2015 Fall Computer Science I Section 2
Exam 2 Multiple Choice
VERSION A

The following code implements a stack using an array. Several lines of the implementation have been omitted. Questions 1 - 5 will be about these missing lines.

```
#define SIZE 10
#define EMPTY -1

struct stack {
    int items[SIZE];
    int top;
};

// Initializes an empty stack.
void initialize(struct stack* stackPtr) {
     stackPtr->top = 0;
}

// Attempts to push value onto the stack pointed to by stackPtr. Returns
// 1 if the push was successful, 0 otherwise.
int push(struct stack* stackPtr, int value) {

    if (full(stackPtr))
        return 0;

    stackPtr->items[/*** Q1 ***/] = value;
    /*** Q2 ***/
    return 1;
}

// Returns 1 iff the stack pointed to by stackPtr is full, 0 otherwise.
int full(struct stack* stackPtr) {
    return /*** Q3 ***/;
}

// If the stack is non-empty, the top of the stack is returned. Otherwise
// -1 (EMPTY) is returned.
int top(struct stack* stackPtr) {

    if (/*** Q4 ***/)
        return EMPTY;

    return /*** Q5 ***/;
}
```

1) What expression should replace /*** Q1 ***/?

a) value b) stackPtr->top-1 c) stackPtr->top

d) stackPtr->top+1 e) None of the Above

2) What line of code should replace /*** Q2 ***/?

a) stackPtr->top++; b) stackPtr->top--; c) top++;

d) top--; e) None of the Above

3) What expression should replace /*** Q3 ***/?

a) top == 0 b) stackPtr->top == 0 c) stackPtr->top == SIZE-1

d) stackPtr->top == SIZE e) None of the Above

4) What expression should replace /*** Q4 ***/?

a) top == -1 b) stackPtr->top == -1 c) stackPtr->top == 0

d) stackPtr->top == SIZE e) None of the Above

5) What expression should replace /*** Q5 ***/?

a) items[top-1] b) stackPtr->items[top-1] c) items[top]

d) stackPtr->items[stackPtr->top] e) None of the Above

The following code implements a queue using a linked list. Several lines of the implementation have been omitted. Questions 6 - 10 will be about these missing lines.

```
#define EMPTY -1

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

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

void init(struct queue* qPtr) {
    qPtr->front = NULL;
    qPtr->back = NULL;
}
```

```
// Pre-condition: qPtr points to a valid struct queue and val is the value to
//                enqueue into the queue pointed to by qPtr.
// Post-condition: If the operation is successful, 1 will be returned,
//      otherwise no change will be made to the queue and 0 will be returned.
int enqueue(struct queue* qPtr, int val) {

    struct node* temp = (struct node*)malloc(sizeof(struct node));


    if (temp != NULL) {
        temp->data = val;
        temp->next = NULL;

        if (qPtr->back != NULL)
            /*** Q6 ***/

        /*** Q7 ***/

        if (qPtr->front == NULL)
            qPtr->front = /*** Q8 ***/;

        /*** Q9 ***/
    }
    else
        /*** Q10 ***/
}
```

6) What line of code should replace /*** Q6 ***/?

a) qPtr = temp; b) qPtr->front = temp; c) qPtr->back = temp;
d) qPtr->front->back = temp; e) None of the Above

7) What line of code should replace /*** Q7 ***/?

a) qPtr = temp; b) qPtr->front = temp; c) qPtr->back = temp;
d) qPtr->front->back = temp; e) None of the Above

8) What expression should replace /*** Q8 ***/?

a) temp b) qPtr->front c) qPtr
d) qPtr->back->next e) None of the Above

9) What line of code should replace /*** Q9 ***/?

a) return 0; b) return 1; c) return qPtr->front->data;
d) return qPtr->back->data e) None of the Above

10) What line of code should replace /*** Q10 ***/?

a) return 0; b) return 1; c) return qPtr->front->data;
d) return qPtr->back->data e) None of the Above

11) What is the value of the following postfix expression?

5 8 * 9 - 3 - 2 2 + /

a) 4 b) 7 c) 21 d) 28 e) None of the Above

12) What is the correct conversion of the following infix expression into postfix?

(4 + 5) / (8 - (2 + 3 * (5 - 4)))

a) 4 5 + / 8 - 2 + 3 * 5 - 4
b) 4 5 + 8 2 3 5 4 - * + - /
c) 4 5 + 8 2 - 3 5 4 * + - /
d) 4 5 + 8 2 3 5 * - 4 + - /
e) None of the Above

13) The array below shows the state of an insertion sort right before the last iteration (to put the last element in its correct location). How many swaps will occur during this last iteration?

2	4	5	9	12	18	22	23	27	10

a) 3 b) 4 c) 5 d) 6 e) None of the Above

14) Show the contents of the array below after it has gone through one iteration of Bubble Sort.

3	12	6	5	9	2	8	13	1	7

a) 3 6 12 5 9 2 8 13 7 1
b) 3 6 5 2 9 8 12 13 1 7
c) 3 5 6 9 2 8 12 1 7 13
d) 3 6 9 2 5 8 1 7 12 13
e) None of the Above

15) How many times does the swap function get called on a Selection Sort of 10 elements? (Assume that every swap in the algorithm is performed by a swap function and that we count when an element swaps with itself.)

a) 0 b) 1 c) 9 d) 10 e) None of the Above

16) What is the worst case run-time of a Quick Sort of n elements?

a) O(n) b) O(nlgn) c) O($n\sqrt{n}$) d) O(n^2) e) None of the Above

17) Why does Quick Sort run faster than Merge Sort in practice, even though the theoretical analysis predicts otherwise?

a) Because its name is "quick".
b) Because the partition function doesn't require to copy values to a temporary array and back while the merge does.
c) Because Quick Sort always splits its input array into two equal sized halves.
d) Because Quick Sort uses pointers.
e) None of the Above

18) Which recurrence relation represents the run-time of Merge Sort best?

a) $T(n) = 2T(n-1) + 1$
b) $T(n) = \frac{T(n-1)}{2} + O(n)$
c) $T(n) = T\left(\frac{n}{2}\right) + O(n)$
d) $T(n) = 2T(\frac{n}{2}) + O(1)$
e) None of the Above

19) Consider running a Quick Sort on 7 elements. What is the fewest number of times that the partition function might be called?

a) 3 b) 4 c) 5 d) 6 e) None of the Above

20) What sound does a Crunch bar make when you bite into it?

a) Crunch! b) Swish! c) Oooooh!!! d) Ahhhh!!! e) None of the Above

2015 Fall Computer Science I Section 2 Exam 2 Free Response

Last Name: _____ , First Name: _____

1) (16 pts) Please determine a Big-Oh bound for the following recurrence using the iteration technique:

$$T(n) = 4T\left(\frac{n}{2}\right) + 1$$

Note: $\sum_{i=0}^{k-1} 4^i = \frac{4^k - 1}{3}$.

2) (10 pts) Write a ***recursive*** function, `equal`, that takes in pointers to two linked lists and returns 1 if the two lists are equal and 0 otherwise. For two lists to be equal, they have to have the same number of elements, and each corresponding element must be equal. (For example, the lists 3, 9, 5 and 3, 9, 5 are equal, but the lists 3, 4, 7 and 3, 7, 4 are not equal and the lists 2, 9, 1 and 2, 9, 1, 4 are not equal.)

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

int equal(node* listA, node* listB) {









}
```

3) (14 pts) Write a function, `completeBinTree`, that takes in a pointer to the root of a binary tree. If the tree is a complete binary tree with the bottom level filled (drawing on overhead), the function return the height of the tree. Otherwise, it should return -10;

```
#define INCOMPLETE -10
typedef struct treenode {
    int data;
    struct treenode* left;
    struct treenode* right;
} treenode;

int completeBinTree (treenode* root) {









}
```