Assignment 4 – Harshit Mawandia

Q1. Making a Calculator:

Our main function is *evaluate* and we use 3 helper functions- *readParen, readNumber. findParen*.

4. Correctness Proof:

findParen:

It iterates over length of string till it finds a ')' which gives us the innermost brackets in which case it returns index of last '(' and first ')', if brackets exist, it returns -1.

readNumber:

It iterates from the i to length of string till it either encounters a non numeric digit or reaches end of string. And returns the number according to assertions given in the comments.

readParen:

It evaluates the value of the expression inside a parenthesis using *readNumber* and also finding the operator between two numbers.

evaluate:

It uses *readParen* recursively to find the values of the expression till it no longer has a bracket, and then evaluates the final expression in the same manner as *readParen*.

Time Complexity Analysis:

Let N be the length of expression s

findParen:

Its time complexity is O(N) in the worst case

readNumber:

Its time complexity can also be O(N) in the worst case

readParen:

Its time complexity is O(*readNumber*()) + O(*findParen*()), so it's also O(N).

evaluate:

Its time complexity is O(n*O(*readParen*)) = O(n^2) in the worst case

Assignment 4 – Harshit Mawandia

Q2. A sequence of unique sums:

Our main function is *sumSequence*, we use 2 helper functions check1 and *findNext*.

2. Correctness Proof:

check(q,l):

It checks whether the number q can be written as sum of 2 distinct numbers in the list l, if yes, then if it possible to write in more than one ways or not.

We use an outer loop which holds the index of 1 number and inner loop which holds the index of second number, we iterate I from 0 to length l and j from i+1 to length l so that no digit is used twice. And check if the sum of the elements is equal to q in which case we increment b by 1. If b >1 then we break the loop since we don't need any more information.

findNext(l):

It finds the next element in the sequence in the list l. keeps incrementing a from the last element of list l +1 till it finds a number for which check returns 1 which is the next element of the list.

sumSequence(n):

it returns the sequence upto n elements in the list using findNext if n>2.

3. Time complexity analysis:

check(q,l):

Its time complexity will be O(n^2) in the worst case where it iterates both outer and inner loop totally.

Findnext(l):

Its iterates from second last element to last element and calls check time. If second last element is l and last element is m then its time complexity is (l-m)*n^2.

sumSequence(n):

It calls *findNext*(l) n times for which *findNext* calls *check*(q,l) for values of q going from 3 to m in total. So net time complexity of our program will be O(mn^2).

Assignment 4 – Harshit Mawandia

Q3. Shortest sublist with sufficient sum

Our main function is *minLength*. We have not used any helper functions.

2. Correctness Proof:

Let's say l is the length of the list given to us.

We first set the value of lmin to l+1 because any sublist will be of length<=l.

Now we use outer for loop to iterate from 0 to l and variable i will store the value of the index from which we start the sublist.

The inner loop iterates from i to l and stores the index at which we end our sublist.

We use a varuiable sum which we intitialise inside the outer loop to 0. It stores the sum of elements from i to j (inner loop counter) at any instant when the counter is inside the inner loop.

If at any moment sum>n then we check if the length of the subist whose sum we have checked is less than lmin, in which case we change the value to lmin to the length of the sublist.

Hence we will find the length of the shortest sublist with sufficient sum if it exists. In which case we will return lmin.

If the value of lmin hasn't changed from initial value which was l+1, we return -1

3. Time Complexity analysis:

Lets say the length of the list is N

The outer for loop goes from 0 to N while inner goes from i to N.

So time complexity will be:

$$\sum_{i=0}^{N}(N-i) = N^2 - \frac{N(N+1)}{2} = O(N^2)$$

Assignment 4 – Harshit Mawandia

Q4. Merging an contact List:

Our main function is mergeContacts. We have used 3 helper functions mergeSort, mergeIt, mergeAB.

2. Correctness Proof:

The 3 helper functions we have used in this program are standard merge sort functions from the lecture with a few minor changes, in which in place of comparing the entire elements of the list, we compare only the first element of the tuples inside the list for the sorting part, everything else remains the same.

Our main function mergeContacts(l):
Let N be the length of the list

It first sorts the list and stores it in a variable A, then it creates another variable b which stores a list with the first element of A as its only element.

Then it iterates over from 0 to N, and checks whether the first element of the i^{th} tuple is equal the $i\text{-}1^{th}$ tuple, if it is not, it just stores the i^{th} as another element in the list, else it appends the value of the last element of the tuple to the last element of b that is the list.

And then finally returns b.

3. Time Complexity:

Let the length of the list be n

Order of sorting the list using merge sort is $O(n\log(n))$.

Then we use a for loop iterating from 0 to n, which will be $O(n)$

So our total time complexity will be $O(n\log(n)) + O(n) = O(n\log n)$.