Assignment2

Akarsh Jain

2020CS10318

cs1200318@iitd.ac.in

Please consider Problem 2,3,4 for normal marking, Problem 1 as bonus question.

Problem 1

1. $n > 5$ is assumed.

$$\text{isprime}(n) = \begin{cases} \text{false} & n \leq 1 \\ \text{prime\_iter}(n,2) & \text{otherwise} \end{cases}$$

(prime_iter is defined inside isprime in SML)

$$\text{prime\_iter}(n,y) = \begin{cases} \text{true} & y^2 > n \\ \text{false} & n \bmod y = 0 \\ \text{prime\_iter}(n,y+1) & \text{otherwise} \end{cases}$$

Here the invariant is that all numbers from $(2, 3 \ldots y-1)$ cannot divide n.

$$\text{Nextprime}(n) = \begin{cases} 3 & n = 2 \\ n+2 & \text{isprime}(n+2) \\ \text{Nextprime}(n+2) & \text{otherwise} \end{cases}$$

findPrimes(n) = pe(n,2,2)

(pe(n,y,z) is defined inside findPrimes in SML code)

$$pe(a,x,y) = \begin{cases} (x,y,a-x-y) & \text{isprime}(a-x-y) \\ (0,0,0) & x > a \text{ div } 2+1 \\ pe(a, \text{nextprime}(x), \text{nextprime}(x)) & 0 > a-x-y \\ pe(a, x, \text{nextprime}(y)) & \text{otherwise} \end{cases}$$

Proof of correctness.

1. isprime(n) evaluates to false whenever $n \leq 1$ which is obviously true and prime_iter(n,2) for other $n > 1$. Hence if prime_iter is proved, isprime is also true.

2. prime_iter(n,2) should evaluate to true if n is prime otherwise false to be correct.

invariant = x is not divisible by numbers between (2,3 ... y-1).

Base case: prime-iter(x,2)

For $x = 2,3$, $2^2 > x$ Hence it will evaluate true Hence correct.

For $x > 3$ $2^2 \neq x$. Hence if $x \bmod 2 = 0$ it will evaluate false which is correct.

otherwise prime-iter(x,3) in which it will

clearly imply that invariant holds.

Induction Hypothesis : Invariant holds for prime-iter(x, y)

To Prove: It will hold for (x, y+1) also.

Proof: If $y^2 > x$ then prime-iter(x, y) will evaluate to true which is correct due to Proof on previous values of y and Note given at last. (i.e. this would be the end case so to effectively prove this earlier cases will be considered where $y^2 < x$)

- If $x \bmod y = 0$ then it will evaluate to false correctly.
- And in the last case where $x \bmod y \neq 0$ it will call prime-iter(x, y+1) which is called noting $x \bmod y \neq 0$ Hence invariant Holds for (x, y+1) also.

Note: Algorithm will terminate surely since y is increasing by 1 and thus y^2 will eventually surpass any finite x.

Note: Algo. is stopped when $y > \sqrt{x}$ and not $y > x$ since if there are no factors of x from $[2, \lfloor\sqrt{x}\rfloor]$ there won't be any from $[\lfloor\sqrt{x}\rfloor, x]$
closed interval — partially closed

Proof: Suppose there is a factor of x say y in $[\lfloor\sqrt{x}\rfloor, x)$ then $\frac{x}{y}$ is Natural No. in $[2, \lfloor\sqrt{x}\rfloor]$ say z then $x = z \times y$. Hence, z is also a factor of x.

- Correctness of nextprime(n) is trivially established also since all primes except 2 are odd n is increased by 2 in each step It is bound to stop since ~~can~~ prime numbers are ~~not~~ bounded. #

This algorithm also does not miss any prime number since correctness of isprime(n+2) is already shown.

- To Prove correctness of pc(a, x, y) following argument is made

whenever a − x − y is prime then a can be represented as sum of 3 primes Hence first step is correct now we have to show that no case of triplets is missed. To avoid repeating cases, we ~~have~~ start increasing y from y = x since y < x would have been ~~the~~ already checked x was smaller. (Proof by contra.)

- first we fix and keep increasing y by nextprime(y) till 0 > a − x − y which means 3rd one cannot be prime. Once it has exhausted ~~the~~ for a particular x we increase x by nextprime(x) and start from y = x exhausting all cases for this x also.

use continues till $x = a \text{ div } 2 + 1$ after which 3rd no. i.e. $(a - x - y)$ would < 0 so we return $(0, 0, 0)$ in that case. Hence all cases are exhausted, Hence proved.

2. Time complexity (analysis may seem a little less formal since it is a bonus ques.)

Time complexity of isprime(n) function is $\sqrt{n}$ since it recursively checks all divisors till $\sqrt{n}$

$O(\sqrt{n})$

now for a particular pq(a, x, y) will atmost check isprime for "a" values since it stops on

$a - x - y < 0 \rightarrow a - x < y$

and x will vary from 2 to $a \text{ div } 2 + 1 < a$

Hence atmax a^2 no. of time isprime is checked since $a = n$

Hence Max time complexity $= n^2 \times \sqrt{n} = n^{2.5}$

Note: Here complexity due to number of additions, mod etc. is neglected as isprime is the $O(\sqrt{n})$ and all naive computations are $O(1)$. Hence their number won't matter much.

Space complexity is $O(1)$ since all calculations are done in one scope other functional calls also create some scopes but since they are also $O(1)$ in space complexity. Hence there will always be a constant no. of ~~scopes acti~~ maximum active scopes.

3. In SML.

Problem 2

1. $$Max(a,b) = \begin{cases} a & a \geq b \\ b & otherwise \end{cases}$$

$$MaximumValue(n, V, w, W) = \begin{cases} -V(n+1) & W<0 \\ 0 & n=0 \\ Max(MaximumValue(n-1, V, w, W), MaximumValue(n-1, V, w, W-w(n)) + V(n)) & otherwise \end{cases}$$

NOTE: Chronology of conditions is not arbitrary. If first is violated second must not be seen.

Here n represents number of items

V the value function

w the weigh function

W the max weight which should not be exceeded.

Note: explanation for $-V(n+1)$ when $W<0$

This is because according to problem statement, we do not have to exceed the weight W.

Let W' be weight before it was exceeded

so recursive call must have been

$\underbrace{MaximumValue(n'-1, V, w, W'-w(n'))}_{1} + \underbrace{V(n')}_{2}$

Now since this is wrong part 1 should evaluate to the negation of part 2 to nullify hence ~~etc~~.

$-V(n') = -V(n+1)$ since $n = n'-1$.

Proof of correctness.

1. The correctness of Max(a,b) is trivially established.
2. The goal of the MaximumValue function is to evaluate the maximum possible value a set of items can have without exceeding weight W.

Proof by Induction. (on n)

Base case: n = 0 then value must be 0 (True).

Induction Hypo: Let the function correctly calculate the desired result for n-1 elements

Proof: We have to show that it correctly evaluates for n elements also.

Case I $W < 0$

In this case either user has input negative W which is useless case and is neglected.
Or this is part of computation in n+1 elements.
-ve W would then mean we should have not done this step and Hence it correctly evaluates to $-V(n+1)$.

Case II $n = 0$ Base case done

Case III logically either to obtain maximum value n^{th} element will be considered or not. Whichever gives max value is to be chosen
And since by IH the algo is true for n-1. By above argument it is also true for n.

Hence Proved.

2. Clearly variables V, W do not have any impact of time and space complexity. Since both take O(1) space, time for evaluation and are independent of, cannot lead to termination.

Now talking of variable W, n

The termination of algorithm by W will strongly depend on w(n) if w(n) is sufficiently large it will lead to faster termination but cannot be quantified and will be diff. for different cases. so it is safer to assume $W \gg w(n)$ in worst case scenerio.

So only effective variable to calculate upper bound on time and space complexity is n.

so let time complexity be T(n).

$T(0) = 2$ (2 comparisions)

$T(n) = 2T(n-1) + 7$ (3 comp, 4 +/-) (Note any constant will do)

Hypothesis: $T(n) < 16 \times 2^n - 10$ (Proof by Induction)

Base case: $n = 0$ $\quad T(0) = 2 < 6$

Induction Hypo. = let it be true for some n

$T(n) < 16 \times 2^n - 10$

Proof: $T(n+1) = 2T(n-1) + 7$

$< 16 \times 2^{n+1} - 13 < 16 \times 2^{n+1} - 10$

Hence clearly time complexity is $O(2^n)$

let space complexity be $s(n)$

$$s(n) = 1 + s(n-1)$$

$$s(0) = 1$$

(clearly since one expression is evaluated at one time as in Fibonacci.)

$$\sum_{n=0}^{n} s(n) = n \cdot + \sum_{n=0}^{n} s(n-1)$$

$$s(n) = n + s(0)$$

$$= n + 1.$$

Hence space complexity is $O(n)$.

3. Done in SML file.

Problem 3

- f: int → string (used in toString function)

$f(0) = "0"$ $\quad f(5) = "5"$

$f(1) = "1"$ $\quad f(6) = "6"$

$f(2) = "2"$ $\quad f(7) = "7"$

$f(3) = "3"$ $\quad f(8) = "8"$

$f(4) = "4"$ $\quad f(9) = "9"$

toString : int → string

uses internally defined function s_iter : int * string → string

$$toString(n) = \begin{cases} s\_iter(n, "") & n \neq 0 \\ "0" & n = 0 \end{cases}$$

$$s\_iter(n, y) = \begin{cases} y & n = 0 \\ s\_iter(n \text{ div } 10, f(n \bmod 10) \hat{} y) & \text{otherwise} \end{cases}$$

Note: "0" for $n = 0$ in toString(n) had to be defined since in s_iter if $n = 0$ i.e. $n = 0$ it returns "" (empty string) which, then would have been false.

Note: toString(n) is O(1) since s_iter(n, "") is O(1) in space complexity since only 1 scope is used in all computations involved. This is used in part 2 of this question.

1.

convertUnitsRec (n, name, factor) = c_recc (n, 0, 0)

(In SML program, c_recc is internally defined)

c_recc (n, y, z) = if n = 0 then "" else c_recc (n div factor(y), y+1, 3H)^ " " ^ tostring (n mod factor(y)) ^ " " ^ name (z)

*n is input in seconds, y is argument for 'factor', z is for 'name'

Note: This though seems to be similar to a iterative algorithm, This is NOT. It is recursive since every new recursive call that is made is required to return its value to the previous one. (In other words the previous steps have to be stored in memory) (i.e. tail recursion does not occur).

Note: n > 0 is assumed in this and next part.

Note: If suppose we want to calculate till name (n) then factor (n') must be sufficiently large. Since stopping of code depends on n div factor (n') = 0 By sufficiently large I mean factor (n) = 1 or 0 should not be used to stop the code. Otherwise error may pop up.

2.

convertUnitsIter(n, name, factor) = c_iter("", n, 0, 0)

(c_iter defined internally in SML code)

c_iter(a, b, c, n) = | a b = 0

| c_iter(tostring(b mod factor(c)) ^ " " ^ name(n) ^ " " ^ a, b div factor(c), c+1, n+1) otherwise

Here c, n represent inputs of factor, name respectively and intuitively they are increased by one in each step.

Invariant: 1. b represents the input in name(n) units which is left to convert to human readable form.

2. a represents the part of input which has been converted to human readable form in units smaller than name(n).

Space complexity analysis.

Clearly all of the computation is happening in 1 scope and maximum scopes being active = 3 (for calling of factor, name, tostring functions one after the other & not necessarily in given order.)

More formal proof on next page.

Let $S(c)$ be the space complexity function.

$S(n)$ could also be chosen but would mean the same thing as c increases with n.

Now clearly, $S(c) = S(c+1)$ (c means c from (a, b, c, n))

and $S(n) = 1$ (in other words complexity is calculated variable c)

where n is the value of c when the algorithm terminates.

Now by Induction

Base case: $c = n$ $S(n) = 1$

Hypo: Let $S(c+1) = 1$

Proof: $S(c) = S(c+1) = 1$

Hence $S(c) = 1$ for all c

Hence Space complexity is $O(1)$.

3. Let in the name function string with largest length have N characters and largest factor in factor function have M digits.

Since both are given to be $O(1)$.

— Time complexity of s-iter(a,y) function.

Let a number a with n digits be passed into it then variable for time comp is considered n

$T(n) = T(n-1) + (1 + n - x^*)$ → (due concatenation ⊛ and size(y) = $n - x^*$ since ⊛ digits in input are n)

$T(0) = 1$

$T(a)$ is time complexity when $a = n$ x^* is digits in x

Now

$$\sum_{n=1}^{a} T(n) = \sum_{n=1}^{a} T(n-1) + a(n+1) - \sum_{n=1}^{a} n^*$$

$$T(a) \leq 1 + a(n+1)$$

But since factor(1) is O(1) Hence there is a constant limit to number input.

Hence time complexity of tostring can be considered O(1) for our purpose

Time complexity of the recursive algorithm calculated wrt variable x as occuring in (x, y, z)

so T(n) at n=n is required.

$$T(0) = 1$$

(considering concatenation is O(n+m))

$$T(n) = T\left(\left\lfloor \frac{x}{M} \right\rfloor\right) + A + M + N$$

Here M is max digits in factor function

A is max constant time from tostring

N is max(seize(name(n)))

(factor, size(name) is O(1) given)

$$T(n) = T\left(\frac{n}{M}\right) + c$$

Now let $M^a \geq n$ and $n > m^{a-1}$ then

$$a \geq \log_M n > a-1 \quad \text{then}$$

$$\sum_{(n=M^i)i=1}^{a} T(n) = \sum_{(n=M^i)i=1}^{a} T\left(\frac{n}{M}\right) + ac$$

$$T(M^a) = T(0) + ac = 1 + c \log_M M^a$$

clearly it can be seen for n also such expression will be formed Hence,

$T(n)$ is $O(\log_M n)$

Time complexity for the iterative algorithm

By invariant the maximum length(a) $= c(M+N+2)+1$

max size of string from to string $= cY + 1$ ← const

max(size(name(n)))

Here since c is involved (from a, b, c, n) we will first find complexity in variable c and then since

$\log_M n \cong \max(c) = a'$ (similar to previous proof const. neglected since order required)

we will use this relation at last.

We require $T(c)$ at $c=0$

$T(a') = 1$

$T(c) = T(c+1) + cY + 1$

$$\sum_{c=0}^{a'-1} T(c) = \sum_{c=0}^{a'-1} T(c+1) + \frac{a'(a'-1)}{2} Y + a'$$

$T(0) = \frac{a'(a'-1)}{2} Y + a(+1)$

Hence wrt c it is $O(a'^2)$

Hence by relation stated at starting wrt input number n it is $O((\log_M n)^2)$

4. Done in SML code file

Problem 4

1. $g(n) = g\text{-iter}(n, 1)$

(g-iter is defined inside g in sml code)

$$g\text{-iter}(n, x) = \begin{cases} x & 4 > n \text{ div } x \geq 1 \\ g\text{-iter}(n, 4 \times x) & \text{otherwise} \end{cases}$$

This function is designed to find out a number 4^a such that $4 > \frac{n}{4^a} \geq 1$. $\left(2^2 > \left[\frac{n}{4^a}\right] \geq 1\right)$

$intsqr(n) = int\_iter(1, n, g(n))$

$$int\_iter(x, y, z) = \begin{cases} x & z = 1 \\ int\_iter(2x+1, y, z \text{ div } 4) & (2x+1)^2 \leq y \text{ div } (\frac{z}{4}) \\ int\_iter(2x, y, z \text{ div } 4) & \text{otherwise.} \end{cases}$$

Note: z is g(n) i.e. 4^n hence z div 4, z/4 are same.

NOTE: $n \geq 1$ is assumed.

Proof of correctness.

1. If g-iter(n, 1) is true then g(n) will be true.
2. To prove correctness for g-iter(n, x)

First it is shown that the algorithm stops

Now for any n there exist p such that ($p \in N$,)

$$4^{p+1} > n \geq 4^p$$

$$4 > \frac{n}{4^p} \geq 1$$

$$4 > q + \frac{\epsilon}{4^p} \geq 1$$ (where q is the quotient and ϵ is the remainder $\epsilon \in [0, 4^p)$)

so q can take values $1, 2, 3$.

Now since n is a multiple of 4, it will eventually be equal to 4^p (Proof by contradiction) and Hence

n div n = n div 4^p = q where $4 > q \geq 1$ and Hence algorithm is bound to terminate.

Now Proof that when it terminates it gives the desired output.

Suppose algo terminates at $n = 4^q$

then $4 > n$ div $4^q \geq 1$

equivalent to $4 > \lfloor \frac{n}{4^q} \rfloor \geq 1$ (directly implies from definition of quotient and remainder)

equivalent to $2^2 > \lfloor \frac{n}{4^q} \rfloor \geq 1^2$

NOTE: Intsqrt($\lfloor \frac{n}{4^q} \rfloor$) is 1.

NOTE: Base case $n = 1$ can be verified.

- Correctness of intsqr(n) is established if int-iter(1, n, g(n)) is correct.

- Correctness of int-iter(x, y, z).

Invariant $(x+1)^2 > \lfloor \frac{y}{z} \rfloor \geq x^2$

equivalent to $(x+1)^2 > y \text{ div } z \geq x^2$

NOTE: z is passed in as g(n) which is a multiple of 4. Hence z div 4, $\frac{z}{4}$ is considered one and the same thing. (In case z=1 algo stops)

Proof of invariant by PMI on z

Here a altered version is used in which it will be assumed true for z and proved for $\frac{z}{4}$

(This is a direct consequence of version 3 of PMI where induction variable would be $\frac{g(n)}{z}$ which would increase. But for simplicity this version is assumed true).

Base case: $z \geq g(n)$ then by definition of g(n), invar. is true.

Induction Hypothesis: Let invariant be true for some (x, y, z). It is to shown to be true for $\frac{z}{4}$.

Induction Proof:

Let $\frac{n}{2^z} = i + f$ (i is the integer part, f is frac. part).

Now due to IH

$(n+1)^2 > i \geq n^2$, $\longrightarrow$ $i \geq n^2$ and also $(n+1)^2 \geq i+1$ (1)

$\frac{4n}{2^z} = 4i + 4f$

$4i+4 > \frac{4n}{2^z} \geq 4i$

by (1)

$(2x+2)^2 > \left\lfloor \frac{4n}{2^z} \right\rfloor \geq (2x)^2$

now either

$(2n+2)^2 > \left\lfloor \frac{4n}{2^z} \right\rfloor \geq (2n+1)^2$

or

$(2n+1)^2 > \left\lfloor \frac{4n}{2^z} \right\rfloor \geq (2n)^2$.

Hence the correctness of invariant for $\frac{z}{4}$

i.e. $(2n+1, y, z \text{ div } 4)$ if $(2n+1)^2 \leq y \text{ div } \left(\frac{z}{4}\right)$

or $(2n, y, z \text{ div } 4)$ otherwise

is established.

also clearly when $z=1$ by invariant, the integer square root will be correctly obtained.

i.e. $(n+1)^2 > \left\lfloor \frac{n}{1} \right\rfloor \geq n^2$

2. Time complexity of g-iter(n, x)

let it be $T(n)$ (It is being calculated wrt to variable n)

then $T(n) = T(4n) + 3$ (3 comp.)

let $4^{p+1} > n \geq 4^p$ then $p+1 > \log_4 n \geq p$

so $T(4^p) = 2$

since $T(n) = T(4n) + 3$

$$\sum_{(n=4^i)\, i=0}^{p-1} T(n) = \sum_{(n=4^i)\, i=0}^{p-1} T(4n) + 3(p-1)$$

$$T(1) = T(4^p) + 3(p-1)$$

$$= 3p - 1$$

clearly $T(1)$ is time complexity of g-iter(n, 1) because $n = 1$.

$$T(1) = 3p - 1 \leq 3\log_4 n - 1 = \underline{O(\log_4 n)}$$

- Space complexity of g-iter(n, 1) is clearly $O(1)$ since everything is being calculated in one scope only i.e. $S(n) = S(4n)$

$S(4^p) = 1$ (Here any other const. can also come if space to store n, names, etc. is cons.)

implies $S(1) = 1$

$= \underline{O(1)}$

Time complexity of int_iter(n, y, z)

Let it be T(z) (calculated wrt variable z)

$T(z) = T(z/4) + 5$ (addition, div are considered as O(1). Here 4 can also come but max complexity considered)

$T(1) = 1$

T(g(n)) is the complexity of required function. (any other constant will also do)

NOTE: complexity of g(n) will be added afterwards since it is calculated only once.

$$\sum_{(z=4^i)\, i=0}^{\log_4 g(n)} T(z) = \sum_{(z=4^i)\, i=1}^{\log_4 g(n)} T\left(\frac{z}{4}\right) + 5\log_4 g(n)$$

$T(g(n)) = 1 + 5\log_4(g(n)) = 1 + 5p$

$(4^p = g(n))$

Total time complexity =

$(4^{p+1} > n \geq 4^p)$

$= 1 + 5p +$ Time complexity of g(n)

$= 1 + 5p + 3p - 1$

$= 8p$

$= 8\log_4 n = O(\log_4 n)$

- Space complexity of int_iter(n, y, z) is one since everything is done in one scope itself.

i.e. $S(z) = S(z/4)$, $S(1) = 1$

Hence $S(g(n)) = 1 = O(1)$