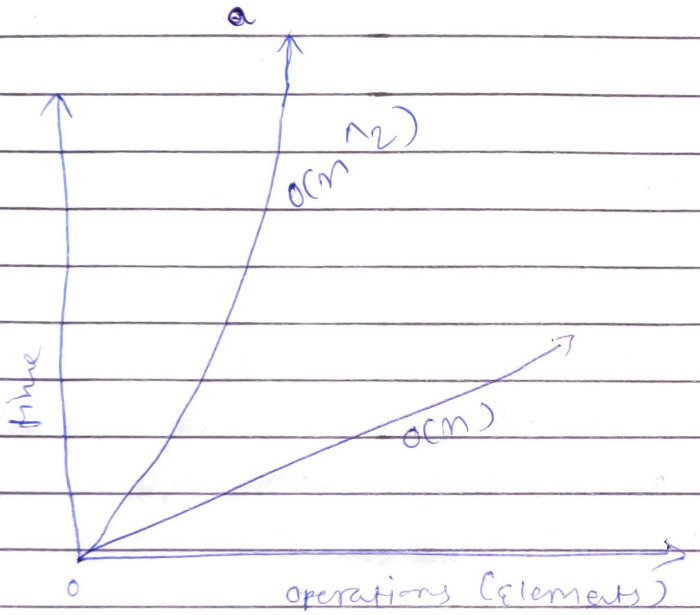
Ans (2)

(c) Running time complexity :-

(a) String operation = $O(n^2)$ since String class is immutable

(b) StringBuilder operation = $O(n)$
- StringBuilder is similar class as String class but mutable in nature. So for every element, the operation will take only $O(n)$.

(D)



Ans: 3

- user class have three attribute name, id and birth date.

- equals, hashcode and compareTo method get override with new defination as per the current requirement. comparable interface is extended to use compareTo method.

- equal method :-
  - equal method get object as argument and return boolean value
  - if current object is same as another object, then it will directly return true without checking further.
  - if another object is null or classtype of both objects are not same then function will return false.
  - cast other object to user class.
  - return true or false after checking each attributes of both objects.

- hashcode:
  - convert every attribute of object into hashcode and into result variable.
  - result variable returns from hashcode function.

- compareTo :
  - compareTo method is from comparable interface which compare the value of id attribute for each object

- compareTo function takes object as argument and return int value after comparing id attribute of current and another object.

- first < second object → -negative value
  first > second object → + positive
  first = second object → 0

Ans: (4)

pass-by-value

In Java, you can only change the content of object and not the reference which means that Java manipulates objects by reference, when the method arguments are passed they are not passed by reference but by value.

(A) - aDog.getName().equals("Bella");

True :- In changeMethod, new object is created which has new reference which doesn't affect old object. So old object will have "Bella" as name not molly.

- aDog.getName().equals("Molly");

False: As explained earlier, original object's reference will not change so it holds "Bella" as Name.

- aDog == oldDog;

True: we are assigning the reference of aDog to oldDog in upper statement which means both object holds same reference.

- d.getName().equals("Bella");

True: In Name attribute of new object d is same as name attribute of aDog which is "Bella" so that is why both have "Bella" in name attribute.

(B)

- aDog.getName().equals("Molly");

True:-

When we pass the object aDog in the method changeName, the refeeence of oldDog and aDog is the same. So when the name is set as molly the value gets updated on the refeeence and when we check the value of a dog as bella, it is true as it has the reference.

- aDog == oldDog;

True: The reference of oldDog and aDog is the same. so both will have the same value.

- d.getName().equals("Bella");

True: In the parameter, the object value of d is bella so it is true.