**5 practicals to be performed in lab :**

1. **binary search**
2. **stack operations( push, pop,display)**
3. **heap sort**
4. **fractional knapsack**
5. **merge sort**

**note : fractional knapsack ,heap sort, merge sort in python**

**binary search and stack in c++**

**Compiling a C++ program:**

The basic command to compile a C++ source file (.cpp) is:

**Code**

**g++ your_program.cpp -o executable_name**

g++: Invokes the C++ compiler.

your_program.cpp: The name of your C++ source code file.

-o executable_name: This option specifies the name of the executable output file. If you omit this, the default executable name will be a.out.

Example:

If you have a file named hello.cpp, you can compile it with:

**Code**

**g++ hello.cpp -o hello**

This will create an executable file named hello in the same directory.

**Running the compiled program:**

Once the program is compiled, you can run the executable using the following command:

**Code**

**./executable_name**

./: This indicates that the executable is located in the current directory.

Example:

To run the hello executable created in the previous step:

**Code**

**./hello**

This will execute your C++ program and display its output in the terminal**.**

**To execute a Python script in Linux:**

Using the python or python3 command.

**Code**

**python3 your_script_name.py**

(Use python if python3 is not the default or if you specifically need Python 2.**)**

binary search

```
#include <iostream>
int binarySearch(int arr[], int size, int target) {
    int low = 0;
    int high = size - 1;
    while (low <= high) {
        int mid = low + (high - low) / 2;
        if (arr[mid] == target) {
            return mid; // Return the index
        }

        else if (arr[mid] < target) {
            low = mid + 1;
        }
        else {
            high = mid - 1;
        }
    }
    return -1; // Target not found
}
```

```
int main() {
    int sortedArray[] = {2, 5, 8, 12, 16, 23, 38, 56, 72, 91};
    int size = sizeof(sortedArray) / sizeof(sortedArray[0]);
    int targetElement = 23;

    int result = binarySearch(sortedArray, size, targetElement);

    if (result != -1) {
        std::cout << "Element " << targetElement << " found at index: " << result <<
std::endl;
    } else {
        std::cout << "Element " << targetElement << " not found in the array." <<
std::endl;
    }

    return 0;
}
```

heap sort

```
def heapify(arr, n, i):
    # Find largest among root and children
    largest = i
    l = 2 * i + 1
    r = 2 * i + 2

    if l < n and arr[i] < arr[l]:
        largest = l

    if r < n and arr[largest] < arr[r]:
        largest = r

    # If root is not largest, swap with largest and continue heapifying
    if largest != i:
        arr[i], arr[largest] = arr[largest], arr[i]
        heapify(arr, n, largest)
```

```
def heapSort(arr):
    n = len(arr)

    # Build max heap
    for i in range(n//2, -1, -1):
        heapify(arr, n, i)

    for i in range(n-1, 0, -1):
        # Swap
        arr[i], arr[0] = arr[0], arr[i]

        # Heapify root element
        heapify(arr, i, 0)
```

```
arr = [1, 12, 9, 5, 6, 10]
 heapSort(arr)
 n = len(arr)
 print("Sorted array is")
 for i in range(n):
     print("%d " % arr[i], end='')
```

**Heap Sort Complexity**

**Time Complexity**

**Best O(nlog n)**

**Worst O(nlog n)**

**Average O(nlog n)**

**Space Complexity O(1)**

fractional knapsack

```
class Item:
    def __init__(self, weight, value):
        self.weight = weight
        self.value = value
        self.ratio = value / weight

def fractional_knapsack(items, capacity):
    items.sort(key=lambda x: x.ratio, reverse=True)

    total_value = 0
    for i in items:
        if capacity >= i.weight:
            capacity -= i.weight
            total_value += i.value
        else:
            fraction = capacity / i.weight
            total_value += i.value * fraction
            break
    return total_value
```

```
# Test the function
items = [Item(20, 100), Item(30, 120), Item(10, 60)]
capacity = 50
print(fractional_knapsack(items, capacity))  # Output: 240.0
```

# viva questions

1. types of data staructure
2. what is algorithm and pseudocode
3. what is time and space complexity
4. linear search versus binary search
5. time complexity of binary search
6. what is bubble sort, insertion, selection sort(difference)
7. sort using bubble sort,insertion,selection,merge,heap
8. introduction to stack, linear queue, circular queue
9. difference between linear/circular queue
10. stack is linear/non linear data structure?
11. enqueue and dequeue of circular queue( algo/pseudocode)
12. what is binary search tree? time complexity of binary search tree
13. how insertion deletion performed on binary search tree
14. what is min spanning tree
15. difference between kruskal and prims
16.

16. what is divide and conquer
17. what is merge sort
what is the difference between greedy and dynamic algo
examples of greedy and dynamic algorithm
greedy: fractional knapsack, dynamic is 0/1 knapsack
what is ai search algorithms?
what is naive string matching?