# Data Structures and Algorithms (Lab 3)

# Submitted by: Afraz Butt , 2021-CS-12

# Submitted to: Mr. Samyan Wahla

# Problem 1:

Collaborated with Muhammad Hamza Saleem, 2021-CS-41.

## Part(vi):

### (a):

The logic is correct. Karatsuba’s Algorithm will work in any base , by recursively multiplying parts.

### (b):

The logic is correct. The length of a number x’*s* base-b representation decreases as b *increases*.

### (c):

The logic is incorrect. Because Karatsuba’s algorithm works by 3 recursive calls. Even if the addition and subtraction were to have O(1) time, which they clearly don’t, the 3 recursive calls would still dominate most of the time, and hence the total time would remain O(n1.58).

# Problem 2:

## Part (I):

### Pseudocode:

Function friendSlower(Input):

Let temp [Input.length] be the new array

k = 0

for i:1 to n:

for j: i+1 to n:

if Input[i][0] in Input[j]:

temp[k] = (i , j )

k = k + 1

end for

end for

### English Description:

The algorithm is a brute force algorithm. It works by picking up an element form the array and then comparing it with all the values of the array ahead of it. This occurs by generating a list of numbers in the next element and then checking if the previous element’s first or last number was present in that list.

### Brief Runtime Analysis:

Since two for loops are involved in the comparison , the total worst case runtime would be O(n2).

## Part(II):

### Pseudocode:

Function friendsFaster(Input):

Let temp[Input.length] be the new array

Sorted = MergeSort(Input,0,Input.length)

for i:1 to n, i = i + 2:

if Input[i][0] in Input[i + 1]:

temp[k] = (i , i + 1 )

k = k + 1

### English Description:

The algorithm is a modified version of the previous part. Instead of comparing with all possible combinations, we are just checking with the immediate successor, and then increasing the loop count by 2 and doing the same thing in a single iteration.

### Runtime Analysis:

The for loop, since having an increment counter of two, has a runtime of *lg*n .The Merge Sort algorithm takes n*lg*n time. So The total worst case time of the algorithm will be O(n*lg*n).

# Problem 3:

Collaborated with Muhammad Hamza Saleem, 2021-CS-41 and Shahzaib Irfan, 2021-CS-7.

## Part(a):

In a nested for loop, the total toad-to-toad comparison occurs in a brute force manner. Two toads are compared and the answer is checked with the truth value of the first toad. If true, then the index of the first toad is appended into the list and returned.

## Part(b):

Using half comparisons in a single loop instead of a nested loop, the toads are checked in a n/2 manner. The method of comparison remains the same, except for the difference that only the next immediate toad is checked with (it is enough to get trustworthiness, *per se* ). If the total output of trustworthy toads m is less than m/2 for range 0 < m ≤ n/2 , the list returned is empty , else the list is definitely returned with indexes, satisfying the constraints.

### Part(c):

Using half comparisons in a single loop instead of a nested loop, the toads are checked in a dn/2e manner. The method of comparison remains the same, except for the difference that only the next immediate toad is checked with (it is enough to get trustworthiness, *per se* ). If the total output of trustworthy toads m is less than m/2 for range 0 < m ≤ dn/2e , the list returned is empty , else the list is definitely returned with indexes, satisfying the constraints. dn/2e represents differential for odd values.

### Part(d):

Let’s say that this *single trustworthy toad* has the name Tiffany. Then to find Tiffany , we would obviously use a searching algorithm super imposed with comparisons. Having the list of indices, we will use a divide and conquer paradigm to get the indices in broken manner and then *fetch* each relevant index at its position. In recursive algorithim, this would take the form of base case at array length 1 returning the value, else splitting the array with one starting value gone and calling the function again from the next value.

## Part(e):

### Inductive hypothesis:

Recursive algorithm breaks down the array skipping the first element. The search continues in the updated array.

### Base case:

If array.length is 0 , return array[0].

### Inductive step:

// Also called the recursive step.

index = 1 + search(k + 1 , n)

### Conclusion:

Index is found.

## Part(f):

Since all values in the array are being checked, in the worst case, every value will be checked. So the algorithm has O(n) time.

### Part(g):

A loop iterating over all values of toads to find trustworthy.