Computer Science Foundation Exam

August 12, 2016

Section I A

COMPUTER SCIENCE

NO books, notes, or calculators may be used,
and you must work entirely on your own.

SOLUTION

Question #	Max Pts	Category	Passing	Score
1	10	DSN	7	
2	10	ANL	7	
3	10	ALG	7	
4	10	ALG	7	
5	10	ALG	7	
TOTAL	50			

You must do all 5 problems in this section of the exam.

Problems will be graded based on the completeness of the solution steps and not graded based on the answer alone. Credit cannot be given unless all work is shown and is readable. Be complete, yet concise, and above all be neat.

1) (10 pts) DSN (Recursive Functions)

Write a recursive function that will return the binary equivalent of its input parameter, `decimalNo`. You may assume that `decimalNo` is in between 0 and 1023, inclusive, thus the converted binary value will fit into an integer variable. For example, toBinary(46) should return the integer 101110 and toBinary(512) should return 1000000000.

```
int toBinary(int decimalNo) {

    if (decimalNo < 2)
        return decimalNo;

    return 10*toBinary(decimalNo/2) + decimalNo%2;

}
```

Grading: 2 pts base case (1 pt for 0 case, 1 pt for 1 case)
8 pts rest - 1 pt return
2 pts mult 10
1 pt toBinary call
2 pt decimalNo/2 in recursive call
2 pts adding decimalNo%2

2) (10 pts) ANL (Summations and Algorithm Analysis)

Find the closed form solution in terms of n for the following summation. Be sure to show all your work.

$$\sum_{i=n}^{3n}\sum_{j=1}^{n-2} j$$

$$= \sum_{i=n}^{3n} \frac{(n-2)(n-1)}{2}$$

$$= \frac{(3n-n+1)(n-2)(n-1)}{2}$$

$$= \frac{(2n+1)(n-2)(n-1)}{2}$$

In the second step, we are summing a constant with respect to the summation index i, thus we can simply multiply the item being summed by the number of times it's summed.

Note, also accepted is the polynomial multiplied out:

$$= \frac{2n^3 - 5n^2 + n + 2}{2}$$

Grading: 5 pts for solving the inner summation, 5 pts for then solving the outer summation, grader decides partial credit within each part.

3) (10 pts) ALG (Stacks and Queues)

Consider the process of merging two queues, `q1` and `q2`, into one queue. One way to manage this process fairly is to take the first item in `q1`, then the first item from `q2`, and continue alternating from the two queues until one of the queues run out, followed by taking all of the items from the queue that has yet to run out in the original order. For example, if `q1` contains 3, 8, 2, 7 and 5, and `q2` contains 6, 11, 9, 1, 4 and 10, then merging the two queues would create a queue with the following items in this order: 3, 6, 8, 11, 2, 9, 7, 1, 5, 4, and 10. Assume that the following struct definitions and functions with the signatures shown below already exist.

```
typedef struct node {
    int data;
    struct node* next;
} node;

typedef struct queue {
    node* front;
    node* back;
} queue;


// Initializes the queue pointed to by myQ to be an empty queue.
void initialize(queue* myQ);

// Enqueues the node pointed to by item into the queue pointed
// to by myQ.
void enqueue(queue* myQ, node* item);

// Removes and returns the front node stored in the queue
// pointed to by myQ. Returns NULL if myQ is empty.
node* dequeue(queue* myQ);

// Returns the number of items in the queue pointed to by myQ.
int size(queue* myQ);
```

On the following page, write a function that takes in two queues, `q1` and `q2`, and merges these into a single queue, emptying out `q1` and `q2` in the process and returning a pointer to the resulting queue.

```
queue* merge(queue* q1, queue* q2) {

    queue* res = malloc(sizeof(queue));
    initialize(res);

    int list = 0;

    while (size(q1) > 0 || size(q2) > 0) {

        if (list == 0 && size(q1) > 0)
            enqueue(res, dequeue(q1));
        else if (list == 1 && size(q2) > 0)
            enqueue(res, dequeue(q2));

        list = (list+1)%2;
    }

    return res;
}
```

Grading: 2 pts for properly creating an empty list to return (1 pt malloc, 1 pt init)
2 pts for toggling mechanism between lists
2 pts for handling unequal list sizes somehow
1 pt dequeing
2 pts enqueing item into their created queue
1 pt return

4) (10 pts) ALG (Hash Tables)

Insert the following numbers (in the order that they are shown…..from left to right) into a hash table with an array of size 12, using the hash function, H(x) = x mod 12.

234, 344, 481, 567, 893, 178, 719, 686, 46, 84

Show the result of the insertions when hash collisions are resolved through

a) linear probing

b) quadratic probing

c) separate chaining

Index	**a** **Linear**	**b** **Quadratic**	**c** **Separate chaining**
0	46	84	84
1	481	481	481
2	686	686	
3	567	567	567
4	84		686
5	893	893	893
6	234	234	234
7		46	
8	344	344	344
9			
10	178	178	178->46
11	719	719	719

Grading: 3 pts total for Linear Probing, 3 pts for Separate Chaining Hashing, 4 pts for Quadratic Probing. Give full credit if all the answers are correct, 2/3 or 3/4 is most of the answers are correct, 1/3 or 1/4 if some answers but no more than half are correct, 0/3 or 0/4 if none of the answers in a column are correct.

5) (10 pts) ALG (Base Conversion)

(a) (5 pts) Convert $FAD8_{16}$ to octal.

Convert to binary
1111 1010 1101 1000

Realign bits (implied leading zeros for single 1 at beginning)

001 111 101 011 011 000

Translate to octal

$1\ 7\ 5\ 3\ 3\ 0_{8}$

Grading: 2 pts converting bits to binary, 2 pts realigning bits, 1 pt converting to octal

(b) (5 pts) Convert 2120_{10} to hexadecimal.

16 | 2120
16 | 132 R 8
16 | 8 R 4
16 | 0 R 8

$8\ 4\ 8_{16}$

Grading: 1 pt for each quotient and remainder (except 0), if solved differently, grade accordingly.

Computer Science Foundation Exam

August 12, 2016

Section I B

COMPUTER SCIENCE

**NO books, notes, or calculators may be used,
and you must work entirely on your own.**

SOLUTION

Question #	Max Pts	Category	Passing	Score
1	10	ALG	7	
2	10	ANL	7	
3	10	DSN	7	
4	10	DSN	7	
5	10	ALG	7	
TOTAL	50		35	

You must do all 5 problems in this section of the exam.

Problems will be graded based on the completeness of the solution steps and not graded based on the answer alone. Credit cannot be given unless all work is shown and is readable. Be complete, yet concise, and above all be neat.

1) (10 pts) ALG (Analysis and Critical Thinking: AVL Trees, Hash Tables, and Heaps)

a) (1 pt) Using big-oh notation, what is the **best-case** runtime for inserting an integer into an AVL tree that contains *n* integers?

O(log n) Grading: All or nothing.

b) (1 pt) Using big-oh notation, what is the **worst-case** runtime for inserting an integer into an AVL tree that contains *n* integers?

O(log n) Grading: All or nothing.

c) (2 pts) What is the worst-case runtime for insertion into a hash table with *n* elements, assuming we use quadratic probing to resolve collisions? (You may assume that our hash table satisfies all conditions necessary to ensure that quadratic probing won't get stuck in an infinite loop.)

O(n) Grading: All or nothing.

d) (2 pts) Given the following hash table, suppose we know that no strings have been deleted, but we don't know the order in which these three strings were inserted into the hash table. If we used linear probing to resolve collisions, what are all the possible hash values for the string "of" (assuming those hash values are modded by the table size)?

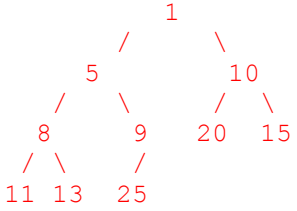
		pied	piper	of		Hamelin
0	1	2	3	4	5	6

Possible hash values: 2, 3, or 4
Grading: Give one point for each correct index given, and subtract half a point for each incorrect index given. Then take the floor.

e) (2 pts) Using big-oh notation, what is the **worst-case** runtime for deletion from a minheap that contains *n* elements?

O(log n) Grading: All or nothing.

f) (2 pts) Draw a minheap that contains 10 elements and which will incur the worst-case runtime if we call deleteMin() on it.



Note: Actual values may vary from answer to answer, but it's important that the percolate down operation takes the value to the left of the root, then left again, and then it can go left or right (doesn't matter).

Grading: 1 pt for a valid minheap, 1 pt for a minheap where the node in 25's position ends up where 11 or 13 are.

2) (10 pts) ANL (Summations and Algorithm Analysis)

a) (8 pts) Give a summation that represents the value returned by the following function, and then derive its closed form:

```
int something_to_ponder_over(unsigned int n)
{
   int i, retval = 0, pow = 1;
   for (i = 0; i < n; i++)
   {
      retval += pow;
      pow *= 14;
   }
   return retval;
}
```

The summation representing the return value is:

$$\sum_{i=0}^{n-1} 14^i$$

This is a geometric sum and can be solved as follows (or one might have the formula committed to memory, or get the formula from the formula sheet):

$$S = \sum_{i=0}^{n-1} 14^i = 14^0 + 14^1 + 14^2 + \cdots + 14^{n-1}$$

$$14S = 14^1 + 14^2 + 14^3 + \cdots + 14^n + 14^n$$

Now, subtracting S from 2S, most of the terms cancel out, leaving us with:

$$14S - S = 14^n - 14^0$$

Since 14S – S = 13S, we've solved for the summation:

$$S = (14^n - 1)/13$$

Grading: Award 5 points for the initial summation (2 pts for the correct bounds, 3 pts for the 14^i term). Award 3 points for deriving the closed form. Award partial credit as appropriate.

b) (2 pts) Using big-oh notation, what is the runtime of the function given in part (a)?

O(n) Grading: All or nothing

3) (10pts) DSN (Linked Lists)

Write a recursive function that takes the head of a linked list (possibly NULL) that contains positive integers only. The function must return -1 if the list contains any integer that is equal to the sum of all integers that come after it in the list. If not, the function can return whatever value you feel is appropriate other than -1. (Figuring out what to return is part of the fun for this problem.)

For example, the function should return -1 for the following linked list because 4 is the sum of all the nodes that follow it (1, 2, and 1):

```
20 -> 3 -> 1 -> 4 -> 1 -> 2 -> 1 -> NULL

^
head
```

The function signature and node struct are:

```
typedef struct node {
    int data;
    struct node *next;
} node;

int listylist(node *head) {

   int sum;

   if (head == NULL)
      return 0;

   sum = listylist(head->next);

   if (sum == -1 || head->data == sum)
      return -1;

   return head->data + sum;
}
```

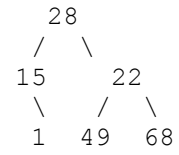
Grading:
2 pts for the base case (which should return 0 to work effectively)
2 pts for returning -1 if the recursive call itself returned -1
2 pts for returning -1 if head->data is equal to the sum from the recursive call
2 pts for returning a valid sum when not returning -1
2 pts for correct syntax and for avoiding segmentation faults

Note: There might be other solutions to this problem. Please award partial credit as necessary for alternate solutions.

4) (10 pts) DSN (Binary Trees)

Write a ***recursive*** function that takes the root of a binary tree (possibly NULL) and returns the sum of all the nodes that are left children in the tree. (See the example below, which returns 15 + 49 = 64, since the only nodes that are left children anywhere in the tree are 15 and 49.)

For this tree, the function should return 15 + 49 = 64:



The node struct and function signature are:

```
typedef struct node {
    int data;
    struct node *left;
    struct node *right;
} node;

int add_all_left_children(node *root) {

    if (root == NULL) return 0;

    int sum = 0;
    if (root->left != NULL)
        sum += root->left->data;

    return sum + add_all_left_children(root->left) +
                 add_all_left_children(root->right);

}
```

Grading: 2 pts base case, 1 pt NULL check left, 2 pts adding left node if it exists, 5 pts for the final return (1 pt for previous sum, 2 pts for each recursive call)

5) (10 pts) ALG (Sorting)

a) (3 pts) The following diagram shows an initial array, followed by what the array looks like after a single pass of some sorting algorithm. Indicate what sorting algorithm is being applied, and give that algorithm's worst-case runtime (using big-oh notation).

22	49	36	22	17	18	4

4	49	36	22	17	18	22

Sorting algorithm being applied: Selection Sort Grading: 2 pt, all or nothing

Worst-case runtime for algorithm: $O(n^2)$ Grading: 1 pt, all or nothing

b) (3 pts) For the following arrays, follow the same instructions from part (a):

84	19	23	66	91	44	42

19	23	66	84	44	42	91

Sorting algorithm being applied: Bubble Sort Grading: 2 pt, all or nothing

Worst-case runtime for algorithm: $O(n^2)$ Grading: 1 pt, all or nothing

c) (4 pts) Give a recurrence relation that represents the runtime for Merge Sort of n items. Let T(n) represent the runtime of Merge Sort of n items in setting up your recurrence relation.

$T(0) = T(1) = c_1$
$T(n) = 2T(n/2) + c_2*n + c_3$ (for $n > 1$)
Alternatively: $T(n) = 2T(n/2) + O(n)$ (for $n > 1$)

Grading: 2 pts for $2T(n/2)$
2 pts for $+ O(n)$ or similar

Computer Science Foundation Exam

August 12, 2016

Section II A

DISCRETE STRUCTURES

NO books, notes, or calculators may be used,
and you must work entirely on your own.

SOLUTION

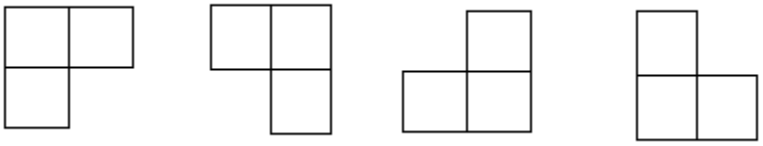
Question	Max Pts	Category	Passing	Score
1	15	PRF (Induction)	10	
2	15	PRF (Logic)	10	
3	10	PRF (Sets)	7	
4	10	NTH (Number Theory)	7	
ALL	50		34	

You must do all 4 problems in this section of the exam.

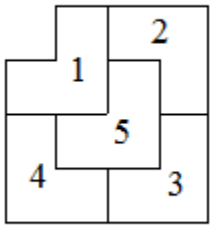
Problems will be graded based on the completeness of the solution steps and not graded based on the answer alone. Credit cannot be given unless all work is shown and is readable. Be complete, yet concise, and above all be neat.

1) (15 pts) PRF (Induction)

A tromino is a tile consisting of three unit squares in an L shape. The following are the four possible orientations a tromino can be placed:



Using induction on n, prove that for all non-negative integers, n, a 2^n x 2^n grid of unit squares with a single unit square removed can be tiled properly with a set of trominos. A proper tiling covers every unit square of the original object with a single unit square of a single tromino. For example, the following is a valid tiling of the 4 x 4 grid with the top left corner missing:



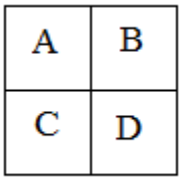
Solution

Base case: n=0, in this case we have a 2^0 x 2^0 square with one square missing, which means that what remains to be tiled is nothing, since we have one square missing from one square. Trivially, we can tile nothing with 0 L tiles.

Inductive hypothesis: For an arbitrary non-negative integer n = k, we can tile a 2^k x 2^k grid of unit squares with one square missing.

Inductive step: Prove for n = k+1 that we can tile a 2^{k+1} x 2^{k+1} grid of unit squares with one square missing.

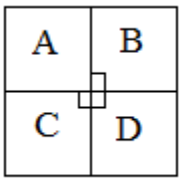
Given our square to tile, we partition it into 4 equal quadrants:



Note that since each quadrant is equal in size, each quadrant MUST BE a 2^k x 2^k square, since $2^k + 2^k = 2^{k+1}$. The missing unit square (that shouldn't be tiled) must be located in one of the four quadrants.

According to our inductive hypothesis, we can tile this quadrant (since it has a missing unit square).

Now, we have three quadrants left to tile. It must be the case that these three quadrants are "next to each other." For example, consider the case that the three quadrants are B, C and D. If this is the case, then we can place a single tromino at the center of the diagram above, with the tromino covering one unit square in B, one unit square in C and one unit square in D:



Now, we have tiled all of A with its square missing, as well as one unit square in quadrants B, C and D. Finally, we are left to tile quadrants B, C and D with ***exactly one unit square missing!!!*** We can perform this tiling based on the inductive hypothesis, completing the tiling of our original 2^{k+1} x 2^{k+1} design with a single unit square missing, completing the proof.

Grading: Base case - 2 pts, IH - 2 pts, IS - 2 pts, quadrant idea 2 pts, noting that one of these can naturally be tiled using the IH - 2 pts, placing a tile in the middle of the other three quadrants - 3 pts, use of IH to tile rest - 2 pts

2) (15 pts) PRF (Logic)

(a) (8 pts) Complete the truth table below.

p	q	r	$p \wedge q$	$\bar{p} \vee r$	$\overline{\bar{p} \vee r}$	$(p \wedge q) \vee (\overline{\bar{p} \vee r})$
F	F	F	**F**	**T**	**F**	**F**
F	F	T	**F**	**T**	**F**	**F**
F	T	F	**F**	**T**	**F**	**F**
F	T	T	**F**	**T**	**F**	**F**
T	F	F	**F**	**F**	**T**	**T**
T	F	T	**F**	**T**	**F**	**F**
T	T	F	**T**	**F**	**T**	**T**
T	F	T	**T**	**T**	**F**	**T**

Grading: 1 pt for each row, all or nothing. All 4 values on the row have to be completely correct to get credit for that row. Accept 0 = false, 1 = true as well.

(b) (7 pts) Create a logical expression using the variables p and q and only the logical operators (∧) and (¯) to create an expression which evaluates as described by the truth table below. (Note: There are many correct answers and each variable and operator may appear in the expression you create as many times as necessary.)

p	q	result
F	F	F
F	T	T
T	F	T
T	T	F

$$(\overline{\boldsymbol{p} \wedge \boldsymbol{q}}) \wedge (\overline{\bar{\boldsymbol{p}} \wedge \bar{\boldsymbol{q}}})$$

Note: There are many possible answers, please check each response by hand.

Grading: 3 pts for following the rules and only using and and not.
1 pt for each row that matches the given truth table for their expression

3) (10 pts) PRF (Sets)

Let A, B and C be finite sets such that $A \subseteq B$, $B \subseteq A \cup C$, and $C \subseteq B$. Prove or disprove the following assertion: $|A| = |B|$ or $|A| = |C|$ or $|B| = |C|$.

This claim is false. Here is a counter-example which satisfies the given requirements with three sets of different sizes:

A = {1}
B = {1, 2, 3}
C = {2, 3}

In this example, we see that both A and C are subsets of B and that B is a subset of $A \cup C$, satisfying the given requirements. Yet, the cardinality of all three sets is different.

Intuitively, we are simply showing that if the union of two sets (A and C) equals a third set (B), which is what the given information forces, then while B is at least as large as A and C, there are no definitive requirements on the sizes of A and C.

Grading: Any proof gets 1 pt maximum. 2 pts for stating the claim is false. 5 pts for a valid counter-example, 3 pts for explaining why the counter-example is valid. If the counter-example is not valid, you may still give some partial credit out of the 5 pts allotted for the counter-example.

4) (10 pts) NTH (Number Theory)

Find an integer, *n*, in between 0 and 231, inclusive, such that $105n \equiv 1 \pmod{232}$. (Note: To earn full credit you must use the Extended Euclidean Algorithm.)

Start with the Euclidean algorithm to find the gcd of 232 and 105:

232 = 2 x 105 + 22
105 = 4 x 22 + 17
22 = 1 x 17 + 5
17 = 3 x 5 + 2
5 = 2 x 2 + 1
2 = 2 x 1, thus the gcd is 1.

Now, we run the Extended Euclidean Algorithm:

5 - 2 x 2 = 1
5 - 2(17 - 3 x 5) = 1
5 - 2 x 17 + 6 x 5 = 1
7 x 5 - 2 x 17 = 1
7(22 - 17) - 2 x 17 = 1
7 x 22 - 7 x 17 - 2 x 17 = 1
7 x 22 - 9 x 17 = 1
7 x 22 - 9 x(105 - 4x22) = 1
7 x 22 - 9 x 105 + 36 x 22 = 1
43 x 22 - 9 x 105 = 1
43(232 - 2 x 105) - 9 x 105 = 1
43 x 232 - 86 x 105 - 9 x 105 = 1
43 x 232 - 95 x 105 = 1

Taking this equation mod 232 we see that $-95 \times 105 \equiv 1 \pmod{232}$. Thus one solution for n that satisfies the given equation is -95. But, this value isn't in the range given. Any value equivalent to -95 mod 232 satisfies the equation, thus the correct answer given the restriction on n is -95 + 232 = 137.

Grading: 3 pts for regular Euclidean Algorithm, 6 pts for extended and getting -95, 1 pt for noting that 137 is equivalent to -95 and is the correct answer for the range given.

Computer Science Foundation Exam

August 12, 2016

Section II B

DISCRETE STRUCTURES

NO books, notes, or calculators may be used,
and you must work entirely on your own.

SOLUTION

Question	Max Pts	Category	Passing	Score
1	10	CTG (Counting)	7	
2	10	PRB (Probability)	7	
3	15	PRF (Functions)	10	
4	15	PRF (Relations)	10	
ALL	50		34	

You must do all 4 problems in this section of the exam.

Problems will be graded based on the completeness of the solution steps and not graded based on the answer alone. Credit cannot be given unless all work is shown and is readable. Be complete, yet concise, and above all be neat.

1) (10 pts) CTG (Counting)

a) (6 pts) In an election with 142,070,000 eligible voters and only three candidates to choose from for some particular office, how many different distributions are possible for the number of votes each candidate could receive, provided that every eligible voter is forced to vote, and they must vote for one of the three candidates (so, the voters can't abstain from voting or choose some write-in candidate)?

For example, if candidate A receives 100,000,000 votes, candidate B receives 32,070,000 votes, and candidate C receives 10,000,000 votes, that is different from A receiving 32,070,000 votes, B receiving 100,000,000 votes, and C receiving 10,000,000 votes.

Note that votes are cast anonymously, so all that matters is the number of votes each candidate receives, with no consideration for which voters those votes came from.

This is a classical stars and bars problem. We have 142,070,000 stars, and we want to throw down 2 bars to create 3 partitions. It's possible to have empty partitions, so from our 142,070,002 objects (stars + bars), we want to choose 2 to be bars (or, equivalently, 142,070,000 to be stars).

The answer is: C(142,070,002, 2)

Grading: +3 for recognizing this as combinations / stars and bars. +2 for identifying n = 142,070,002 and +1 for k = 2. (Award partial credit for n and/or k if the values are very close.)

b) (4 pts) What would be the answer to (a) if instead of voters being forced to vote, they were allowed to sit at home and not vote for any of the candidates? (But still, no write-in candidates are allowed on the ballot. Those who vote are constrained to the three candidates on the ballot.)

In this case, we need to create a fourth partition for all the discarded votes. So, we need three bars this time. The answer becomes: C(142,070,003, 3)

Grading: +2 for incrementing n by one, and +2 for incrementing k by one.

2) (10 pts) PRB (Probability)

Suppose six wizards are seated in a row along one side of a long, straight banquet table, in totally random order. Among those wizards are Lily Evans, James Potter, and Severus Snape.

Let P be the event that Lily Evans and James Potter end up sitting next to one another, and S the event that Severus Snape and Lily Evans end up sitting next to one another. Prove or disprove that P and S are independent events. (You may assume there are no magical shenanigans at play that would affect the probabilities of these events.)

First, to make this solution clear, let's number our seats like so:

$\underline{\quad}_{S_1}$ $\underline{\quad}_{S_2}$ $\underline{\quad}_{S_3}$ $\underline{\quad}_{S_4}$ $\underline{\quad}_{S_5}$ $\underline{\quad}_{S_6}$

Note that there are 6! ways to seat our six wizards: $|S| = 6!$.

For event P, we must choose a pair of seats for Lily and James to occupy. There are 5 ways to choose adjacent seats: S_1 and S_2, S_2 and S_3, and so on, up to S_5 and S_6. We can then seat Lily and James in those seats in two ways: either Lily is to the left of James, or James it to the left of Lily. We can seat the four remaining wizards in the four remaining seats in 4! Ways.

So, $|P| = 5 \cdot 2 \cdot 4!$, which means $p(P) = 5 \cdot 2 \cdot 4! \,/\, 6! = \frac{1}{3}$.

Similarly, $|S| = \frac{1}{3}$.

Now we must find $p(P \cap S)$. For both James and Severus to sit next to Lily, we must first choose three adjacent seats for them to occupy. There are 4 ways to do that. Then, we must seat Lily in the middle seat, and either James is to her left (and Severus to her right), or James is to her right (and Severus to her left). So, there are 2 ways to arrange the three wizards within their three adjacent seats. Finally, we can seat the three remaining wizards in 3! possible ways.

So, $|P \cap S| = 4 \cdot 2 \cdot 3!$, which means $p(P \cap S) = 4 \cdot 2 \cdot 3! \,/\, 6! = \frac{1}{15}$.

Since $p(P \cap S) = \frac{1}{15}$ and $p(P) \cdot p(S) = \frac{1}{3} \cdot \frac{1}{3} = \frac{1}{9} \neq \frac{1}{15}$, the events are ***not*** independent. (That is to say, they are dependent.)

Grading: +2 for the cardinality of the sample space, +3 for the probability of P (which is also the probability of S), +3 for the probability of $P \cap S$, and +2 for showing the events are not independent by establishing the relationship between $p(P \cap S)$ and $p(P) \cdot p(S)$.

Note: If they compared $p(P \cap S)$ and $p(P) \cdot p(S)$, but thought that the inequality meant the results were independent, only award +2 for that part of the grading.

Please award partial credit as appropriate for alternate (but equally correct) solutions.

3) (15 pts) PRF (Functions)

Let $f: \mathbb{Z} \to \mathbb{Z}$ and $g: \mathbb{Z} \to \mathbb{Z}$, where $f(x) = 5x + 10$ and $g(x) = 10x + 5$. Then:

(a) (3 pts) Give $f \circ g$.

$f \circ g = f\big(g(x)\big) = 5(10x + 5) + 10 = 50x + 25 + 10 = 50x + 35$

Grading: +3 for correct answer. +2 if they gave $g \circ f$ instead of $f \circ g$. +2 if they had a simple algebraic mistake.

(b) (4 pts) Prove or disprove that $f \circ g$ is surjective.

Disproof by counterexample: There is no $x \in \mathbb{Z}$ such that $(f \circ g)(x) = 1$. If there were, we would have $50x + 35 = 1$, which implies $x = -34/50$, which is not an integer.

Grading: +4 for a valid counterexample. +1 if they try to prove this is true and use good form to do so (even though the result is incorrect). Award partial credit in other cases you encounter as necessary.

(c) (4 pts) Prove or disprove that $f \circ g$ is injective.

Proof: Let $(f \circ g)(x_1) = (f \circ g)(x_2)$. Then we have:

$50x_1 + 35 = 50x_2 + 35$

$50x_1 = 50x_2$

$x_1 = x_2$

Thus, $f \circ g$ is injective.

Grading: +4 for showing that $(f \circ g)(x_1) = (f \circ g)(x_2) \Rightarrow x_1 = x_2$. Award partial credit if they are on the right track but have minor errors.

(d) (4 pts) For a function $h: \mathbb{Z} \to \mathbb{Z}$, use quantifiers to write a statement in symbolic logic that says h is a surjective function.

There are many valid solutions here. One of them is: $\forall y \in \mathbb{Z}\ (\exists x \in \mathbb{Z}\ (h(x) = y))$

Grading: +4 for a correct solution, +3 for something very close, +2 if they have the right idea for surjectivity but their use of quantifiers has errors.

4) (15 pts) PRF (Relations)

(a) (3 pts) What three properties must a relation satisfy in order to be an equivalence relation?

It must be reflexive, transitive, and symmetric.

Grading:
-1 pt for each incorrect property listed
-1 pt for each correct property missing from the list

(b) (6 pts) Is it possible to define an equivalence relation R on $A = \{1, 2, 3, 4, 5, 6, 7\}$ such that $|R|$ is even? If so, give one such equivalence relation. If not, *briefly* explain why not.

No, it's not possible for $|R|$ to be even if it is an equivalence relation on A. We know that R must contain the seven ordered pairs from the set $\{(x, x) \mid x \in A\}$, giving it odd cardinality initially. All other pairs that we could add to R are of the form (x, y), where $x \neq y$. If we add such a pair to R, then we must also add the pair (y, x) in order to keep R symmetric. Thus, we always add an even number of pairs to R, meaning that its cardinality always remains odd.

Grading: 2 pts for saying "no." 4 pts for a reasonable explanation. Award partial credit as necessary.

(c) (6 pts) Suppose we define a relation R by choosing 9 random ordered pairs (without replacement) from $A \times A$, where $A = \{1, 2, 3, 4, 5, 6, 7\}$. What is the probability that R will be an equivalence relation? (You must show your work.)

For R to be an equivalence relation, we must select for inclusion all the ordered pairs from the set $\{(x, x) \mid x \in A\}$. We then get to select two additional pairs, and they must be symmetric images of one another. (For example, if we select $(1, 2)$, we must also select $(2, 1)$ in order for R to remain symmetric, which would thereby ensure R remained an equivalence relation.)

There are $7 \cdot 7 = 49$ ordered pairs to choose from in $A \times A$. We note that order does not matter when choosing these pairs for our relation. So, there are C(49, 9) ways to select ordered pairs to be in our relation.

The number of ways to select 9 pairs while keeping R an equivalence relation is simply the number of ways to choose ordered pairs (x, y) and (y, x), where $x \in A$, $y \in A$, and $x \neq y$. There are 42 such (x, y) ordered pairs, meaning there are 21 such pairs of ordered pairs.

The probability is: $21/C(49, 9)$

Grading: +2 for mentioning the 49 possible pairs somewhere in their result. +2 for the choose function in the denominator. +2 for the numerator.