M. Rohith
3122 21 5001 085

Sri Sivasubramaniya Nadar College of Engineering, Kalavakkam - 603 110
(An Autonomous Institution, Affiliated to Anna University, Chennai)

UCS2403: DESIGN & ANALYSIS OF ALGORITHMS

Assignment 5

1. (a) First, find the k^{th} smallest element in an unsorted list using insertion sort
   (b) Next, find the element by modifying the divide-and-conquer algorithm of Quicksort
   (c) Compare the time complexity of both the algorithms
2. Consider the code given below that has to find the sum of the values in the nodes of a binary tree.

```
# Code to populate a tree starts here
import random
class TreeNode:
    def __init__(self):
        self.data = 0
        self.left = None
        self.right = None

    def insert(self, data):
        if data < self.data:
            if self.left == None:
                tempNode = TreeNode()
                self.left = tempNode
                self.left.data = data
            else:
                self.left.insert(data)
        elif data > self.data:
            if self.right == None:
                tempNode = TreeNode()
                self.right = tempNode
                self.right.data = data
            else:
                self.right.insert(data)

    def traverseInOrder(self):
```

M. Rohith
3122 21 5001 085

```
        if self.left != None:
            self.left.traverseInOrder()
        print(self.data, end=' ')
        if self.right != None:
            self.right.traverseInOrder()

def createRoot():
    i = random.randint(0, 10)
    rootNode = TreeNode()
    rootNode.data = i
    return rootNode

def createTree():
    rootNode = createRoot()
    numNodes = random.randint(1, 10)
    currentNode = rootNode
    j = 0
    L = []
    while (j <= numNodes):
        newVal = random.randint(1,20)
        if newVal not in L:
            currentNode.insert(newVal)
            L.append(newVal)
        j+=1
    rootNode.traverseInOrder()
    return rootNode
# Code to populate the tree ends here

def getSum(node):
    if node == None:
        return 0
    else:
        leftSum = getSum(node.left)
        rightSum = getSum(node.right)
        return leftSum + rightSum

rootNode = createTree()
print("Sum = ",getSum(rootNode))
```

(a) The code is known to have some bugs. Modify the given program to correctly find the sum.

(b) Use Hypothesis to find counterexamples to show that the given code has errors.

(c) Please note that the number of nodes in the tree and the value in each node are generated randomly.

M. Rohith
3122 21 5001 085

1. **A) Program code:**

```
#Function for performing insertion sort
def insertionSort(arr):
    if(len(arr)<=1):
        return
    n=len(arr)
    for i in range(1,n):
        key=arr[i]
        # Move elements of arr[0..i-1], that are
        # greater than key, to one position ahead
        # of their current position
        j=i-1
        while j>=0 and key<arr[j]:
            arr[j+1]=arr[j]
            j-=1
        arr[j+1]=key


# Function to return K'th smallest
# element in a given array
def kthsmallest(arr,k):
    insertionSort(arr)
    return arr[k-1]


if __name__=='__main__':
    arr=[]
    n=int(input("Enter the number of elements: "))
    for i in range(n):
        arr.append(int(input("Enter the element: ")))
    print("\nEntered array: ",arr)
    k=int(input("\nEnter the k value: "))
    val=kthsmallest(arr,k)
    print("\nSorted array: ",arr)
    print("\n",k,"th smallest value is: ",val)
```

Output:

```
PS C:\Rohith\Backup\Desktop\SEM 4\Design and Analysis of Algorithms lab\Assignment-6> python 1a.py
Enter the number of elements: 5
Enter the element: 12
Enter the element: 5
Enter the element: 7
Enter the element: 13
Enter the element: 19

Entered array:  [12, 5, 7, 13, 19]

Enter the k value: 2

Sorted array:  [5, 7, 12, 13, 19]

 2 th smallest value is:  7
PS C:\Rohith\Backup\Desktop\SEM 4\Design and Analysis of Algorithms lab\Assignment-6>
```

b) Program code:

```python
# Python3 implementation of QuickSort

# Function to find the partition position
def partition(array, low, high):

    # Choose the rightmost element as pivot
    pivot = array[high]

    # Pointer for greater element
    i = low - 1

    # Traverse through all elements
    # compare each element with pivot
    for j in range(low, high):
        if array[j] <= pivot:
            # If element smaller than pivot is found
            # swap it with the greater element pointed by i
            i = i + 1

            # Swapping element at i with element at j
            (array[i], array[j]) = (array[j], array[i])

    # Swap the pivot element with
    # e greater element specified by i
    (array[i + 1], array[high]) = (array[high], array[i + 1])
```

M. Rohith
3122 21 5001 085

```
    # Return the position from where partition is done
    return i + 1

# Function to perform quicksort
def quick_sort(array, low, high):
    if low < high:

        # Find pivot element such that
        # element smaller than pivot are on the left
        # element greater than pivot are on the right
        pi = partition(array, low, high)

        # Recursive call on the left of pivot
        quick_sort(array, low, pi - 1)

        # Recursive call on the right of pivot
        quick_sort(array, pi + 1, high)


# Function to return K'th smallest
# element in a given array
def kthsmallest(arr,k):
    quick_sort(arr,0,len(arr)-1)
    return arr[k-1]


# Driver code
if __name__=='__main__':
    arr=[]
    n=int(input("Enter the number of elements: "))
    for i in range(n):
        arr.append(int(input("Enter the element: ")))
    print("\nEntered array: ",arr)
    k=int(input("\nEnter the k value: "))
    val=kthsmallest(arr,k)
    print("\nSorted array: ",arr)
    print("\n",k,"th smallest value is: ",val)
```

Output:

```
PS C:\Rohith\Backup\Desktop\SEM 4\Design and Analysis of Algorithms lab\Assignment-6> python 1b.py
Enter the number of elements: 5
Enter the element: 12
Enter the element: 5
Enter the element: 7
Enter the element: 13
Enter the element: 19

Entered array:  [12, 5, 7, 13, 19]

Enter the k value: 2

Sorted array:  [5, 7, 12, 13, 19]

 2 th smallest value is:  7
PS C:\Rohith\Backup\Desktop\SEM 4\Design and Analysis of Algorithms lab\Assignment-6>
```

c) Time complexity for Insertion sort algorithm:

The time complexity of the given code can be analyzed as follows:

- The insertionSort function takes O(n^2) time complexity because it uses two nested loops. The outer loop runs n-1 times, and the inner loop may run up to n-1 times as well in the worst case. Therefore, the overall time complexity of insertionSort is O(n^2).
- The kthsmallest function calls the insertionSort function once, which takes O(n^2) time complexity. Then, it returns the k-th element of the sorted array, which takes constant time O(1).
- The for loop in the main function runs n times, and each iteration of the loop takes constant time O(1) to append an element to the array. Therefore, the time complexity of the main function is O(n).

Overall, the time complexity of the given code is dominated by the insertionSort function, which takes O(n^2) time complexity. Therefore, the time complexity of the given code is O(n^2).

Time complexity for Quick sort algorithm:

- The time complexity of the QuickSort algorithm used in the given code is O(n log n) in the average case and O(n^2) in the worst case, where n is the number of elements in the input array.
- The partition function takes O(n) time in the worst case, where n is the number of elements in the input array. The quick_sort function divides the array into two subarrays around the pivot element using the partition function and recursively calls itself on the two subarrays. The time complexity of the quick_sort function depends on the depth of the recursive calls and the time taken by the partition function at each level. In the average case, the depth of the recursive calls is O(log n) and the time taken by the partition function at each level is O(n). Therefore, the overall time complexity of the quick_sort function in the average case is O(n log n).
- In the worst case, the pivot element chosen at each level of the quick_sort function is the largest or smallest element in the subarray. In this case, the depth of the recursive calls is n-1

and the time taken by the partition function at each level is O(n). Therefore, the overall time complexity of the quick_sort function in the worst case is O(n^2).

- The kthsmallest function calls the quick_sort function once and returns the kth smallest element from the sorted array. Therefore, the time complexity of the kthsmallest function is the same as the time complexity of the quick_sort function, i.e., O(n log n) in the average case and O(n^2) in the worst case.

2. **A) Program code:**

```
# Code to populate a tree starts here
import random
class TreeNode:
    def __init__(self):
        self.data = 0
        self.left = None
        self.right = None

    def insert(self, data):
        if data < self.data:
            if self.left == None:
                tempNode = TreeNode()
                self.left = tempNode
                self.left.data = data
            else:
                self.left.insert(data)
        elif data > self.data:
            if self.right == None:
                tempNode = TreeNode()
                self.right = tempNode
                self.right.data = data
            else:
                self.right.insert(data)

    def traverseInOrder(self):
        if self.left != None:
            self.left.traverseInOrder()
        print(self.data, end=' ')
        if self.right != None:
            self.right.traverseInOrder()

def createRoot():
    i = random.randint(0, 10)
```

M. Rohith
3122 21 5001 085

```
    rootNode = TreeNode()
    rootNode.data = i
    return rootNode

def createTree():
    rootNode = createRoot()
    numNodes = random.randint(1, 10)
    currentNode = rootNode
    j=0
    L = []
    while (j <= numNodes):
        newVal = random.randint(1,20)
        if newVal not in L:
            currentNode.insert(newVal)
            L.append(newVal)
        j+=1
    rootNode.traverseInOrder()
    return rootNode
# Code to populate the tree ends here

def getSum(node):
    if node == None:
        return 0
    else:
        leftSum = getSum(node.left)
        rightSum = getSum(node.right)
        return node.data + leftSum + rightSum

rootNode = createTree()
print("Sum = ",getSum(rootNode))
```

Output:

```
PS C:\Rohith\Backup\Desktop\SEM 4\Design and Analysis of Algorithms lab\Assignment-6> python 2a.py
3 6 8 9 10 12 17 19 20 Sum =  104
PS C:\Rohith\Backup\Desktop\SEM 4\Design and Analysis of Algorithms lab\Assignment-6>
```

M. Rohith
3122 21 5001 085

b) Incorrect Program code:

```
from hypothesis import given
from hypothesis import given
from hypothesis.strategies import integers

# Code to populate a tree starts here
import random
class TreeNode:
    def __init__(self):
        self.data = 0
        self.left = None
        self.right = None

    def insert(self, data):
        if data < self.data:
            if self.left == None:
                tempNode = TreeNode()
                self.left = tempNode
                self.left.data = data
            else:
                self.left.insert(data)
        elif data > self.data:
            if self.right == None:
                tempNode = TreeNode()
                self.right = tempNode
                self.right.data = data
            else:
                self.right.insert(data)

    def traverseInOrder(self):
        if self.left != None:
            self.left.traverseInOrder()
        print(self.data, end=' ')
        if self.right != None:
            self.right.traverseInOrder()

def createRoot():
    i = random.randint(0, 10)
    rootNode = TreeNode()
    rootNode.data = i
    return rootNode
```

M. Rohith
3122 21 5001 085

```
def createTree():
    rootNode = createRoot()
    numNodes = random.randint(1, 10)
    currentNode = rootNode
    j=0
    L = []
    while (j <= numNodes):
        newVal = random.randint(1,20)
        if newVal not in L:
            currentNode.insert(newVal)
            L.append(newVal)
        j+=1
    rootNode.traverseInOrder()
    return rootNode
# Code to populate the tree ends here

def getSum(node):
    if node == None:
        return 0
    else:
        leftSum = getSum(node.left)
        rightSum = getSum(node.right)
        return  leftSum + rightSum

rootNode = createTree()
print("Sum = ",getSum(rootNode))

@given(integers())
def test_sum_property(value):
    #create BST with a single node containing the value
    root=TreeNode()
    root.data=value
    assert getSum(root)==value

#run the property based test
test_sum_property()
```

Output:

```
PS C:\Rohith\Backup\Desktop\SEM 4\Design and Analysis of Algorithms lab\Assignment-6> python 2b.py
3 7 10 16 17 20 Sum =  0
Traceback (most recent call last):
  File "C:\Rohith\Backup\Desktop\SEM 4\Design and Analysis of Algorithms lab\Assignment-6\2b.py", line 77, in <module>
    test_sum_property()
  File "C:\Rohith\Backup\Desktop\SEM 4\Design and Analysis of Algorithms lab\Assignment-6\2b.py", line 70, in test_sum_property
    def test_sum_property(value):
  File "C:\Python310\lib\site-packages\hypothesis\core.py", line 1396, in wrapped_test
    raise the_error_hypothesis_found
  File "C:\Rohith\Backup\Desktop\SEM 4\Design and Analysis of Algorithms lab\Assignment-6\2b.py", line 74, in test_sum_property
    assert getSum(root)==value
AssertionError
Falsifying example: test_sum_property(
    value=1,
)
```

Correct Program code:

```python
from hypothesis import given
from hypothesis import given
from hypothesis.strategies import integers

# Code to populate a tree starts here
import random
class TreeNode:
    def __init__(self):
        self.data = 0
        self.left = None
        self.right = None

    def insert(self, data):
        if data < self.data:
            if self.left == None:
                tempNode = TreeNode()
                self.left = tempNode
                self.left.data = data
            else:
                self.left.insert(data)
        elif data > self.data:
            if self.right == None:
                tempNode = TreeNode()
                self.right = tempNode
                self.right.data = data
            else:
                self.right.insert(data)
```

M. Rohith
3122 21 5001 085

```
    def traverseInOrder(self):
        if self.left != None:
            self.left.traverseInOrder()
        print(self.data, end=' ')
        if self.right != None:
            self.right.traverseInOrder()

def createRoot():
    i = random.randint(0, 10)
    rootNode = TreeNode()
    rootNode.data = i
    return rootNode

def createTree():
    rootNode = createRoot()
    numNodes = random.randint(1, 10)
    currentNode = rootNode
    j=0
    L = []
    while (j <= numNodes):
        newVal = random.randint(1,20)
        if newVal not in L:
            currentNode.insert(newVal)
            L.append(newVal)
        j+=1
    rootNode.traverseInOrder()
    return rootNode
# Code to populate the tree ends here

def getSum(node):
    if node == None:
        return 0
    else:
        leftSum = getSum(node.left)
        rightSum = getSum(node.right)
        return node.data + leftSum + rightSum

rootNode = createTree()
print("Sum = ",getSum(rootNode))

@given(integers())
def test_sum_property(value):
```

M. Rohith
3122 21 5001 085

```
    #create BST with a single node containing the value
    root=TreeNode()
    root.data=value
    assert getSum(root)==value

#run the property based test
test_sum_property()
```

Output:

```
PS C:\Rohith\Backup\Desktop\SEM 4\Design and Analysis of Algorithms lab\Assignment-6> python 2b.py
1 3 4 6 9 14 16 17 18 20 Sum =  108
PS C:\Rohith\Backup\Desktop\SEM 4\Design and Analysis of Algorithms lab\Assignment-6>
```