Name: Mugilkrishna D U
Reg. No.: 3122225001073

# UCS 2403 Design & Analysis of Algorithms

## Assignment 4

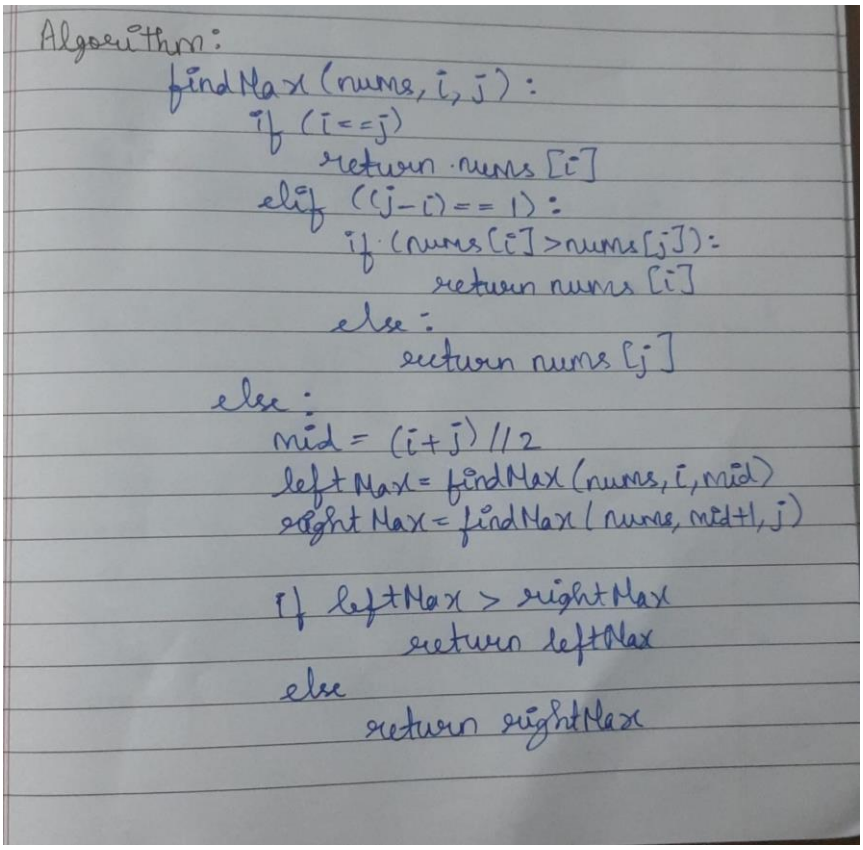
**Date of Exercise: 21.03.2024**

**Aim:** To gain understanding and proficiency on the divide and conquer strategy

**Question 1:**

Finding MAX using divide-and-conquer: Using the technique of divide-and-conquer, write a recursive program to find the maximum value in a given (unsorted) list of numbers. Write the recurrence relation to find the time complexity of the algorithm. Find a closed form expression for the time complexity. Do NOT use built-in Python methods for finding MAX.

**Algorithm:**



```
Algorithm:
    findMax(nums, i, j):
        if (i==j)
            return nums[i]
        elif ((j-i)==1):
            if (nums[i]>nums[j]):
                return nums[i]
            else:
                return nums[j]
        else:
            mid = (i+j)//2
            leftMax = findMax(nums, i, mid)
            rightMax = findMax(nums, mid+1, j)

            if leftMax > rightMax
                return leftMax
            else
                return rightMax
```

**Code:**

```
# Input list
size = int(input("Enter the size of the List: "))
nums = []

for k in range(size):
    nums.append(int(input("Enter element " + str(k+1) + ": ")))
```

Name: Mugilkrishna D U
Reg. No.: 3122225001073

```
print("\nList:", nums)

# Finding MAX using divide-and-conquer
def findMax(nums, i, j):
    if (i == j):
        return nums[i]
    elif ((j - i) == 1):
        if (nums[i]>nums[j]):
            return nums[i]
        else:
            return nums[j]
    else:
        mid = (i + j) // 2
        leftMax = findMax(nums, i, mid)
        rightMax = findMax(nums, mid+1, j)

        if (leftMax > rightMax):
            return leftMax
        else:
            return rightMax

print("The maximum of the given list is: ", findMax(nums, 0, size-1))
```

**Output:**

```
PS C:\Users\Mugilkrishna D U\OneDrive\Desktop\My Files\.vscode> & "C:/Users/Mugilkrishna D U/AppDat
a/Local/Programs/Python/Python311/python.exe" "c:/Users/Mugilkrishna D U/OneDrive/Desktop/My Files/
SSN/SEM4/DESIGN AND ANALYSIS OF ALGORITHMS/LAB/4.1.py"
Enter the size of the List: 6
Enter element 1: 3
Enter element 2: 4
Enter element 3: 2
Enter element 4: 1
Enter element 5: 6
Enter element 6: 5

List: [3, 4, 2, 1, 6, 5]
The maximum of the given list is:  6
PS C:\Users\Mugilkrishna D U\OneDrive\Desktop\My Files\.vscode> & "C:/Users/Mugilkrishna D U/AppDat
a/Local/Programs/Python/Python311/python.exe" "c:/Users/Mugilkrishna D U/OneDrive/Desktop/My Files/
SSN/SEM4/DESIGN AND ANALYSIS OF ALGORITHMS/LAB/4.1.py"
Enter the size of the List: 5
Enter element 1: 3
Enter element 2: 4
Enter element 3: 5
Enter element 4: 6
Enter element 5: 7
List: [3, 4, 5, 6, 7]
The maximum of the given list is:  7
PS C:\Users\Mugilkrishna D U\OneDrive\Desktop\My Files\.vscode>
```

**Time Complexity:** O(n) – Linear Time Complexity

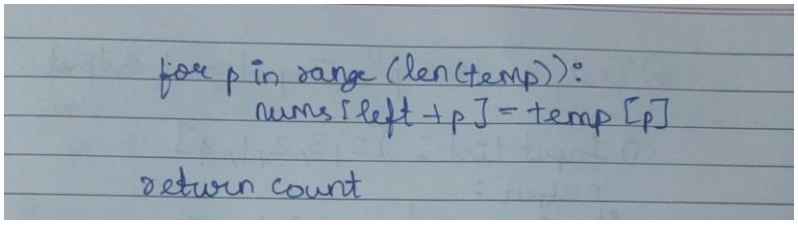
**Question 2:**

Mergesort to count inversions: Modify the algorithm of Mergesort to count inversions in a given list. Compare the time complexity of this algorithm against the time complexity of the code you wrote in Assignment 3 to compute the count of inversions.

**Algorithm:**

```
Algorithm:
    count_inversions(nums):
        return merge_sort(nums, 0, len(nums)
                                    -1)

    merge_sort (nums, left, right):
        count → 0
        if left < right:
            mid → (left + right) /2
            count += mergesort (nums, left, mid)
            count += merge_sort (nums, mid+1, right)
            count += merge (nums, left, mid,
                                    right)
        return count

    merge (nums, left, mid, right)
        count → 0
        i → left, j → mid+1, k → 0

        while i <= mid and j <= right
            if nums [i] <= nums [j]
                temp[k] ⇒ nums [i]
                k → +1
                i → +1
            else:
                temp [k] → nums [j]
                k → +1, j → +1
                count → + mid - i + 1

        while i <= mid
            temp [k] = nums [i]
            k → +1, i → +1
        while j <= right
            temp [k] = nums [j]
            k → +1, j → +1
```

Name: Mugilkrishna D U
Reg. No.: 3122225001073




**Code:**

```
#INVERSIONS IN A LIST

#Input list
size = int(input("Enter the size of the List: "))
nums = []

for k in range(size):
    nums.append(int(input("Enter element " + str(k+1) + ": ")))

print("\nList:", nums)

#Time Complexity: O(n*logn)
#Mergesort algorithm

def count_inversions2(nums):
    return merge_sort(nums, 0, len(nums) - 1)

def merge_sort(nums, left, right):
    count2 = 0
    if left < right:
        mid = (left + right) // 2
        count2 += merge_sort(nums, left, mid)
        count2 += merge_sort(nums, mid + 1, right)
        count2 += merge(nums, left, mid, right)
    return count2

def merge(nums, left, mid, right):
    count2 = 0
    temp = [0] * (right - left + 1)
    i = left
    j = mid + 1
    k = 0

    while i <= mid and j <= right:
        if nums[i] <= nums[j]:
            temp[k] = nums[i]
            k += 1
            i += 1
```

```
        else:
            temp[k] = nums[j]
            k += 1
            j += 1
            count2 += mid - i + 1

    while i <= mid:
        temp[k] = nums[i]
        k += 1
        i += 1

    while j <= right:
        temp[k] = nums[j]
        k += 1
        j += 1

    for p in range(len(temp)):
        nums[left + p] = temp[p]

    return count2
print("Number of inversions using mergesort algorithm is: ",
count_inversions2(nums))
```

**Output:**

```
PS C:\Users\Mugilkrishna D U\OneDrive\Desktop\My Files\.vscode> & "C:/Users/Mugilkrishna D U/AppDat
a/Local/Programs/Python/Python311/python.exe" "c:/Users/Mugilkrishna D U/OneDrive/Desktop/My Files/
SSN/SEM4/DESIGN AND ANALYSIS OF ALGORITHMS/LAB/4.2.py"
Enter the size of the List: 6
Enter element 1: 3
Enter element 2: 4
Enter element 3: 1
Enter element 4: 2
Enter element 5: 7
Enter element 6: 5

List: [3, 4, 1, 2, 7, 5]
Number of inversions using mergesort algorithm is:  5
PS C:\Users\Mugilkrishna D U\OneDrive\Desktop\My Files\.vscode> & "C:/Users/Mugilkrishna D U/AppDat
a/Local/Programs/Python/Python311/python.exe" "c:/Users/Mugilkrishna D U/OneDrive/Desktop/My Files/
SSN/SEM4/DESIGN AND ANALYSIS OF ALGORITHMS/LAB/4.2.py"
Enter the size of the List: 4
Enter element 1: 4
Enter element 2: 3
Enter element 3: 7
Enter element 4: 8

List: [4, 3, 7, 8]
Number of inversions using mergesort algorithm is:  1
```

**Time Complexity:** O(n*logn) – Linear Logarithmic Time Complexity

**Question 3:**

Finding the Maximum Subarray Sum: Given a list A of size n, find the sum of elements in a subset A' of A such that the elements of A' are contiguous and has the largest sum among all such subsets. Please note that:

- the subset should be having elements that are contiguous in the original list.
- the input list may have negative values.
- the algorithm should be based on divide and conquer strategy.

Example:
Input: A = [-2,1,-3,4,-1,2,1,-5,4]
Output: 6

Write the recurrence relation for the time complexity of your algorithm, and find a closed form expression for the same.

**Algorithm:**

Algorithm:

```
max_crossing_subarray (arr, low, mid, high)
    left_sum → float ('-inf')
    sum → 0
    for i → range (mid, low-1, -1)
        sum → + arr[i]
        if sum > left_sum
            left_sum → sum

    right_sum → -∞, sum → 0
    for i → range (mid+1, high+1)
        sum → + arr[i]
        if sum > right_sum
            right_sum → sum
    return left_sum + right_sum

max_subarray_sum (arr, low, high)
    if low → high
        return arr[low]
    mid → (low+high) // 2

    left_sum → max_subarray_sum (arr, low, mid)
    right_sum → max_subarray_sum (arr, low, mid)
    cross_sum → max_crossing_subarray (arr, low, mid, high)
    return max(left_sum, right_sum, cross_sum)
```

Name: Mugilkrishna D U
Reg. No.: 3122225001073

**Code:**

```
#Input list
size = int(input("Enter the size of the List: "))
nums = []

for k in range(size):
    nums.append(int(input("Enter element " + str(k+1) + ": ")))

print("\nList:", nums)

#Maximum SubArray sum
def max_crossing_subarray(arr, low, mid, high):
    left_sum = float('-inf')
    sum = 0
    for i in range(mid, low - 1, -1):
        sum += arr[i]
        if sum > left_sum:
            left_sum = sum

    right_sum = float('-inf')
    sum = 0
    for i in range(mid + 1, high + 1):
        sum += arr[i]
        if sum > right_sum:
            right_sum = sum

    return left_sum + right_sum

def max_subarray_sum(arr, low, high):
    if low == high:
        return arr[low]

    mid = (low + high) // 2

    left_sum = max_subarray_sum(arr, low, mid)
    right_sum = max_subarray_sum(arr, mid + 1, high)
    cross_sum = max_crossing_subarray(arr, low, mid, high)

    return max(left_sum, right_sum, cross_sum)

print("Maximum Subarray Sum:", max_subarray_sum(nums, 0, len(nums) - 1))
```

**Output:**

```
PS C:\Users\Mugilkrishna D U\OneDrive\Desktop\My Files\.vscode> & "C:/Users/Mugilkrishna D U/AppDat
a/Local/Programs/Python/Python311/python.exe" "c:/Users/Mugilkrishna D U/OneDrive/Desktop/My Files/
SSN/SEM4/DESIGN AND ANALYSIS OF ALGORITHMS/LAB/4.3.py"
Enter the size of the List: 5
Enter element 1: 2
Enter element 2: 3
Enter element 3: 2
Enter element 4: 1
Enter element 5: 4

List: [2, 3, 2, 1, 4]
Maximum Subarray Sum: 12
PS C:\Users\Mugilkrishna D U\OneDrive\Desktop\My Files\.vscode> & "C:/Users/Mugilkrishna D U/AppDat
a/Local/Programs/Python/Python311/python.exe" "c:/Users/Mugilkrishna D U/OneDrive/Desktop/My Files/
SSN/SEM4/DESIGN AND ANALYSIS OF ALGORITHMS/LAB/4.3.py"
Enter the size of the List: 6
Enter element 1: -2
Enter element 2: -1
Enter element 3: 4
Enter element 4: -2
Enter element 5: 2
Enter element 6: 1

List: [-2, -1, 4, -2, 2, 1]
Maximum Subarray Sum: 5
PS C:\Users\Mugilkrishna D U\OneDrive\Desktop\My Files\.vscode>
```

**Time Complexity:** O(n*logn) – Linear Logarithmic Time Complexity

**Learning Outcome:**
Upon completing this exercise, I have understood the importance of the divide and conquer algorithm and I have learnt to maximise their efficiency and how to find the time complexities. I have also learnt to write and calculate the recurrence relation and the closed form expressions.