**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?

**class** SinglyList:

**class** Node:  
 **def** \_\_init\_\_(self, e, next):  
 self.element = e  
 self.next = next  
  
 **def** \_\_init\_\_(self):  
 self.head = **None**   
**def** find\_middle(self):

**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?

**class** BinaryTree:

**class** Node:  
 **def** \_\_init\_\_(self, element, left = **None**, right = **None**):  
 self.left = left  
 self.right = right  
 self.element: int = element  
  
 **def** \_\_init\_\_(self):  
 self.root = **None**

**def** count(self, N):

**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?

**class** ArrayBinaryTree:  
 **def** \_\_init\_\_(self):  
 self.heap = []  
  
 **def** find\_ancestors(self, i):

**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

**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.

|  |  |  |  |
| --- | --- | --- | --- |
| Sorting Algorithm | Worst Case | Average Case | Sorting In Place |
| selection sort |  |  |  |
| insertion sort |  |  |  |
| merge sort |  |  |  |
| 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

|  |  |
| --- | --- |
| 0 |  |
| 1 |  |
| 2 |  |
| 3 |  |
| 4 |  |
| 5 |  |
| 6 |  |
| 7 |  |
| 8 |  |
| 9 |  |
| 10 |  |
| 11 |  |

**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?

1. Regular queue
2. Double-ended queue
3. Priority queue
4. Ordered queue

* Which construct is used by regular queues?

1. first-in, last-out
2. last-in, first-out
3. last-in, last-out
4. first-in, first-out

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

1. pollFirst
2. removeFirst
3. remove
4. peek

* 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. 1
2. 2
3. 3
4. 4

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

1. dequeue()
2. enqueue()
3. push()
4. pop()

* Conversion of infix arithmetic expression to postfix expression uses:

1. Stack
2. Circular queue
3. Linked list
4. 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?

1. front = 1 rear = 5

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

1. front = 2 rear = 5

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

1. front = 2, rear = 0

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

1. front = 2, rear = 6

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

* 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)

1. \*\*\*\*\*
2. \*\*\*\*\*\*
3. \*\*\*\*\*\*\*
4. \*\*\*\*\*\*\*\*

* 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)

1. 0 1 2 0 1 0 3
2. 0 1 2 0 0 1 3
3. 0 1 2 0 3 0 1
4. 0 1 2 0 1 3 0

* 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))

1. 101
2. 200
3. 100
4. 201

* 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

1. 1 2 3 4 5
2. 5 4 3 2 1
3. 1 1 1 1 1
4. 5 5 5 5 5