BBM 201 – Data Structures - Fall 2019

1st Midterm

November 19, 2019

Name: _____

Student ID Number: _____ Section:_____

Problem	Points	Grade
1	15	
2	20	
3	15	
4	15	
5	20	
6	20	
Total	105	

INSTRUCTIONS

- Do not open this exam booklet until you are directed to do so. Read all the instructions first.
- When the exam begins, write your name on every page of this exam booklet.
- The exam contains six multi-part problems. You have **120 minutes** to earn 105 points (5pts bonus).
- The exam booklet contains **6 pages** including this one.
- This exam is an **open book and notes exam**. You are allowed to have two pieces of **bound** books or notebooks.
- Please write your answers in the space provided on the exam paper.
- Be neat.
- Good luck!

QUESTIONS

Question 1) Performance analysis (15 pts):

a) (3 pts) For a collection of algorithms that runs in O(1), O(n log n), O(n), $O(n^2)$, O(log n), O(n!), order the algorithms from fastest to slowest.

O(1), O(log n), O(n), O(n log n), $O(n^2)$, O(n!)

b) (12 pts) Find the big-O time complexity of each of the following code fragments.
Hint:

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

$$1 + 2 + 4 + \cdots + \frac{n}{4} + \frac{n}{2} + n = 2n - 1 = O(n)$$

```
int i = 1;
while (i <= n) {
      print("*");
      i = 2 * i;
}
```

Answer: O(log n)

```
int i = n;
while (i > 0) {
      for (int j = 0; j < n; j++)
            print ("*");
      i = i / 2;
}
```

Answer: O(n log n)

```
while (n > 0) {
      for (int j = 0; j < n; j++)
            print("*");
      n = n / 2;
}
```

Answer: O(n)

```
for (int i = 0; i < n; i++)
      for (int j = i+1; j > i; j--)
            for (int k = n; k > j; k--)
                  print("*");
```

Answer: O(n^2)

Question 2) Recursion (20 pts):

a) (12 pts) Write a recursive function (that takes a string and the length of the string as arguments) in C that returns 1 if the given string is palindrome, and 0 if it is not. A string is said to be a palindrome if the string read from left to right is equal to the string read from right to left. For example, "*kayak*", "*level*", "*radar*", "*repaper*", "*noon*" are all palindrome words in English, however "*paper*", "*book*", "*little*" are not. Only recursive solutions will be accepted.

```
int isPalindrome (char *str, int len){

    if (len == 0 || len == 1)
        return 1;
    if (str[0] == str[len-1])
        return isPalindrome (str+1, len-2);
    else return 0;
}
```

b) (8 pts) Please write the output of the following method if recursiveFun(6) is called.

```
void recursiveFun(int value)
{
      if(0 < value && value < 10)
      {
              recursiveFun(value - 2);
              recursiveFun(value + 1);
              printf(" %d", value);
      }
}
```

The code produces no output and ends with "segmentation fault".

Question 3) Multi-dimensional arrays (15 pts):

Consider the following multi-dimensional array:

int A[3][4][2] =

100 192

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23

Let p be defined as,

```
int (*p)[4][2] = A;
```

Fill in the values for the second column in the following table according to the expressions/code in the first column:

Expression/code	**Result**
`p`	100
`p + 2`	164
`*(p + 1)`	132
`**(p + 1)`	132
`***(p + 1)`	8
`*(*p + 2)`	116
`*(*(*p + 2) + 1)`	5
`*(*(*(p + 1) + 1) + 1)`	11

`int sum = 0;`
`for (int i = 0; i < 3; i++)`
`  sum += **(*(p + i) + 1);`
`printf("Sum : %d\n", sum);`
	Sum: 30

`int sum = 0;`
`for (int i = 0; i < 3; i++)`
`  sum += *(*(*(p + 2) + i) + 1);`
`printf("Sum : %d\n", sum);`
	Sum: 57

Question 4) Triangular Matrix (15 pts):

Let A[n][n] be a lower triangular matrix (see the example given below in A and C). The elements of this triangular matrix are stored in a one-dimensional array (U). We want to convert the given left-aligned lower triangular matrix into a right-aligned lower triangular matrix as given in B and D without changing the order of the items in the one-dimensional array U.

gettriangularmatrix(int i,int j,int n) method returns the item in the given index (A[i][j]).

Which lines in the method should we change to make it work for the right-aligned lower triangular matrix for the same one-dimensional array? Please write the updated lines on the side of each line that needs to be changed. No new lines will be added. Leave the space blank for the lines which do not require any changes.

A

a_{00}	0	0	0
a_{10}	a_{11}	0	0
a_{20}	a_{21}	a_{22}	0
a_{30}	a_{31}	a_{32}	a_{33}

B

0	0	0	a_{03}
0	0	a_{12}	a_{13}
0	a_{21}	a_{22}	a_{23}
a_{30}	a_{31}	a_{32}	a_{33}

C

1	0	0	0
2	3	0	0
4	5	6	0
7	8	9	10

D

0	0	0	1
0	0	2	3
0	4	5	6
7	8	9	10

U

a_{00}	a_{10}	a_{11}	a_{20}	a_{21}	a_{22}	a_{30}	a_{31}	a_{32}	a_{33}

U

a_{03}	a_{12}	a_{13}	a_{21}	a_{22}	a_{23}	a_{30}	a_{31}	a_{32}	a_{33}

```
int gettriangularmatrix(int i, int j, int n){

    if(i<0||i>=n||j<0||j>=n){  ..................................... 1
            printf("\n invalid index\n");........................... 2
            exit(-2);
     }
     else if(i>=j) //valid index      ........ else if (i+j>=n-1)..... 3
            return (i+1)*i/2+j  ........... (i*(i+1)/2)+j-(n-i-1)..... 4
     else return -1;          ..................................... 5
}
```

Question 5) Stacks (20 pts):

Answer the following questions by putting a check mark on the correct column. Unless otherwise stated, you have access to only the stack interface functions. You cannot directly access the underlying array structure.

Q	Question	True	False
1	To reverse the order of all the items within a stack, it is sufficient to use an additional stack (the items should end up in the original stack).		F
2	To process a list of tasks in the arriving order, using a single stack is sufficient.		F
3	Summing up all the items in a stack requires O(n) time.	T	
4	Finding the maximum item in a stack requires O(nlog n) time due to sorting.		F
5	Consider you have stacks S1 and S2 containing k1 and k2 integer values and both of them are sorted such that the item values increase towards the bottom of the stack. Let k be the sum of k1 and k2. Using only one more stack, S3, we want to merge both stacks, in same sorting order, in S1. We need at most 2*k **pop** operations to do this.	T	
6	To detect whether a word is a **palindrome** (which is same when read left-to-right and right-to-left) we can use a single stack. Some palindromes are racecar, radar, level, civic.	T	
7	When a stack of size n is full, we can transfer its contents to a larger stack in n pop and push operations.		F
8	For the 7^{th} question, if we had access to the underlying array structure, the stack capacity could be extended in constant O(1) time.		F
9	If we decide to replace a queue with stacks, we need two stacks to mimic the full queue functionality.	T	
10	Assume a **popBottom()** operation to pop the bottom item in a stack. If you have access to the underlying array structure, this operation can be implemented to require at most O(1) time.	T	

Question 6) Queues (20 pts):

a) (10 pts) A *priority queue* is a data structure that supports storing a set of values, each of which has an associated key. Each key-value pair is an entry in the priority queue. The basic operations on a priority queue are:

- *insert(k, v)* – insert value v with key k into the priority queue
- *removeMin()* – return and remove from the priority queue the entry with the smallest key

Other operations on the priority queue include *size()*, which returns the number of entries in the queue and *isEmpty()* which returns true if the queue is empty and false otherwise.

Two simple implementations of a priority queue are using an unsorted array, where new entries are added at the end of the array, and a sorted array, where entries in the array are sorted by their key values.

Fill in the following table to give the running times of the priority queue operations for these two implementations using *O()* notation. You should assume that the implementation is optimised.

Operation	Unsorted Array	Sorted Array
isEmpty()	O(1)	O(1)
insert(k, v)	O(1)	O(n)
removeMin()	O(n)	O(1)

b) (10 pts) For a given integer queue *Q*, fill in the blanks in the below pseudocode so that the function *findMaxOfQueue* will find the maximum element in *Q*. You may only use the operations: *enqueue(), dequeue(), size()*.

Usage:

dequeue() : Dequeues an item from Q

enqueue(X) : Enqueues X into Q

size() : Returns the size of Q

Queue must remain intact after finding the max. Each blank is either a complete single line or part of a single line.

```
findMaxOfQueue (Queue Q)
{
        max = ..... dequeue() ..... ;
        .....enqueue (max) ..... ;
        for ( int i=1 ; .....i<size()..... ; .....i++..... )
        {
                next = ..... dequeue().....;
                if (.....next>max.....) {
                        .....max=next..... ;
                }
                ..... enqueue(next)..... ;
        }
        return .....max..... ;
}
```