**Python & Data Structures Laboratory B.Tech. 3rd Semester**



**Name :**

**Roll Number :**

**Department : Computer Science and Engineering**

**Faculty of Engineering & Technology**

**Ramaiah University of Applied Sciences**

**Ramaiah University of Applied Sciences**

Private University Established in Karnataka State by Act No. 15 of 2013

|  |  |
| --- | --- |
| Faculty | Engineering & Technology |
| Programme | B. Tech. in Computer Science and Engineering |
| Year/Semester | 2nd year/ 3rd semester |
| Name of the Laboratory | Data Structures and Algorithms Laboratory |
| Laboratory Code | CSD201B |

List of Experiments

1. Array
2. Linked List
3. Stack
4. Queue
5. Binary Tree
6. Binary Search Tree
7. Heap
8. AVL Tree
9. Quick Sort
10. Merge Sort

# Index Sheet

|  |  |  |  |
| --- | --- | --- | --- |
| **No** | **Lab Experiment** | **Lab Documents** | **Viva** |
| 1 | Array | 10% of Lab CE |  |
| 2 | Linked List | 10% of Lab CE |  |
| 3 | Stack | 10% of Lab CE |  |
| 4 | Queue | 10% of Lab CE |  |
| 5 | Binary Tree | 10% of Lab CE |  |
| 6 | Binary Search Tree | 10% of Lab CE |  |
| 7 | Heap | 10% of Lab CE |  |
| 8 | AVL Tree | 10% of Lab CE |  |
| 9 | Quick Sort | 10% of Lab CE |  |
| 10 | Lab Internal | 10% of Lab CE | |

|  |  |  |
| --- | --- | --- |
| **No** | **Experiment Name** | **Page No** |
| 1 | Array | 5 |
| 2 | Linked List | 7 |
| 3 | Stack | 9 |
| 4 | Queue | 11 |
| 5 | Binary Tree | 12 |
| 6 | Binary Search Tree | 14 |
| 7 | Heap | 15 |
| 8 | AVL Tree | 16 |
| 9 | Quick Sort | 17 |

# Experiment 1

**Title of the laboratory experiment**: Array

**1. Aim:**

To understand and implement the basic operations in arrays using python.

**2. Objective:**

To execute the below operations:

1. Traverse − print all the array elements one by one.
2. Insertion − Adds an element at the given index.
3. Deletion − Deletes an element at the given index.
4. Search − Searches an element using the given index or by the value.
5. Update − Updates an element at the given index.

**3. Exercise:**

To develop a python to perform the below tasks:

1. Create your own list of your favourite five sportsperson. Using this find out,
2. Length of the list.
3. Add a sixth sportsperson at the end of this list.
4. You realize that you need to add the sixth sportsperson after the second sportsperson, so remove it from the list first and then add it after the second sportsperson.
5. Now you don't like two sportspersons. Now remove those two and replace them with any other two sportspersons.
6. Sort the sportspersons list in alphabetical order (hint: use the dir() functions to list down all functions available in the list).
7. Create a list of all even numbers between number x and number y.
8. Sort the sportspersons list in alphabetical order (hint: use the dir() functions to list down all functions available in the list). The number x should be your age, and the number y should be your father's or mother's age.

**4. Experimental Procedure**

1. Algorithm design

“Write the pseudocode of the main operations of the given data structure”

1. Program

“Paste the screenshot of the executed python code”

1. Presentation of the results

“Paste the output of the program”

1. Analysis and discussions

“Discuss the time complexities of all the operations of the given data structure”

# Experiment 2

**Title of the Laboratory Exercise**: Linked List

**1. Aim:**

To understand and implement the basic operations in Circular Doubly Linked List using python.

**2. Objective:**

To execute the below operations in Circular Doubly Linked List:

1. Insert: Inserts an element after a specific value.
2. Delete: Deletes an element having a specific value.
3. Display: Prints the elements in the forward direction as well as in the reverse direction.

**3. Exercise:**

In a Circular Doubly Linked List class, implement the below four operations:

**def insert\_after\_value(self, data\_after, data\_to\_insert):**

# Search for first occurance of data\_after value in linked list

# Now insert data\_to\_insert after data\_after node

**def remove\_by\_value(self, data):**

# Remove first node that contains data

**def print\_forward(self):**

# This method prints list in forward direction. Use node.next. Use a print statement to print the nodes in forward direction starting from the first node to the last node.

**def print\_backward(self):**

# Print linked list in reverse direction. Use node.prev for this. Use a print statement to print the nodes in backward direction starting from the last node to the first node.

Now make following calls,

LL = LinkedList()

LL.insert\_values(["Red","Yellow","Purple","Orange"])

LL.print()

LL.insert\_after\_value("Yellow","Blue") # insert Blue after Yellow

LL.print()

LL.remove\_by\_value("orange") # remove Orange from linked list

LL.print()

LL.remove\_by\_value("Green")

LL.print()

LL.remove\_by\_value("Red")

LL.remove\_by\_value("Yellow")

LL.remove\_by\_value("Blue")

LL.remove\_by\_value("Purple")

LL.print()

LL.print\_forward()

LL.print\_backward()

**4. Experimental Procedure**

1. Algorithm design

“Write the pseudocode of the main operations of the given data structure”

1. Program

“Paste the screenshot of the executed python code”

1. Presentation of the results

“Paste the output of the program”

1. Analysis and discussions

“Discuss the time complexities of all the operations of the given data structure”

# Experiment 3

**Title of the Laboratory Exercise**: Stack

**1. Aim:**

To understand and implement the basic operations in stack using python.

**2. Objective:**

To execute the below operations in stack:

1. Push: Pushing (storing) an element on the stack.
2. Pop: Removing (accessing) an element from the stack.
3. Peek: get the top data element of the stack, without removing it.
4. Check if stack is full.
5. Check if stack is empty.

**3. Exercise:**

1. Write a function in python that can reverse a string (your full name) using stack data structure. Create a function called “reverse\_myname” which does this operation.

Follow the steps given below to reverse a string using stack:

1. Create an empty stack.
2. One by one push all characters of string to stack by calling a push().
3. One by one pop all characters from stack and put them back to string
4. by calling a pop().
5. Create a Python function named "isit\_balanced" that determines if the string's parentheses are balanced or not. "{}',"()" or "[]" are examples of parentheses.

**4. Experimental Procedure**

1. Algorithm design

“Write the pseudocode of the main operations of the given data structure”

1. Program

“Paste the screenshot of the executed python code”

1. Presentation of the results

“Paste the output of the program”

1. Analysis and discussions

“Discuss the time complexities of all the operations of the given data structure”

# Experiment 4

**Title of the Laboratory Exercise**: Queue

**1. Aim:**

To understand and implement the basic operations in deque using python.

**2. Objective:**

To execute the below operations in a full binary tree:

1. Insert an element at the front end of the deque.
2. Delete an element at the rear end of the deque.

**3. Exercise:**

Using the deque data structure, insert some elements at the front and delete an element at the rear end of the deque. The maximum size of the array is 6. Check the conditions of overflow and underflow before carrying out insertion and deletion, respectively.

**4. Experimental Procedure**

1. Algorithm design

“Write the pseudocode of the main operations of the given data structure”

1. Program

“Paste the screenshot of the executed python code”

1. Presentation of the results

“Paste the output of the program”

1. Analysis and discussions

“Discuss the time complexities of all the operations of the given data structure”

# Experiment 5

**Title of the Laboratory Exercise**: Binary Tree

**1. Aim:**

To understand and implement the basic operations in full binary tree using python.

**2. Objective:**

To execute the below operations in a full binary tree:

1. Search − Searches an element in a tree.
2. Insert − Inserts an element in a tree.
3. Pre-order Traversal − Traverses a tree in a pre-order manner.
4. In-order Traversal − Traverses a tree in an in-order manner.
5. Post-order Traversal − Traverses a tree in a post-order manner.

**3. Exercise:**

Construct a full binary tree with 10 nodes, where the data item inserted at every node should be a random value between 1 and 100. Add the following methods to the class named "FullBinaryTree" and perform the operation on the constructed full binary tree.

1. find\_min(): finds the minimum element stored in the constructed Full binary tree.
2. find\_max(): finds the maximum element stored in the constructed Full binary tree.
3. calculate\_sum(): calculates the sum of all elements stored in the constructed Full binary tree.
4. pre\_order\_traversal(): performs pre-order traversal of the constructed Full binary tree.
5. post\_order\_traversal(): performs post-order traversal of the constructed Full binary tree.
6. in\_order\_traversal(): performs in-order traversal of the constructed Full binary tree.

**4. Experimental Procedure**

1. Algorithm design

“Write the pseudocode of the main operations of the given data structure”

1. Program

“Paste the screenshot of the executed python code”

1. Presentation of the results

“Paste the output of the program”

1. Analysis and discussions

“Discuss the time complexities of all the operations of the given data structure”

# Experiment 6

**Title of the Laboratory Exercise**: Binary Search Tree

**1. Aim:**

To understand and implement the basic operations in Binary Search Tree using python.

**2. Objective:**

To execute the below operations in a Binary Search Tree (BST):

1. Search − Searches an element in a BST.
2. Insert − Inserts an element in a BST.
3. Delete − Deletes an element in a BST.
4. Check the balance of the BST.
5. Determine the height of the BST.

**3. Exercise:**

Construct a binary search tree with the below values: {12, 35, 14, 97, 36, 65, 89}. Write a python program to perform the following operations:

1. Insert a new element which is having a value equivalent to the “last two digits of your roll number”.
2. To determine the height of the constructed BST.
3. Delete any element from the constructed BST.
4. To check if the constructed BST is Balanced or not.

**4. Experimental Procedure**

1. Algorithm design

“Write the pseudocode of the main operations of the given data structure”

1. Program

“Paste the screenshot of the executed python code”

1. Presentation of the results

“Paste the output of the program”

1. Analysis and discussions

“Discuss the time complexities of all the operations of the given data structure”

# Experiment 7

**Title of the Laboratory Exercise**: Heap

**1. Aim:**

To understand and implement the basic operations in Heap using python.

**2. Objective:**

To execute the below operations in a Heap:

**Exercise:**

Implement a Python program that constructs a Max Heap tree having the following elements: 2, 4, 6, 8, 10, 12, 14, 16, 18 and 20 and perform the following operation on the constructed Heap Tree.

1. Insert a new element whose value is equivalent to the sum of the digits of your roll number.
2. Find the maximum element in the constructed Max Heap.
3. Delete the root element (maximum element) two times from the Max Heap.

**4. Experimental Procedure**

1. Algorithm design

“Write the pseudocode of the main operations of the given data structure”

1. Program

“Paste the screenshot of the executed python code”

1. Presentation of the results

“Paste the output of the program”

1. Analysis and discussions

“Discuss the time complexities of all the operations of the given data structure”

# Experiment 8

**Title of the Laboratory Exercise**: AVL Tree

**1. Aim:**

To understand and implement the basic operations in AVL using python.

**2. Objective:**

To execute the below operations in an AVL Tree:

1. Left rotation
2. Right rotation
3. Left-Right rotation
4. Right-Left rotation

**3. Exercise:**

Implement a Python program that constructs an AVL tree having the following elements: Z, I, J, F, A, E, C, P, B, D, H, N. Consider the order of the elements in ascending order. Explain the rotations diagrammatically.

**4. Experimental Procedure**

1. Algorithm design

“Write the pseudocode of the main operations of the given data structure”

1. Program

“Paste the screenshot of the executed python code”

1. Presentation of the results

“Paste the output of the program”

1. Analysis and discussions

“Discuss the time complexities of all the operations of the given data structure”

# Experiment 9

**Title of the Laboratory Exercise**: Quick Sort

**1. Aim:**

To implement Quick Sort Algorithm using Python

**2. Objective:**

1. To understand the concept of Quick Sort Algorithm
2. To learn how to implement Quick Sort Algorithm using Python
3. To analyze the time complexity of Quick Sort Algorithm

**3. Exercise:**

In this exercise, you will implement Quick Sort Algorithm using Python. Follow the steps below:

**Step 1:** Write a function called quick\_sort that takes an array of integers as input and returns a sorted array.

**Step 2:** Implement the Quick Sort Algorithm. The steps of the Quick Sort Algorithm are as follows:

i. Choose a pivot element from the array (can be the first or last element).

ii. Partition the array into two subarrays: one with elements less than or equal to the pivot, and one with elements greater than the pivot.

iii. Recursively sort the two subarrays.

**Step 3:** Test your implementation using a test case that includes a list of 10 unsorted integers.

**Step 4:** Analyze the time complexity of Quick Sort Algorithm.

**Step 5:** Submit your code along with a brief explanation of the Quick Sort Algorithm and its time complexity analysis.

Note: You can use the time module in Python to measure the time taken by your quick\_sort function to sort an array.

**4. Experimental Procedure**

1. Algorithm design

“Write the pseudocode of the main operations of the given sorting technique”

1. Program

“Paste the screenshot of the executed python code”

1. Presentation of the results

“Paste the output of the program”

1. Analysis and discussions

“Discuss the time complexities of all the operations of the given sorting technique”