331 midterm sheet

1.Integer-valued-function--> decrease in value by at least one every time --> halts without calling ieself Since n is an integer input, this is certainly an *integer-valued*function.

The slytherin algorithm only calls itself recursively (twice) during the execution of thestepatline5. Onec anseeatthe input nis replaced by either **n−1 or n−2**when the algorithm calls itself. Thus (since f is the identity function) the value of this function is *decreased in value by at least one*every times the algorithm calls itself recursively.

Suppose that the precondition for the “Slytherin Number Computation” problem is satisfied, and that the value of this function is less than or equal to zero when the slytherin algorithm is called. Then n ≥ 0, since the precondition for the problem is satisfied, but n ≤ 0 since f(n) = n. Thus n = 0.  In this case the execution of the algorithm halts after the test at line 1, which is passed, and the execution of the return statement at line 2. Thus the algorithm halts without calling itself again recursively in this case, as required.

2.basis: case (描述点：input/test line pass/return) --> indutive hypothesis ( new 0 <= l <= k, return kth number Sk) -->inductive claim(input k +1 input/return k+1st Sk+1)--> k+1 since k >=1 so k+1 >=2 test line pass -->  n-1 = k input return Sk -- > n-2 = k -1 input return Sk-1 -- >

The claim will be proved by induction on n, using the strong form of mathematical induction. The cases n = 0 and n = 1 will be considered in the basis.

*Basis:*

*Case:*n = 0*.*Suppose first that the algorithm is executed with the integer n = 0 as input. In this case, the test at line 1 is checked and passed, and the value 0 is returned after the execution of the return statement at line 2. Since Sn = S0 = 0, this establishes the claim in this case.

*Case:*n = 1*.*Suppose, next, that the algorithm is executed with the integer n = 1 as input. In this case, the test at line 1 is checked and fails, the test at line 3 is checked and passes, and the value 1 is returned as output after the execution of the return statement at line 4. Since Sn = S1 = 1, this establishes the claim in this case as well.

*Inductive Step:*Let k be an integer such that k ≥ 1. It is necessary and sufficient to use the following

Inductive Hypothesis: For every integer l such that 0 ≤ l ≤ k, if the slithering algorithm is executed with the integer l as input, then this execution of the algo- rithm eventually halts, returning the lth Slytherin number, Sl, as output.

to prove the following

Inductive Claim: If the slytherin algorithm is executed with the integer k+1 as input, then this execution of the algorithm eventually halts, returning the k + 1st Slytherin number, Sk+1, as output.

With that noted, suppose that the slytherin algorithm is executed on the input k + 1. Since k ≥ 1, k+1 ≥ 2, so that the tests at lines 1 and 3 are both checked and fail, and the execution of the algorithm continues with the execution of the statement at line 5.

• The first of the recursive applications of the algorithm at line 5 has input n − 1 = k. Since 0 ≤ k ≤ k it follows by the inductive hypothesis that this execution of the algorithm eventually halts, with the kth Slytherin number, Sk, returned as output.

• The second of the recursive applications of the algorithm at line 5 has input n − 2 = k−1. Sincek≥1,0≤k−1≤k,anditfollowsbytheinductivehypothesisthatthis execution of the algorithm eventually halts, with the k − 1st Slytherin number, Sk−1, returned as output.

• It follows that this execution of the algorithm ends with the value 2×Sk −Sk−1

returned as output. Now, since k ≥ 1, k + 1 ≥ 2, and it follows by the recursive definition of the Slytherin numbers that

2×Sk −Sk−1 =Sk+1.   
Thus Sk+1 is returned as output, as needed to establish the inductive claim.

The claim now follows by induction on n.

3.This question also concerned the “Slytherin Number Computation” problem and the recursive slytherin algorithm that were shown on the first page of the supplement for this test.

You were asked to write a ***recurrence***for the number Tslytherin(n) executed by the the algo- rithm slytherin when this executed with a nonnegative integer n as input.

**Recurrence:**If n = 0 then two steps — at lines 1 and 2 — are executed. Thus Tslytherin(0) = 2.

If n = 1 then three steps — at lines 1, 3 and 4 — are executed. Thus Tslytherin(1) = 3.

Finally, if n ≥ 2 then the execution of the algorithm includes three steps — at lines 1, 3, and 5 — along with recursive applications of the algorithm with inputs n − 1 and n − 2. The numbers of steps used by these recursive applications are (by definition) Tslytherin (n − 1) and Tslytherin(n−2), respectively.

It follows that the desired recurrence is as follows: For every integer n such that n ≥ 0,

Tslytherin(n) = 2

Tslytherin(n) = 3

Tslytherin(n − 1) + Tslytherin(n − 2) + 3   
***How Large is This Function?***It is certainly *not*obvious, but the above recurrence can be

used to prove that

Tslytherin (n) = Fn + 5Fn+1 − 3   
for every integer n ≥ 0. As material from previous lecture supplements and tutorial exercises

can be used to show, it follows from this that

i f n = 0 , if n = 1, if n ≥ 2.

so that Tslytherin(n) is exponential in n.

(a)  You were first asked to describe how the values of hocus and pocus are related to the value of i at the *beginning*and the *end*of the jth execution of the body of the while loop, for every positive integer j such that the body of the while loop is executed at least j times.

**Relationship Between the Values of These Variables:**These variables satisfy the fol- lowing properties at the beginning and end of the jth execution of the body of the while loop:

• iisanintegersuchthat1≤i≤n. • hocus = Si−1.   
• pocus = Si.

(b)  You were next asked to give a ***loop invariant***for the while loop in this algorithm. **Loop Invariant:**

i. n is an integer input such that n ≥ 2.   
ii. i is an integer variable such that 1 ≤ i ≤ n.

iii. hocus = Si−1. iv. pocus = Si.

(c) You were asked to apply a theorem, stated in class, to show that your loop invariant is correct.

**Solution:**Loop Theorem #1 can be used to show this.

• Suppose that this program is executed when the precondition for the “Slytherin Num- ber Computation” problem is satisfied, so that a nonnegative integer n is given as input. The loop is only reached if the tests at lines 1 and 3 are checked and failed, so n ≥ 2, as needed to establish part (i) of the loop invariant.

Since i is an integer variable whose value is set to be 1 at line 7, 1 ≤ i ≤ 2 ≤ n when the loop is reached, establishing part (i) of the loop invariant.   
As noted in the answer to part (a), hocus = Si−1 and pocus = Si, establishing parts (iii) and (iv) of the loop invariant.

Thus the loop is invariant when the loop is reached (if it is ever reached at all).

Suppose that the loop is body is executed when the loop invariant is initially satisfied.

Part (i) was satisfied at the start and, since none of the statements in the loop change the value of n, part (i) is still satisfied at the end.   
Since part (ii) was satisfied, i is an integer variable such that 1 ≤ i ≤ n at the beginning of this execution of the loop body — but i < n as well, because the loop test at line 8 was checked and passed. Since i and n both have integer values, i≤n−1. Thestepsatlines10and11donotchangeeitheriorn,butthestepat line 12 increases the value of i by one, so that 2 ≤ i ≤ n when the end of the loop body is reached, establishing part (ii).

Since parts (iii) and (iv) of the loop invariant at the beginning of the execution of the loop body, hocus = Si−1 and pocus = Si. After the executions of the steps at lines 9 and 10, shazam = Si−1 and hocus = Si. Thus, after the execution of the stepatline11,pocus=2×Si−Si−1. However,sincei≥1,i+1≥2,andthis implies that pocus = Si+1.

Since the step at line line 12 increases the value of i by one, shazam = Si−2, hocus = Si−1 and pocus = Si when the end of the loop both is reached, as needed to establish parts (iii) and (iv). Thus the loop invariant is also satisfied at the end of this execution of the loop body.

Since the loop test at line 8 has no side-effects (that is, it does not change the values of any variables) it now follows by Loop Theorem #1 that the loop invariant, given above, is correct.

**Brief Proof of Partial Correctness:**Suppose that the cSlytherin algorithm is executed with a nonnegative integer n as input. One can see by inspection of the code that the algorithm halts, returning Sn as input, if n = 0 or n = 1, so that it is sufficient to consider the case that n ≥ 2.

In this case, if the execution of the algorithm halts then the beginning of the loop is reached (because n ≥ 2), and the execution of the loop eventually ends, since the execution of the algorithm does. At this point i ≤ n, since part (ii) of the loop invariant holds, but i is not less than n, because the loop test failed. Thus i = n and pocus = Si = Sn (by part (iv) of the loop invariant) at this point, so that Sn is returned as output, as required, after the execution of the return statement at line 13.

Thus Sn is returned as output if the execution of the algorithm ends. Since the execution of this algorithm has no undesired side-effects it follows that this algorithm is partially correct.

**Bound Function for Loop:** The function f (n, i) = n − i

is a bound function for the while loop in this algorithm.

Brief Proof That This is a Bound Function: Note first that n is an input for this algorithm and i is a variable that has been defined before the loop is reached. Thus the above function f is a (well-defined) function of the inputs and variables in this algorithm.

It is therefore sufficient to prove that this functions satisfies all three of the properties in- cluded in the definition of a “bound function for a while loop.”

Since n is an integer input, i is an integer input, and the difference between two integers is an integer, this is an integer-valued function.

If the body of this while loop is executed then the value of n is not changed, but the value of i is increased in value by one, when the step at line 12 is executed. Thus an execution of the body of the while loop decreases the value of this function by at least one.

Finally, suppose that the precondition for the “Slytherin Number Computation” prob- lem is satisfied when this algorithm is executed, and the value of this function is less than or equal to zero at some point when the loop’s test is checked. Then, since f (n − i) = n − i ≤ 0, n ≥ i, and the loop’s test (shown at step 8) fails, causing the execution of the loop to terminate.  It follows that the above function is a bound function for the while loop in this algorithm, as claimed.  Note: For full marks, a student’s proof should include at least a few references to the specific computational problem being solved and the while loop in this algorithm, so that the student is not just repeating the definition of a “bound function for a a while loop” and claiming that it is satisfied.

(b) You were asked to use this to give an upper bound

Upper Bound for Running Time of Loop: Loop Theorem #2 can be applied to conclude that the initial value of this function is an upper bound for the number of times that the body of the while loop is executed. Since i = 1 immediately before the execution of this loop begins, this upper bound is n − 1.

It follows that the loop’s test (at line 8) is executed at most one more times than this, that is, at most n times.

A formula included in the lecture on bounding the cost of while loops can now be applied:

For 1 ≤ j ≤ n let TTest(j) be an upper bound for the cost of the jth execution of the loop’s test. One can see by examination of the code that (since this is a simple comparison) TTest(j) = 1.

For 1 ≤ j ≤ n−1, let TBody(j) be an upper bound for the cost of the jth execution of the body of the loop. One can see by inspection of the code that the loop body consists of four (reasonably simple) assignment statements, so that TBody(j) = 4.

It now follows by an application of the formula given in the lecture notes that the number of steps included in an execution of this while loop is at most  n n−1 nn−1 􏰋 T Te s t ( j ) + 􏰋 T B o d y ( j ) = 􏰋 1 + 􏰋 4 = n + 4 ( n − 1 ) = 5 n − 4 .  j=1 j=1 j=1 j=1

(c) Finally, you were asked to used this to given an upper bound for the number of steps executed during an execution of this algorithm, using the uniform cost criterion to define this. This should also be a function of n.

Upper Bound for Running Time of Algorithm: Suppose that the precondition for the “Slytherin Number Computation” problem is satisfied, so that n is a nonnegative (input) integer. Let TcSlytherin(n) be the number of steps executed by the cSlytherin algorithm on such an input n.

If n = 0 then two steps — the steps at lines 1 and 2 — are executed when the algorithm is executed on input n. Thus TcSlytherin(0) = 2.

If n = 1 then three steps — at lines 1, 3 and 4 — are executed when the algorithm is executed on input n. Thus TcSlytherin(1) = 3.

Ifn≥2thensixsteps—thestepsatlines1,3,5–7and13—areexecuted,along with an execution of the while loop, during an execution of the algorithm on input n. It follows by the result of the previous question that TcSlytherin (n) ≤ 6+(5n−4) = 5n+2 in this case.

Thus if n is a nonnegative integer then

 2 i f n = 0 , TcSlytherin ≤ 3 if n = 1, 5n+2 ifn≥2.

Note: It follows by the remarks given at the end of the solution for Question #2 and the solu- tion for part (c), above, that the cSlytherin algorithm is much faster than the slytherin algorithm, when the input n is large.

(a)  Youwerefirstaskedtodescribehowanarray(orArrayList)Acanbeusedtoimplement a bounded stack. You were asked where the value at the bottom of the stack be should stored in the array and (if this stack has size n and the array has length m ≥ n), where the value at the top of the stack should be stored, if stack operations are to be implemented efficiently, and where the other values on the stack should be stored.  Location of Bottom of Stack: A[0] Location of Top of Stack: A[n − 1]  Location of Other Values of Stack: Positions from 1 to n − 2, in order from oldest to newest

(b)  YouwerenextaskedtogivepseudocodeoradescriptioninEnglishforamethodtoremove (or pop) a value x from a stack when this array-based implementation is being used — assuming that the size of the stack is also being stored.  Method To Pop a Value Off of a Stack — Brief Description in English: If the stack is already empty — so that its size is already 0 — throw an exception and do not change the array. Otherwise set x to be the value A[size − 1], decrement size and return x.  Method to Pop a Value Off of a Stack — Pseudocode  if (size == 0) { throw a NoSuchException

} else { x = A[size-1];

size = size − 1; return x; }