Ayush Ranjan

BT22GCS 059

1. You have an array containing known set of numbers {1,2,3}. Example array:

   1,2,1,3,1,1,2,2,3,1,2...

   Write an efficient code to sort these numbers in O(N) time.

```python
def counting_sort(arr):

    count = [0, 0, 0]

    for i in arr:
        count[i - 1] += 1

    sorted_arr = []

    for i in range(3):
        sorted_arr.extend([i + 1] * count[i])

    return sorted_arr

arr = [1, 2, 1, 3, 1, 1, 2, 2, 3, 1, 2]
sorted_arr = counting_sort(arr)

print(sorted_arr)
```

2. You have an array containing only zeros and ones. Data is available in chunks. Sort the array in O(N) time. Given array:

    00000000000000111111111111111111000000000000000011111111111000000001111111111
    ...

```
def linear_search (arr):
    arr=[0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1]
    zeros=0
    ones=0

    for i in arr:
        if i==0:
            zeros+=1
            i+=1
        else:
            ones+=1


    if zeros+ones == len(arr):
        for i in range(zeros):
            arr[i]=0

        for i in range(zeros,zeros+ones):
            arr[i]=1

        return arr

input_arr=[0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1]
arr=linear_search(input_arr)
print(arr)
```

3. Consider a list of 10 numbers. Answer the following questions with reference to the same.
   Compare linear search and binary search in terms of best case, average case and worst case time taken. Hint:

Linear Search	Binary Search
Best case time taken: When the beast case occurs:	Best case time taken: When the beast case occurs:
Average case time taken: How you find the average case:	Average case time taken: How you find the average case:
Worst case time taken: When the worst case occurs:	Worst case time taken: When the worst case occurs:

```python
def linear_search(arr, g):
    count = 0
    for i in range(len(arr)):
        count += 1
        if arr[i] == g:
            return f"Element found at index {i}", count
    return "Element not found", count

# Example usage:
unsorted_list = [9, 4, 2, 7, 1, 5, 8, 3, 6, 10]
element_to_find = 7
result, linear_search_count = linear_search(unsorted_list, element_to_find)
print(result)
print(f"Linear Search Count: {linear_search_count}")

```

Linear Search:

Worst Case: When looking for an item at the end of an unsorted list, linear search checks every item before finding the desired one.
Best Case: Even in the best case (when the item is at the start), linear search may still need to check the entire list.
Average Case Time: On average, linear search takes longer as it needs to check each item, resulting in a time proportional to the list's length (linear time).

Binary Search:

Worst Case: If the list is in reverse order, binary search takes a long time as it repeatedly checks the entire list in each step before finding the item.

Best Case: Both binary search and merge sort work well when the item is at the beginning of a sorted list, taking very little time (constant time).
Average Case Time: Binary search is faster on average because it splits the search space in half with each step, resulting in a quicker search (logarithmic time).

```python
def binary_search(arr, x):
    low = 0
    high = len(arr) - 1
    mid = 0

    while low <= high:
        mid = (high + low) // 2

        if arr[mid] < x:
            low = mid + 1
        elif arr[mid] > x:
            high = mid - 1
        else:
            return mid

    return -1

# Testing the code
sorted_list = [1, 2, 3, 4, 5, 7, 8, 9, 11, 12]
search_for = 5

count = 0
index = binary_search(sorted_list, search_for)
count += 1

if index != -1:
    print(f"Element {search_for} found at index {index}")
else:
    print(f"Element {search_for} not found in the list")
```

4. Modify the partition() function in quicksort to implement the following. Take a list as input. This list contains the firstname, lastname and date of birth of n persons. Priority rule for a person:

(i) First letter of firstname (ii) First letter of lastname (iii) Date of birth.

In case of clash we look for second rule and if there is a clash then look for the third rule. Print the position of the chosen person in the list according to his/her priority. Show the best case, worst case and average case time taken for 5 element list.

Example:

Input Anil Kumar 04/04/2000, Amit Kumar 04/04/2001, Asraf Seikh 08/12/2008

Position of Asraf Seikh is 3

```python
def custom_comparison(person1, person2):
    if person1[0][0] < person2[0][0]:
        return -1
    elif person1[0][0] > person2[0][0]:
        return 1
    if person1[1][0] < person2[1][0]:
        return -1
    elif person1[1][0] > person2[1][0]:
        return 1
    if person1[2][0] < person2[2][0]:
        return -1
    elif person1[2][0] > person2[2][0]:
        return 1
    return 0
def partition(a, p, r):
    pivot = a[r]
    i = p - 1
    for j in range(p, r):
        if custom_comparison(a[j], pivot) == -1:
            i += 1
            a[i], a[j] = a[j], arr[i]
    a[i + 1], a[r] = a[r], a[i + 1]
    return i + 1
def quicksort(a, p, r):
    if low < high:
        pivot_index = partition(a, p, r)
        quicksort(a, p, pivot_index - 1)
        quicksort(a, pivot_index + 1, high)
person = []
n = int(input("Enter the number of inputs: "))
for i in range(n):
    First_Name = str(input(f"Enter the first name of person {i + 1}: "))
    Last_Name = str(input(f"Enter the last name of person {i + 1}: "))
    DOB = input(f"Enter the dob of person {i + 1} in the format YYYY-MM-DD: ")
    person.append((First_Name, Last_Name, DOB))
quicksort(person, 0, len(person) - 1)
print("Sorted List:")
for i, person in enumerate(person):
    print(f"Position {i + 1}: {person[0]} {person[1]}, DOB: {person[2]}")
```

5. Modify Bellman Ford algorithm to find longest path from the given source to the destination via a given node. Analyze the time taken for the code written by you.

```
def longest_path(g, start, destination, via):
    vertices = len(g)
    dist = [0] * vertices
    path = [''] * vertices

    for i in range(vertices):
        updated = False
        for u in range(vertices):
            for v, weight in enumerate(g[u]):
                if weight != 0 and dist[u] != float('inf') and dist[u] + weight > dist[v]:
                    dist[v] = dist[u] + weight
                    path[v] = path[u] + str(v)
                    updated = True
        if not updated:
            break

    longest_path = path[destination]
    return longest_path if longest_path != '' else 'No Longest Path Found'

# example usage:

g = [
    [0, 2, 0, 6, 0],
    [0, 0, 3, 8, 5],
    [0, 0, 0, 0, 7],
    [0, 0, 0, 0, 9],
    [0, 0, 0, 0, 0]
]

print(longest_path(g, 0, 4, 2))
```

6. You need to create a tertiary tree using linked list. Each node in this tree stores an alphabet (upper case only). You need to check that this tree is a max-heap or not. Show the best-case, worst-case and average case time taken for 10 element list.

```python
class TernaryTreeNode:
    def abc(self, value):
        self.value = value
        self.left = None
        self.middle = None
        self.right = None

def max_heap(root):
    if root is None:
        return True

    if root.left:
        if root.left.value > root.value:
            return False

    if root.middle:
        if root.middle.value > root.value:
            return False

    if root.right:
        if root.right.value > root.value:
            return False

    return max_heap(root.left) and max_heap(root.middle) and max_heap(root.right)


node1 = TernaryTreeNode('A')
node2 = TernaryTreeNode('B')
node3 = TernaryTreeNode('C')
node4 = TernaryTreeNode('D')
node5 = TernaryTreeNode('E')
node6 = TernaryTreeNode('F')
node7 = TernaryTreeNode('G')
node8 = TernaryTreeNode('H')
node9 = TernaryTreeNode('I')
node10 = TernaryTreeNode('J')

node1.left = node2
node1.middle = node3
node1.right = node4
node2.left = node5
node2.middle = node6
node2.right = node7
node3.left = node8
node3.middle = node9
node3.right = node10

is_heap = is_max_heap(node1)
print(f"Is the tree a max-heap? {is_heap}")
```

7. Implement merge sort and quick sort. Analyse best case, worst case and average case.

```python
def merge_sort(arr):
    if len(arr) <= 1:
        return arr

    mid = len(arr) // 2
    left_half = arr[:mid]
    right_half = arr[mid:]

    return merge(merge_sort(left_half), merge_sort(right_half))

def merge(left, right):
    merged = []
    left_index = 0
    right_index = 0

    while left_index < len(left) and right_index < len(right):
        if left[left_index] < right[right_index]:
            merged.append(left[left_index])
            left_index += 1
        else:
            merged.append(right[right_index])
            right_index += 1

    merged.extend(left[left_index:])
    merged.extend(right[right_index:])

    return merged

print(merge_sort([3, 2, 5, 1, 4]))
```

```python
def quick_sort(arr):
    if len(arr) <= 1:
        return arr

    pivot = arr[len(arr) // 2]
    left = [x for x in arr if x < pivot]
    middle = [x for x in arr if x == pivot]
    right = [x for x in arr if x > pivot]

    return quick_sort(left) + middle + quick_sort(right)

print(quick_sort([3, 2, 5, 1, 4]))
```

8. You have a unsorted list of ten numbers. Which searching algorithm you are going to apply. You have two options (i) Linear search (ii) Sorting followed by binary search. (You can use either bubble , selection or insertion sort). If you choose binary search which sorting algorithm you choose and why? Write codes with count variable for each case.

```python
def binary_search(arr, x):
    low = 0
    high = len(arr) - 1
    mid = 0

    while low <= high:
        mid = (high + low) // 2

        if arr[mid] < x:
            low = mid + 1
        elif arr[mid] > x:
            high = mid - 1
        else:
            return mid

    return -1

# Testing the code
sorted_list = [1, 2, 3, 4, 5, 7, 8, 9, 11, 12]
search_for = 5

count = 0
index = binary_search(sorted_list, search_for)
count += 1

if index != -1:
    print(f"Element {search_for} found at index {index}")
else:
    print(f"Element {search_for} not found in the list")
```