**Question-1 (10 points)**

**Consider the following singly linked list data structure, implement a method find\_middle() that finds and returns the middle element in the list. What is time complexity of your function?**

# Consider the following singly linked list data structure,  
# Implement a method find\_middle() that finds and returns the middle element in the list.  
# What is time complexity of your function?  
  
# Python3 program to find middle of linked list  
# Node class  
class Node:  
  
 # Function to initialise the node object  
 def \_\_init\_\_(self, data):  
 self.data = data # Assign data  
 self.next = None # Initialize next as null  
  
  
# Linked List class contains a Node object  
class LinkedList:  
  
 # Function to initialize head  
 def \_\_init\_\_(self):  
 self.head = None  
  
 # Function to insert a new node at the beginning  
 def push(self, new\_data):  
 new\_node = Node(new\_data)  
 new\_node.next = self.head  
 self.head = new\_node  
  
 # Print the linked list  
 def printList(self):  
 node = self.head  
 while node:  
 print(str(node.data) + "->", end="")  
 node = node.next  
 print("NULL")  
  
 # Function that returns middle.  
 def find\_Middle(self):  
 # Initialize two pointers, one will go one step a time (slow), another two at a time (fast)  
 slow = self.head  
 fast = self.head  
  
 # Iterate till fast's next is null (fast reaches end)  
 while fast and fast.next:  
 slow = slow.next  
 fast = fast.next.next  
  
 # return the slow's data, which would be the middle element.  
 print("The middle element is ", slow.data)  
  
  
# Code execution starts here  
if \_\_name\_\_ == '\_\_main\_\_':  
  
 # Start with the empty list  
 llist = LinkedList()  
  
 for i in range(5, 0, -1):  
 llist.push(i)  
 llist.printList()  
 llist.find\_Middle()  
  
  
# Time complexity- Logarithmic Time — O(log n)

**Question-2 (10 points)**

**In a binary tree, implement a Python function count, that takes in an integer N and returns the number of times N appears in the binary tree. Note that this is not a binary search tree. What is the time complexity of your function?**

# In a binary tree, implement a Python function count, that takes in an integer N  
# and returns the number of times N appears in the binary tree.  
# Note that this is not a binary search tree. What is the time complexity of your function?  
  
class BinaryTree:  
 class \_Node:  
 def \_\_init\_\_(self, element, left=None, right=None):  
 self.\_left = left  
 self.\_right = right  
 self.\_element = element  
  
 def \_\_init\_\_(self):  
 self.\_root = None  
 self.\_size = 0  
  
 def count(self, N):  
 if self.root is None:  
 return 0  
 queue = []  
 queue.append(self.root)  
 count = 0  
 while (len(queue) > 0):  
 node = queue.pop(0)  
 if node.\_element == N:  
 count = count + 1  
 if node.left is not None:  
 queue.append(node.left)  
 if node.right is not None:  
 queue.append(node.right)  
 return count

**Question-3 (10 points)**

**Consider an array-based binary tree implementation, write a method find\_parents, that takes in an index i and returns all ancestors of node located at index i. What is time complexity of your function?**

**Question-4 (10 points)**

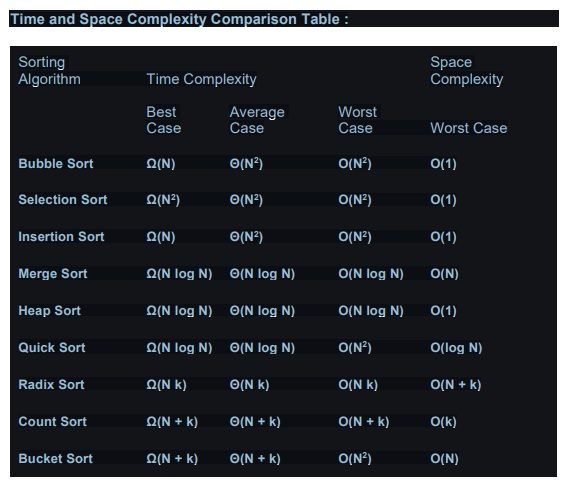
**In a binary search tree:**

* **What is worst case time complexity of the binary search function?**
* **Provide an example binary search tree that exhibits worst case running time of binary\_search function**
* **Write a function that prints elements in binary search tree in order**

# In a binary search tree:  
# What is worst case time complexity of the binary search function?  
# Provide an example binary search tree that exhibits worst case running time of binary\_search function.  
# Write a function that prints elements in binary search tree in order.  
  
# Ans-  
# Worst case of time complexity of the binary\_search function = 0(n) n is the number of elements in BST  
  
class Node:  
 def \_\_init\_\_(self,key):  
 self.left = None  
 self.right = None  
 self.val = key  
 def insert\_element(root,node):  
 if root is None:  
 root = node  
 else:  
 if root.val < node.val:  
 if root.right is None:  
 root.right = node  
 else:  
 insert\_element(root.right,node)  
 else:  
 if root.left is None:  
 root.left = node  
 else:  
 insert\_element(root.left,node)  
  
 def binary\_search(root,key):  
 print(root.val)  
 if root is None or root.val == key:  
 return root  
 if root.val < key:  
 return binary\_search(root.right,key)  
 return binary\_search(root.left,key)

**Question-5 (10 points)**

**Compare the following sorting algorithms with respect to their time complexity (worst and average cases) and whether sorting happens in place: selection sort, insertion sort, heap sort, merge sort and quick sort.**

****

**Question-6 (10 points)**

**Draw the contents of the hash table in the boxes below given the following conditions: The size of the hash table is 12. Open addressing and Linear probing is used to resolve collisions. The hash function used is H(k) = k mod 12 What values will be in the hash table after the following sequence of insertions?**

**Draw the values in the boxes below, and show your work for partial credit.**

**36, 10, 9, 13, 12, 45, 25, 34**

Given

- Size of the Hash table is 12

- Open addressing of type linear probing is used to resolve collisions

- H(K) = K mod 12 or H(K) = K%12

- K values = 36, 10, 9, 13, 12, 45, 25, 34

Therefore,

Hash values

36%12 = 0

10%12 = 10

9%12 = 9

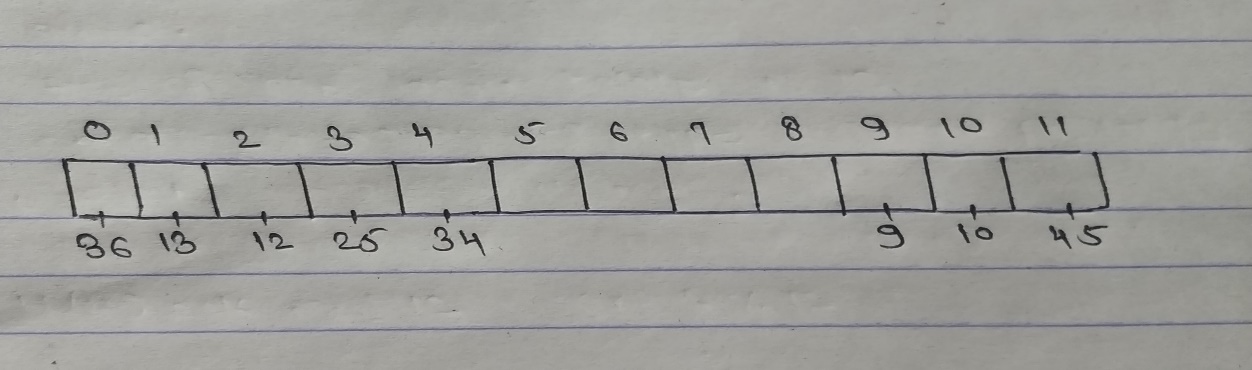
13%12 = 1

12%12 = 0

45%12 = 9

25%12 = 1

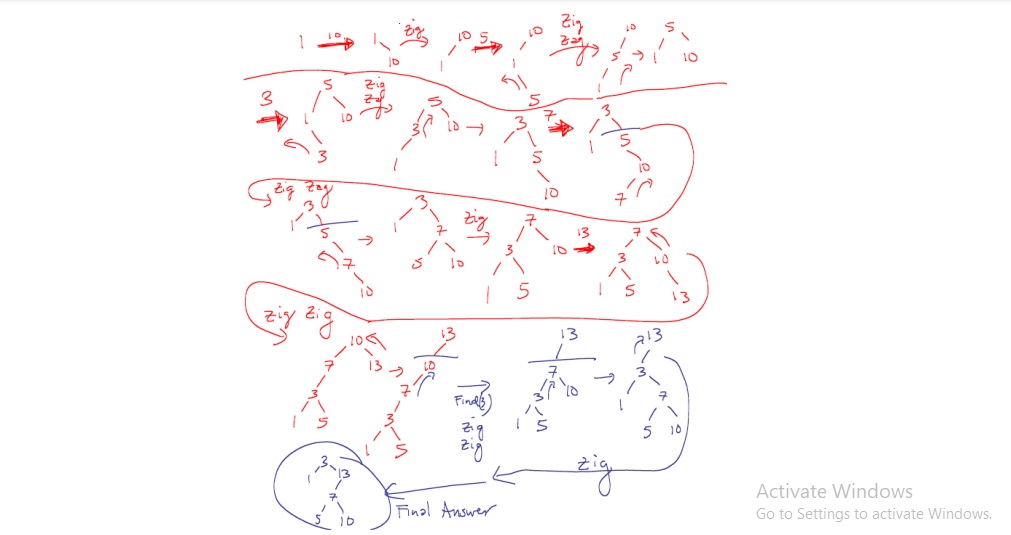
34%12 = 10



36 is placed at index ‘0’, 10 is placed at index ‘10’, 9 is placed at index ‘9’, 13 is placed at index ‘1’. Now Hash value of 12 is also 0which is already occupied so next empty index position is used to place 12 [here at index ’2’], again collision for 45 so next empty position after index 9 in circular way is used to place 45 [at index ‘11’], again collision for 25 so next empty position after index 1 in circular way is used to place 25 [at index ‘3’], again collision for 34 so next empty position after index 10 in circular way is used to place 34 [at index ‘4’]

**Question-7 (10 points)**

**Imagine that the following operations are performed on an initially empty splay tree: Insert(1), Insert(10), Insert (5), Insert (3), Insert (7), Insert (13), Find (3). Show the state of the splay tree after performing each of the above operations. Be sure to label each of your trees with what operations you have just completed.**

****

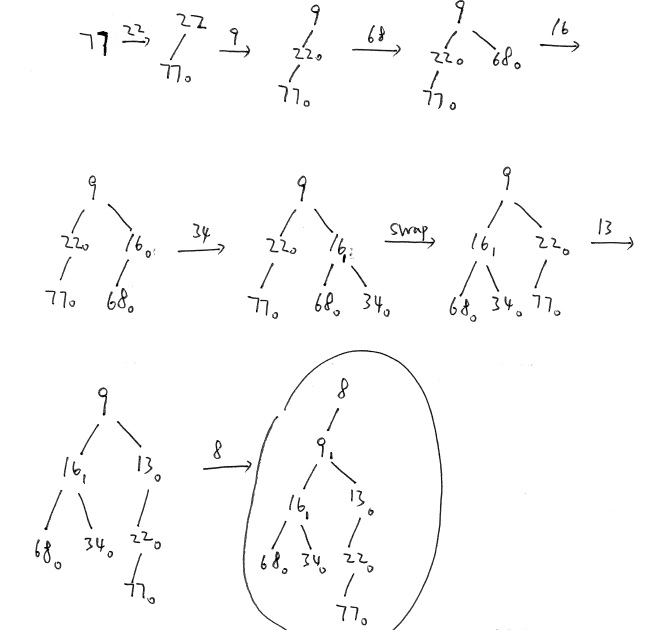
**Question-8 (10 points)**

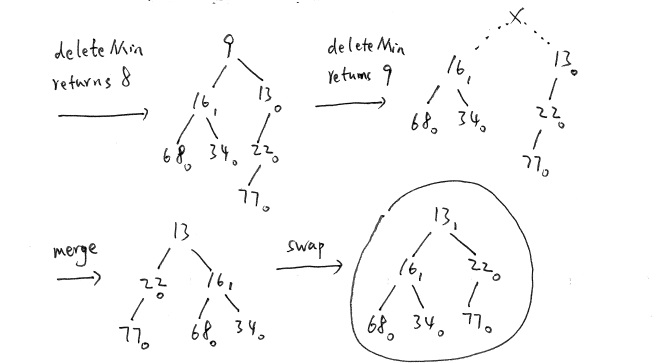
**Heaps**

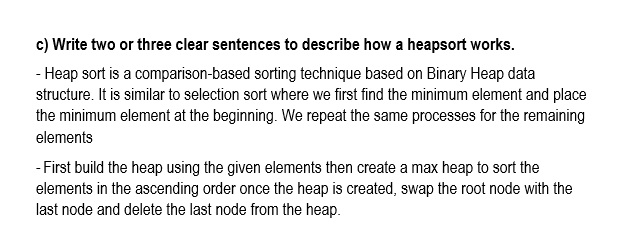
**a) Draw the binary min heap that results from inserting: 77, 22, 9, 68, 16, 34, 13, 8 in that order into an initially empty binary min heap. You do not need to show the array representation of the heap. You are only required to show the final heap, although if you draw intermediate heaps, please circle your final result for ANY credit.**

**b) Draw the binary min heap that results from doing 2 deletemins on the heap you created in part a. You are only required to show the final heap, although if you draw intermediate heaps please circle your final result for ANY credit.**

**c) Write two or three clear sentences to describe how a heapsort works.**

****

****

****

**Question-9** (20 points)

* Which type of queue can be used as a queue or a stack?

# Double-ended queue

* Which construct is used by regular queues

# first-in, first-out

* Which method retrieves and removes the first element from a deque?

# removeFirst()

* Consider the following operation performed on a stack of size 5. Push(1), Pop(), Push(2), Push(3), Pop(), Push(4), Pop(), Pop(), Push(5). After the completion of all operation, the number of elements present on stack are

# 1

* What method is used to add an element to a Queue?

# enqueue()

* Conversion of infix arithmetic expression to postfix expression uses:

# Queue

* The following circular queue can accommodate a maximum six elements with the following data front = 2 rear = 4

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | A | B | C |  |

What will happen after inserting D and E operations take place?

# Answer: front = 2, rear = 0

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **E** |  | **A** | **B** | **C** | **D** |

* Explain the functionality of below recursive functions. def fun1(n):

i = 0

if (n > 1):

fun1(n - 1)

for i in range(n):

print(" \* ",end="")

# Driver code

a = 3 fun1(a)

# Answer: \*\*\*\*\*\*

* Predict the output of the following program: def fun(x):

if(x > 0):

x -= 1

fun(x)

print(x , end=" ") x -= 1

fun(x)

# Driver code fun(4)

# Answer: 0 1 2 0 3 0 1

* Predict the output of the following program: def fun( a, n):

if n == 1:

return a[0] else:

x = fun(a, n - 1)

if x > a[n - 1]:

return x else:

return a[n - 1]

# Driver code

arr = [12, 10, 30, 50, 100]

print(fun(arr, 5))

# Answer: 100

* Assume the structure of a Linked List node is as follows class Node:

def init (self, data):

self.data = data self.next = None

What does the following function do for a given Linked List?

def fun1(head):

if head == None:

return fun1(head.next)

print(head.data, end = " ")

Assume this the current linked list as follows: 1→2→3→4→5

# Answer: 5 4 3 2 1