Q1)

a) Decode the binary numbers into MIPS instructions *with proper register names* ($s0, $t0, etc.).If there are any memory addresses, represent them in hex.

Address	32-bit Binary Instruction	Type (R, I, J)	MIPS Instruction w/args
0xAFFFFFF8	0000 0001 0000 1000 0100 0000 0010 0110	R	xor $t0, $t0, $t0
0xAFFFFFFC	0001 0100 0000 1000 1111 1111 1111 1110	I	bne $0, $t0, -2
0xB0000000	0000 1000 0000 0000 0000 0000 0000 0001	J	j 0xB0000004
0xB0000004	001101 00000 00010 0000011000011100	I	ori $v0, $0, 0x61C
0xB0000008	000000 11111 00000 00000 00000 001000	R	jr $ra

b)

lui $t0, 0xFC00 or srl $v0 $a0 26
and $v0 , $a0, $t0
jr $ra

SID: _____

Q2)

`mystery`, a mysterious MIPS function outlined below, is written without proper calling conventions. `mystery` calls a correctly written function, **`random`**, that takes an integer `i` as its only argument, and returns a random integer in the range `[0, i - 1]` inclusive.

```
 1   mystery:   addiu $sp $sp ________-16_________
 2              __________sw $s0 0($sp)__________
 3              __________sw $s1 4($sp)__________
 4              __________sw $s2 8($sp)__________
 5              __________sw $ra 12($sp)_________
 6              ____________________________________
 7              addu $s0 $0 $0
 8              move $s1 $a0
 9              move $s2 $a1
10   loop:      srl $t0 $s0 2
11              beq $t0 $s2 exit
12              subu $a0 $s2 $t0
13              jal random
14              sll $v0 $v0 2
15              addu $v0 $v0 $s0
16              addu $t0 $s1 $s0
17              addu $t1 $s1 $v0
18              lw $t2 0($t0)
19              lw $t3 0($t1)
20              sw $t2 0($t1)
21              sw $t3 0($t0)
22              addiu $s0 $s0 4
23              j loop
24   exit:      __________lw $s0 0($sp)__________
25              __________lw $s1 4($sp)__________
26              __________lw $s2 8($sp)__________
27              __________lw $ra 12($sp)_________
28              _________addiu $sp $sp 16________
29              _____________jr $ra______________
30              ____________________________________
```

1) Fill in the prologue and the epilogue of this MIPS function. Assume that **`random`** follows proper calling conventions, and that it may make its own function calls. You may not need all of the lines.

2) What operation does this function perform on an integer array? Assume that both the integer array and the length of the array are passed into the function.

The function shuffles the integer array in place.

3) Would this function work as expected if a string was passed into the function instead? Write down the line numbers of all lines of MIPS code that must be changed (if any at all), so that the function works correctly on strings. Do not write down any extraneous line numbers.

10, 14, 18, 19, 20, 21, 22

SID: _____

Q3)

The function **countChars(char *str, char *target)** returns the number of times characters in **target** appear in **str**. For example:

```
countChars("abc abc abc", "a") = 3
countChars("abc abc abc", "ab") = 6
countChars("abc abc abc", "abcd") = 9
```

The C code for countChars is given to you in the box on right. The helper function **isCharInStr(char *target, char c)** returns 1 if **c** is present in **target** and 0 if not.

```
int countChars(char *str, char *target) {
  int count = 0;
  while (*str) {
    count += isCharInStr(target, *str);
    str++;
  }
  return count;
}
```

Finish the implement of countChars in TAL MIPS below. You may not need every blank.

```
countChars:
      addiu $sp, $sp, ___-16____
      _sw $ra, 0($sp)___________   # Store onto the stack if needed
      _sw $s0, 4($sp)___________
      _sw $s1, 8($sp)___________
      _sw $s2, 12($sp)__________
      _____________________________
      addiu $s0, $zero, 0          # We'll store the count in $s0
      addiu $s1, $a0, 0
      addiu $s2, $a1, 0
loop:
      addiu $a0, $s2, 0
      _lb $a1, 0($s1)___________
      beq _$a1, $zero, done_____
      jal isCharInStr
      _addu $s0, $s0, $v0_______
      _addiu $s1, $s1, 1________
      _j loop___________________
done:
      _addiu $v0, $s0, 0________   # Load from the stack if needed
      _lw $ra, 0($sp)___________
      _lw $s0, 4($sp)___________
      _lw $s1, 8($sp)___________
      _lw $s2, 12($sp)__________
      _____________________________
      addiu $sp, $sp, ___16_____
      jr $ra
```

SID: _____

Q4: *beargit* redux (15 points)

From project 1, you may remember the function `is_commit_msg_ok()` that you needed to implement in C. Here is a simpler rendition where commit messages are deemed okay *if and only if* those null-terminated commit messages exactly match `go_bears`. Using the **fewest number of empty lines possible**, finish writing the code below. You are only allowed to use the registers already provided **and** registers **`$t0-3`**, and **`$s0-s2`** (but you will not need all of them). Assume these registers are initialized to 0 before the call to ISCOMMITOK.

```
const char* go_bears = "THIS IS BEAR TERRITORY!";

int is_commit_msg_ok(const char* msg, const char* go_bears) {
    for (int i = 0; msg[i] && go_bears[i]; i++) {
        if (go_bears[i] != msg[i]) return 0;
    }
    if (!msg[i] && !go_bears[i]) return 1;
    return 0;
}
```

```
ISCOMMITOK:     ____________________________
                ____________________________
                _lb_ $t0   __0_($a0)
                _lb_ $t1   __0_($a1)
      COND:     ____and $t2 $t0 $t1_____
                ____beq $t2 $0 EXIT_____
                ____bne $t0 $t1 FAILED__
                addiu $a0 $a0 1
                addiu $a1 $a1 1
                _____j ISCOMMITOK_______
                ____________________________
      EXIT:     _or_ $t2 $t0 $t1
                ____bne $t2 $0 FAILED__
                li $v0 1
                ____j END_______________
      FAILED:   li $v0 0
      END:      ____jr $ra______________
                ____________________________
```

Q5)

We wish to free a linked list of strings (example below) whose nodes are made up of this struct. Complete the code below; we have started you off with some filled in. You may use fewer lines, but do not add any.

```
// Assume compiler packs tightly
struct node {
        char *string;
        struct node *next;
};

void FreeLL(struct node *ptr) {
        if (ptr == NULL) return;
        else {
                FreeLL(ptr->next);
                free(ptr->string);
                free(ptr);
        }
}
```

```
FreeLL:    beq $a0, $0, NULL_CASE
           addiu $sp $sp -8
           sw $ra 4($sp)
           sw $a0 0($sp)
           lw $a0 4($a0)
           ------------------------------
           jal FreeLL
           lw $a0 0($sp)
           lw $a0 0($a0)
           jal free
           lw $a0 0($sp)
           ------------------------------
           jal free
           lw $ra 4($sp)
           addiu $sp $sp 8
           ------------------------------
NULL_CASE:      jr $ra
```

Q6)

Answer the questions below about the following MIPS function. Answer each part separately, assuming each time that mystery() has not been called yet.

```
      mystery:
1           andi  $a0, $a0, 3
2           ori   $t0, $0,  1
3           sll   $t0, $t0, 6
4     Lbl1: beq   $a0, $0,  Lbl2
5           sll   $t0, $t0, 5
6           addi  $a0, $a0, -1
7           j     Lbl1
8     Lbl2: la    $s0, Lbl3
8           lw    $s1, 0($s0)
9           add   $s1, $s1, $t0
10          sw    $s1, 0($s0)
11    Lbl3: add   $v0, $0,  $0
12          jr    $ra
```

A. Which instruction (number) gets modified in the above function?
   < line 11: add $v0, $0, $0 >

B. Write an equivalent arithmetic (not logical) C expression to instruction 1. a0 = ____
   <a0 % 4 >

C. Which instruction field gets modified when mystery is called with $a0 = 3?
   <Executing mystery with $a0 = 3 results in $t0 being shifted left by 21. The 1 bit in $t0 was aligned with the last bit of the rs field, so the addition incremented rs by 1, changing $0 to $at>

D. How many times can mystery(0) be called before the behavior of mystery() changes?
   <31 times because the $a0 field is written into the shamt field, which as 5 bits (can be incremented up to 2^5-1 = 31)

E. A program calls mystery with the following sequence of arguments: 0, 1, 2, 3, 4, 5. What MIPS instruction gets stored in memory?

   add $a0, $at, $at
   The first instruction takes the modulus of $a0 by 4, so it was equivalent to calling the function with arguments 0, 1, 2, 3, 0, 1. Thus, rs and rt incremented by 1 while rd and shamt are incremented by 2.