Akarsh Jain
2020CS10318
Assignment 4
CS1200318@iitd.ac.in

Problem 1

1, 2, 3 done in .py file.

4. Proof of correctness of isint.

Invariant: string does not contain integers from 0 to i-1

Proof.

Base case: i = 0. loop just starts and no return True/False statement till now.

Maintenance: Let it hold for i = j then when loop is executed, if string s contains 'j' then func. returns True and loop stops. If it continues ans was False and Hence invariant holds for i = j+1 also.

Termination: i = 9. Hence string does not contain '0'---'8' after execution of loop, it does not contain '0'---'9'. Which was the required output.

Proof for readNumber(s,i)

Invariant: s[i:j] contains a integer or a float and j ≠ a

Proof.

Base case: j = i+1. s[i:i+1] is the element at index i which must be a integer by function definition.

Maintenance: Let it be true for j = k then since at the time of execution of j = k+1 both conditions k+1 ≠ len(s) and isint(s[k+1]) are checked, Invariant is maintained.

Termination:

case 1: if j = a then clearly there are no further integer elements to be checked since we are at the end of the string. Hence this is correct output.

case 2: isint(s[j]) is False then the integer / float was only till index j-1 Hence we have to stop. This is required output.

Correctness of signeval is trivially established

Correctness of evalParen (s,i).

This works on the assumption that there are four meaningfull cases for inputs in a parenthesis.

case 1. x + y

Case 2. x + ( ) (sign may be + − / *)

case 3. ( ) + x

case 4. ( ) + ( )

There is no sense in giving input as (x) since use of parenthesis was then not required.

In cases where brackets appear recursive calls are made.

When there are integers/floats they are extracted using readNumber.

Since all cases are handled exhaustively code must be correct.

Proof of correctness of evaluate (s)

This code works on assumption that there can be two cases in input.

Case 1 — single number

Case 2 — Binary operation like x+y or x+( ) or ( )+x or ( )+( ) has to be performed.

For case 2 : brackets can be added on both sides and since evalParen is correct, ans can be obtained from evalParen (s, 0)

For case 1 : readNumber (s, 0) can be used.

Since all cases are exhaustively dealt with. The algorithm must be correct.

Problem (2)

1. Done in .py file

2. correctness of sumcheck(a,l)function

Invariant 1 (outer loop): count represents the number of ways x can be represented as a sum of two distinct numbers from l, one of the numbers constrained by index 0 to i-1.

invariant 2 (inner loop): count represents the number of ways x can be represented as a sum of two distinct numbers from l, one of the numbers constrained by index 0 to i-1 OR one with index i, other from i+1 to j-1

Proof for invariant 1, assuming 2 is correct

Base case: i=0 computation will start now Hence count = 0 is correct.

Maintenance: let it be true for i = x then by inv. 2 since it has checked for i = x+1, $\forall i<j<$ len(l)-1. It holds for i = x+1.

Termination: i = len(l) - 1. Hence after loop has been executed all different combinations of a+b a, b ∈ l would have been checked and count will contain the desired result.

correctness of nextterm(l)

invariant: all numbers between l[-1] and n including n cannot be part of sequence.

Proof by contradiction.

Suppose $l[-1] < y < n$ and y be part of sequence then sumcheck(y, l) must be true and the loop must not have been executed further due to condition in while statement. But it has to run till n which is contradi. Hence invariant is correct.

~~Correctness of~~

As soon as a number satisfies sumcheck(x, l) the loop will stop and that number will be output which was as required.

correctness of sumsequence(n) is trivially established since we require a list of size n Hence that is the condition in the while statement.

3. Time complexity

- Time comp. of sumcheck(a, l)

let len(l) = a

Now, time taken by 1 step of the inner most loop is O(1)

And it runs a - i - 2 times

Hence innermost loop is O(a - i)

outer loop runs from i = 0 to a - 1

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

Hence the function sumcheck(a, l) is $O(a^2)$

- time complexity of nextterm depends on how much later the next term of sequence is from previous term which cannot be explicitely counted.

Hence time in this is not explicitely mentioned but time taken here will be taken ~~it~~ into account directly in SumSequence function.

- Time complexity of sumSequence(n) function

n is the length of the list.

M is the last element of the list.

Now, to make a list of the sequence upto n terms, all integers from 1 to M have to be passed into sumcheck(n, l) and whenever it is True, nextterm sends that integer to sumSequence where it is appended (O(1)).

Since, the maximum length of list that can be passed into sumcheck is n. Hence max time it will take is $O(n^2)$ and since it is called M times, the final complexity will be $\boxed{O(Mn^2)}$.

(Note here that some constants have been ignored since they will not affect time complexity in Order notation. eg. loop is being called n-2 times but 2 can be ignored since Order of growth is important)

Problem 3

2. Proof of correctness for sumlist (l)

desired result = sum of all elements of the list input (l)

Invariant: Auxiliary variable sum = sum of all elements till index i of the list and $0 \leq i \leq len(l)-2$, (at start of loop)

Proof for invariant.

Base case: i = 0 then sum = l[0]

Maintenance: let it be true for some $0 \leq j < len(l)-2$ then we have to prove that it is true for j+1.

Now when i = j sum = l[0] ---- l[j]

loop executed l[j+1] added

and i = j+1.

Hence invariant is maintained.

termination: When $i = len(l)-2$, sum is the sum of elements till $(len(l)-2)^{th}$ index after loop execution till $(len(l)-1)^{th}$ index

Hence we obtain desired result from sum.

Correctness of min(a, b) is trivial.

Correctness of minLength(a, n)

Invariant 1 (for outer loop) = ans is the least length of a contigous list whose sum > n and starts from index 0 or 1 ... i-1 if such list exists.

invariant 2: ans is the least length of a contigous list whose sum > n and starts from index 0 or 1 ... i-1 or, starts at index i and is contained in a[i:j] if such list exists.

Proof for invariant 1 assuming invariant 2 holds correct.

Base case: i = 0 → i-1 = -1 Means loop not yet started. Hence, ans = len(a) + 2 is fine since it is greater than any realistic value and will help in detecting no such list exists case.

Maintenance: suppose invariant holds for some value i = x in the loop. Then by invariant 2 of inner loop, after execution of the loop it will also hold for i = x+1.

termination: $i = len(a) - 1$. Hence ans is least length till index $len(a) - 2$ and after execution of final loop till index $len(a) - 1$. Hence ans is least length for the entire list. This is what was required.

Proof for invariant-2 (inner loop)

Base case: $j = i$ in this the computation for loop will start and till this by invariant 1 ans is least length starting either from 0 or 1 or 2 ... $i-1$.

Maintenance: let invariant hold for $j = x$ then since in the subsequent loop we are checking for sublists $a[i, x+1]$ and reassigning the value to ans according to condition, invariant will hold for $j = x+1$ also.

termination: $j = len(a) - 1$. Hence we have checked till $len(a) - 2$ and subsequent loop we will have checked till index $len(a) - 1$. Hence we have reassigned value to ans, and checked condition for all $a[i, i+1]$ .... $a[i, len(a)]$. Hence we obtain the desired output from the loop.

3. Time complexity

let len(a) = n.

- the Time complexity of Min function is trivially O(1).

- time complexity of sumlist(l)

  Here if length of the list is n.

  in the loop sum is incremented by l[i+1] which is O(1)

  but since loop runs for i = 0 to n-1

  Hence Time complexity = O(n)

- time complexity of ~~Min length~~ Minlength(a,n)

  - first we find the time complexity of ~~one~~ step of the innermost loop

    there are 2 comparisions and sumlist a[i:j+1] is called which is Order(n)

    so time complexity is O(j-i)

  - Now for complete inner loop.

    Since j takes values from i to n with step being O(j-i)

    let n-i = y then

    Time comp. = $O(\sum_{z=0}^{y} y) = O(\frac{y(y+1)}{2}) = O(y^2)$

- Note for complete function.

$y = n - i$ so it takes value from 1 to n

each step being $O(y^2)$

Time comp. $= O\left(\sum_{1}^{n} y^2\right) = O\left(\frac{n(n+1)(2n+1)}{6}\right)$

$= O(n^3)$

Hence the function has $O(n^3)$ time complexity.

Problem 4

1. Done in .py file.

2. Proof of correctness

Proof for mergeContacts assuming others correct

invariant: emails have been merged for all people having entries till index i-1 and appended to ans.

Proof.

Base case: i = 1 so i-1 = 0 singleton list. 2nd element is converted to list also since duplicates cannot exist inv. holds.

Maintenance: let it hold for i = x. Then since while executing loop for i = x+1 we are checking names of l[i] and l[i-1] (list is sorted) and merging contacts if same. Hence it holds for i = x+1 also.

Termination: i = len(l) - 1. Hence after execution of the loop all the duplicates will be merged.

Proof for mergeSort(A)

invariant: A or B according to value of dir contain n/l sorted lists of size l each

(except last list) may be shorter

Proof.

Base case: $l = 1$. list of size 1 are sorted.

Maintenance: let it be true for $l = n$ then due to correctness of mergit function, it merge lists of length l each into size 2l

Hence it will hold for $l = 2n$

Termination: As soon as $l > n$ then the entire loop stop as the entire list was merged in the last step (directly follows from maintenance of invariant part of proof) which is the required output.

Correctness of mergeit (A, B, n, l)

invariant: all the elements till 2il have been copied into b as i sorted lists of length 2l each.

Base case: i=0, no element has been copied Hence correct.

Maintenance: let it be true for i=n then since now we merge list from left – left+l – right using mergeAB (which is assumed correct.) it will hold from i = n+1 also.

termination: if count was even then clearly in final iteration right=n and we would have completed task for full list.

but if count = odd then the last list would remain which is effectively copied by the next loop.

correctness of mergeAB(arr, b, l, m, r)

invariant = list b from index i to k is sorted

Proof

Base case = k = l - empty list is sorted. (k-1 = l-1)

Maintenance = let it be true for k = n then since we are checking both subarrays in subsequent iteration for smallest numbers and since both subarrays are sorted the minimum of first element of both will be the overall minimum. Hence the invariant holds for k = n+1 also.

Termination = since by the end we would have merged both subarrays into one sorted array by invariant, the result obtained is what was required.

- Proof for code of copying is left since it is trivial.

3. Time complexity

time complexity of mergesort(l) is O(n log n)
(done and dealt extensively in lectures explicitely)
where n is the length of the list.

Now I prove that subsequent steps in the mergeContacts(l) ~~algorithm~~ have lower complexity compared to mergesort(l) (1st step).

Time comp of loop.

inside the loop in one step, there are only assignment statements and append which are both O(1). Hence one step is O(1). But since loop runs n-1 iterations the time comp. of the loop is O(n)

Hence since the first step in O(nlogn) and subsequent are O(n).

The time comp. = O(nlogn)