King Saud University	College of Computer and Information Sciences Department of Computer Science
Data Structures CSC 212	**Final Exam - Spring 2020**
Date: 02/05/2020	Duration: 3 hours

Do not answer in this sheet and do not submit it.

Question 1 .... 20 points

(a) Choose the correct frequency for every line as well as the total O of the following code:

```
1 sum = 1;
2 for (i = 1; i <= n; i++) {
3   sum+= i;
4   for (j = i; j >= 2; j--)
5     sum--;}
```

1. Line 1: [✓ (A) 1] (B) 2 (C) 0 (D) n (E) $2n$
2. Line 2: (A) n [✓ (B) $n+1$] (C) $n-1$ (D) $n+2$ (E) $n-2$
3. Line 3: [✓ (A) n] (B) $n+1$ (C) $n-1$ (D) $n+2$ (E) $n-2$
4. Line 4: (A) n^2 (B) $n(n-1)/2$ (C) $(2n+1)/2$ (D) $(2n-1)/2$ [✓ (E) $n(n+1)/2$]
5. Line 5: (A) n^2 [✓ (B) $n(n-1)/2$] (C) $(2n+1)/2$ (D) $(2n-1)/2$ (E) $n(n+1)/2$
6. Total O: (A) 1 (B) n [✓ (C) n^2] (D) $n\log(n)$ (E) n^3

(b) Choose the correct frequency for every line as well as the total O of the following code:

```
1 count = 0;
2 for (i = 1; i < n+1; i++)
3   count ++;
4 for (j = 0; j <= count; j++)
5   k = j+1;
```

1. Line 1: (A) 0 [✓ (B) 1] (C) 2 (D) n (E) n^2
2. Line 2: (A) n [✓ (B) $n+1$] (C) $n-1$ (D) $n+2$ (E) $n-2$
3. Line 3: [✓ (A) n] (B) $n+1$ (C) $n-1$ (D) $n+2$ (E) $n-2$
4. Line 4: (A) $count+1$ (B) $n+1$ (C) $n-1$ [✓ (D) $n+2$] (E) $n(n+1)/2$
5. Line 5: (A) $count+1$ [✓ (B) $n+1$] (C) $n-1$ (D) $n+2$ (E) $n(n-1)/2$
6. Total O: (A) 1 [✓ (B) n] (C) n^2 (D) $n\log(n)$ (E) n^3

(c) Choose the correct frequency for every line as well as the total O of the following code:

```
1 int i = 1;
2 while (i < n) {
3   i++;
4   if (i > 7) break;
5 }
```

1. Line 1: [✓ (A) 1] (B) 0 (C) i (D) n (E) $n+1$
2. Line 2: (A) 8 [✓ (B) 7] (C) n (D) $n-1$ (E) $n+1$
3. Lines 3: (A) n (B) $n-1$ (C) 6 [✓ (D) 7] (E) 8
4. Lines 4: (A) n (B) $n-1$ (C) 6 [✓ (D) 7] (E) 8
5. Tightest Total O: [✓ (A) 1] (B) n (C) $\log(n)$ (D) n^2 (E) 2^n

(d) Choose the correct answer:

1. $n^7 + n^4 + n^2 + \log n$ is : (A) $O(n^2)$ (B) $O(n^4)$ [✓ (C) $O(n^7)$] (D) $O(\log(n))$ (E) None
2. $2^n + n!$ is : (A) $O(n^2)$ (B) $O(2^n)$ [✓ (C) $O(n!)$] (D) $O(n^n)$ (E) None
3. $n + \log n^3 + 6$ is : [✓ (A) $O(n)$] (B) $O(\log n^3)$ (C) $O(n \log n)$ (D) $O(n^3)$ (E) None
4. The time complexity of inserting an element in a heap of n elements is:
   (A) $O(n^2)$ (B) $O(n)$ (C) $O(2^n)$ [✓ (D) $O(\log(n))$] (E) None
5. $n^2 + n \log n^4$ is : (A) $O(n)$ [✓ (B) $O(n^2)$] (C) $O(n \log(n))$ (D) $O(n^4)$ (E) None
6. $n^2 + 1000n$ is : (A) $O(n)$ [✓ (B) $O(n^2)$] (C) $O(n \log(n))$ (D) $O(nn^2)$ (E) None
7. $n^4 \log n + n!$ is : [✓ (A) $O(n!)$] (B) $O(n^4)$ (C) $O(n^5)$ (D) $O(\log(n))$ (E) None
8. Algorithm A is $O(n)$, and Algorithm B is $O(2n)$. Given the same input:
   (A) A always finishes before B. (B) B always finishes before A. (C) A and B finish at the same time. (D) B requires double the time taken by A. [✓ (E) None]

Question 2 . . . . 20 points

(a) Write the method `removeOddElems`, member of the class `LinkedList`, that removes all the elements having an odd position (the position of the first element is 0). The method signature is: `public void removeOddElems ()`.

Example 1. *If l : A → B → C → D → E, then* `l.oddElems()` *returns: A → C → E.*

1. Line 1:
   - (A) `while (head != null)`
   - (B) `if (head.next != null)`
   - [✓ (C) `if (head == null)`]
   - (D) `if (head != null)`
   - (E) None
2. Line 2:
   - (A) `return head;`
   - (B) `return false;`
   - (C) `return null;`
   - (D) `return current;`
   - [✓ (E) None]
3. Line 3:
   - [✓ (A) `Node<T> p = head;`]
   - (B) `Node<T> p = head.next;`
   - (C) `Node<T> p = current;`
   - (D) `Node<T> p = head.next.next;`
   - (E) None
4. Line 4:
   - (A) `while (p.next != null){`
   - (B) `while (p != null){`
   - (C) `while (p.next.next != null){`
   - (D) `while (current.next != null){`
   - [✓ (E) None]
5. Line 5:
   - (A) `p.next = p;`
   - (B) `p = p.next;`
   - [✓ (C) `p.next = p.next.next;`]
   - (D) `current.next = p;`
   - (E) None
6. Line 6:

[✓ (A) `p = p.next;}`]

(B) `p.next.next = current.next;}`

(C) `p = p.next.next;}`

(D) `p.next.next = p.next;}`

(E) None

(b) Consider the function `f` below, member of `LinkedList`:

```
public void f(int n) {
  Node<T> p = head; Node<T> q =  null;
  for (int i = 0; i < n; i++) {
    q = p;
    p = p.next;
  }
  if (p != null) {
    while (p.next != null)
      p = p.next;
    p.next = head;
    head = q.next;
    q.next = null;
  }
}
```

Choose the correct result in each of the following cases:

1. The list `l`: A, B, C, D, E, after calling `l.f(2)`, `l` becomes:

   (A) A, B (B) D, E, A, B, C [✓ (C) C, D, E, A, B] (D) A, D, E, B, C (E) None

2. The list `l`: A, B, C, D, E, after calling `l.f(0)`, `l` becomes:

   (A) *empty* (B) A, B, C, D, E (C) B, C, D, E, A (D) B, C, D, E [✓ (E) None]

3. The list `l`: A, B, C, D, E, after calling `l.f(5)`, `l` becomes:

   (A) *empty* [✓ (B) A, B, C, D, E] (C) E, D, C, B, A (D) A, D, E, B, C (E) None

4. The list `l`: A, B, C, D, E, after calling `l.f(1)`, `l` becomes:

   (A) A (B) E, A, B, C, D (C) C, D, E, A, B [✓ (D) B, C, D, E, A] (E) None

Question 3 . . . . 20 points

Write the method **public** `<T>` **boolean** `isReverse(Queue<T> q1, Queue<T> q2)`, user of the ADT `Queue`. The method checks if `q2` is the reverse of `q1`. The two queues must not change after the call.

1. Line 1:

   (A) `if (q1.length()== q2.length())`

   (B) `if ((q1.length()== 0)&& (q2.length()==0))return true;`

   [✓ (C) `if (q1.length()!= q2.length())return false;`]

   (D) `if ((q1.length()== 0 )|| (q2.length()== 0))return true;`

   (E) None

2. Line 2:

   (A) `Stack<T> x1 = new Stack<T>(), x2 = new Stack<T>();`

   (B) `Queue<T> x1 = new LinkedQueue<T>(), x2 = new LinkedQueue<T>();`

   [✓ (C) `Stack<T> x1 = new LinkedStack<T>(), x2 = new LinkedStack<T>();`]

   (D) `Stack<T> x = new LinkedStack<T>();`

   (E) None

3. Line 3:

   (A) `while (!q1.empty())`

   [✓ (B) `while (q1.length()!= 0)`]

   (C) `while (x1.length()!= 0)`

   (D) `while (!x.full ())`

   (E) None

4. Line 4:

   (A) `x.push(q1.retrieve());`

   (B) `x1.push(current.data);`

   [✓ (C) `x1.push(q1.serve());`]

   (D) `x1.enququе(q1.serve());`

   (E) None

5. Line 5:

   (A) `while (!x.empty()){`

   (B) `while (x1.size()>0){`

   (C) `boolean eq = false;`

   [✓ (D) `boolean eq = true;`]

   (E) None

6. Line 6:

   (A) `T e1 = x1.pop(), e2 = x2.pop();`

(B) `T e1 = x.pop(), e2 = q2.serve();`

[✓ (C)] `while (!x1.empty()){`

(D) `while (x1.size()>0){`

(E) None

7. Line 7:

(A) `if (!e1.equals(e2))break;`

(B) `T e1 = x1.pop(), e2 = q1.serve();`

(C) `if (!e1.equals(e2))`

[✓ (D)] `T e1 = x1.pop(), e2 = q2.serve();`

(E) None

8. Line 8:

(A) `if (!e1.equals(e2))continue;`

[✓ (B)] `if (!e1.equals(e2))eq = false;`

(C) `x2.push(e1); q2.enqueue(e1); }`

(D) `return false;`

(E) None

9. Line 9:

(A) `x.push(e1); q2.enqueue(e2); }`

[✓ (B)] `x2.push(e1); q2.enqueue(e2); }`

(C) `x1.push(e1); q2.enqueue(e1); }`

(D) `return false`

(E) None

10. Line 10:

(A) `while (!q2.empty())`

[✓ (B)] `while (!x2.empty())`

(C) `while (!q1.full())`

(D) `while (!x.last())`

(E) None

11. Line 11:

(A) `q1.enqueue(x1.pop());`

[✓ (B)] `q1.enqueue(x2.pop());`

(C) `q2.enqueue(x.pop());`

(D) `q2.enqueue(x2.pop());`

(E) None

12. Line 12:

Ⓐ **return false**;

Ⓑ **return true**;

Ⓒ **return** `e1.equals(e2);`

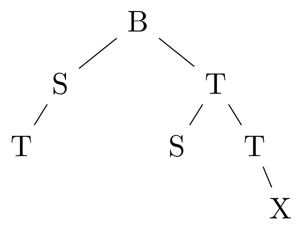
[✓ Ⓓ **return** `eq;`]

Ⓔ None

Question 4 . . . . 20 points

A player wants to escape a maze where at most two options are available at each step: go straight or turn. This maze can be represented by a binary tree where the data is a character that can take four values: 'B' for begin (only at the root), 'S' for go straight, 'T' for turn, or 'X' meaning this is an exit. Exits are located at leaf nodes only, but not all leaf nodes are exits, they could be dead ends.

Example 2. *In this maze, you see that you reach the exit by moving:* `T,T,X` *staring from the root.*



This is the class `MNode`:

```
class MNode {
  public char data;
  public MNode left, right;
  ...
}
```

(a) Write the method **private boolean** `follow(MNode t, String path)`, which tests if the path indicated by `path` and starting from `t` is valid. A path is valid if its directions are available (not necessarily leading to an exit).

Example 3. *In the maze shown above, the paths: "TT", "TTX", "ST" are valid, whereas the paths "SS" and "TST" are not valid.*

```
1  private boolean follow(MNode t, String path) {
2    MNode p = t;
3    int i = 0;
4    while (...) {
5      char c = path.charAt(i++);
6      if (...)
7        ...;
8      else if (...)
9        ...;
10     else
11       return ...;
12   }
13   return ...;
14 }
```

1. Line 4:

   (A) `while (p!=null && i<path.length()){`

   (B) `while (p==null){`

   (C) `while (i==path.length()){`

   [✓ (D) `while (i<path.length()){`]

   (E) None

2. Line 6:

(A) `if (p.left.data==c)`

[✓ (B) `if (p.left!=null && p.left.data==c)`]

(C) `if (p.data==c)`

(D) `if (p.left!=null)`

(E) None

3. Line 7:

   (A) `p.data=c;`

   [✓ (B) `p=p.left;`]

   (C) `p.left=p;`

   (D) `return false;`

   (E) None

4. Line 8:

   (A) `else if (p.data==c)`

   (B) `else if (p.right!=null)`

   [✓ (C) `else if (p.right!=null && p.right.data==c)`]

   (D) `else if (p.right.data==c)`

   (E) None

5. Line 9:

   [✓ (A) `p=p.right;`]

   (B) `p.right=p;`

   (C) `p.data=c;`

   (D) `return false;`

   (E) None

6. Line 11:

   (A) `return p.data!='X';`

   [✓ (B) `return false;`]

   (C) `return p.data=='X';`

   (D) `return t.data=='X';`

   (E) None

7. Line 13:

   (A) `return p==null;`

   [✓ (B) `return true;`]

   (C) `return p.data=='T'||p.data=='S';`

   (D) `return p.data!='X';`

   (E) None

(b) Write the method **private boolean** `escape(MNode t)`, which searches preorder for an exit starting at `t` and returns true if it finds one.

```
 1  private boolean escape(MNode t){
 2    LinkStack<MNode> s = new LinkStack<MNode>();
 3    MNode p = ...;
 4    while (...) {
 5      if (...)
 6        ...;
 7      if (...)
 8        ...;
 9      else if (...)
10        break;
11      else
12        ...;
13    }
14    return ...;
15  }
```

1. Line 3:
   - (A) `MNode p=root;`
   - [✓ (B) `MNode p=t;`]
   - (C) `MNode p=t.right;`
   - (D) `MNode p=t.left;`
   - (E) None
2. Line 4:
   - (A) **while** (p.data!='X'){
   - (B) **while** (!s.empty()){
   - (C) **while** (p.data!='S'){
   - (D) **while** (p.data=='X'){
   - [✓ (E) None] while (P!=null && p.data!='x'){
3. Line 5:
   - (A) **if** (p.right==**null**)
   - [✓ (B) **if** (p.right!=**null**)]
   - (C) **if** (p.right.data=='T')
   - (D) **if** (p.right.data=='S')
   - (E) None
4. Line 6:
   - (A) s.push('S');
   - (B) s.push(p);
   - (C) s.push(p.left);
   - [✓ (D) s.push(p.right);]
   - (E) None
5. Line 7:
   - (A) **if** (p.left.data=='T')
   - (B) **if** (p.left.data!='S')

- (C) `if (p.left==null)`
- [✓ (D) `if (p.left!=null)`]
- (E) None

6. Line 8:
   - [✓ (A) `p=p.left;`]
   - (B) `s.push(p.left);`
   - (C) `p=p.right;`
   - (D) `s.push('T');`
   - (E) None

7. Line 9:
   - (A) `else if (s.pop()=='X')`
   - (B) `else if (p.data == 'S')`
   - (C) `else if (st.full())`
   - [✓ (D) `else if (s.empty())`]
   - (E) None

8. Line 12:
   - [✓ (A) `p=s.pop();`]
   - (B) `s.push(p);`
   - (C) `p=s.push();`
   - (D) `p=t;`
   - (E) None

9. Line 14:
   - [✓ (A) `return p.data='X';`]
   - (B) `return p.data!='X';`
   - (C) `return p!=null;`
   - (D) `return p.data=='S';`
   - (E) None

Question 5 .... 10 points

(a) Choose the most appropriate answer.

1. In a min heap:

   (A) All keys at level k are smaller than all keys in level $k+1$. (B) The largest key is always at the last level. (C) The key of the left child of any node is smaller than the key of its right child. (D) All of the above. [✓ (E) None of the above].

2. The worst case run time for bottom-up heap construction (min or max) is:

   (A) $O(1)$. (B) $O(\log n)$. [✓ (C) $O(n)$]. (D) $O(n \log n)$ (E) $O(n^2)$.

3. The worst case run time for retrieving (without removing) the minimum key from a min heap is:

   [✓ (A) $O(1)$.] (B) $O(\log n)$. (C) $O(n)$. (D) $O(n \log n)$ (E) $O(n^2)$.

4. Suppose we are in the bottom-up heap constructing phase of heap-sort. So far, the array looks like this: 16, 14, 15, 10, 12, 27, 28. How many heap-down operations were done on the root?

   (A) 1. [✓ (B) 2.] (C) 3 (D) 4. (E) None.

(b) Consider the following heap represented as an array: 20, 18, 12, 10, 11, 5, 2, 6, 4, 3. Choose the correct answer for every operation (all operations are done on the above heap).

1. Heap after inserting 19:

   [✓ (A) 20, 19, 12, 10, 18, 5, 2, 6, 4, 3, 11] (B) 20, 19, 12, 11, 18, 5, 2, 6 ,4, 11, 3 (C) 20, 18, 12, 10, 11, 5, 2, 6, 4, 3, 19 (D) 20, 18, 12, 10, 19, 5, 2, 6, 4, 3,11 (E) None

2. Heap after inserting 1 **then** 23: (A) 20,18, 23, 10, 11, 12, 2, 6, 4, 3, 1, 5 (B) 20, 18, 12, 10, 11, 5, 2, 6, 4, 3, 1, 23 [✓ (C) 23, 18, 20, 10, 11, 12, 2, 6, 4, 3, 1, 5] (D) 23, 20, 18, 10, 11, 12, 2, 6, 4, 3, 1, 5 (E) None

3. Heap after deleting one key: (A) 20, 18, 12, 10, 11, 5, 2, 6, 4 (B) 3, 18, 12, 10, 11, 5, 2, 6, 4 (C) 12, 18, 5, 10, 11, 3, 2, 6, 4 [✓ (D) 18, 11, 12, 10, 3, 5, 2, 6, 4] (E) None

4. Heap after deleting two keys: (A) 4, 18, 12, 10, 11, 5, 2, 6 (B) 18, 11, 12, 10, 4, 5, 2, 6 (C) 12, 4, 5, 10, 11, 3, 2, 6 (D) 12, 11, 4, 10, 3, 5, 2, 6 [✓ (E) None]

Question 6 .... 10 points

(a) Choose the most appropriate answer:

1. Adjacency matrix takes $O(n^2)$ memory space, when is it appropriate to use it instead of adjacency list:

   Ⓐ The graph is sparse. [✓ Ⓑ The graph is dense.] Ⓒ The graph is unweighted Ⓓ The number of edges is less than the number of nodes. Ⓔ None.

2. You apply DFS from a given node and you find out that all nodes were visited. This means:

   Ⓐ The graph is directed. Ⓑ Some nodes have no edges. [✓ Ⓒ The graph is connected.] Ⓓ The graph contains cycles. Ⓔ None.

(b) Given the following graph adjacency matrix, answer the questions below.

	A	B	C	D	E	F
A		3	3			5
B	3		1	5		
C	3	1				
D		5				2
E						
F	5			2		

1. Which of the following sequences are simple paths in this graph? Answer by T (true) or F (false).
   (a) $(C, B, D, E,)$ F
   (b) (C, B, A) T
   (c) (C, A, F, D, B) T
   (d) (B, A, F, D, B, C) F
2. Answer by T (true) or F (false).
   (a) The graph is connected. F
   (b) The number of edges in the graph is 6. T
   (c) (A, B, D, F, A) is a cycle. T
   (d) The shortest path from F to B is (F, A, B). F
3. The BFS traversal of this graph starting from A is (insert neighbors in the data structure in increasing alphabetic order):

   Ⓐ A, D, B, C, F. Ⓑ A, B, C, D, F. [✓ Ⓒ A, B, C, F, D.] Ⓓ A, F, B, C, D. Ⓔ A, C, B, F, D.

4. The DFS traversal of this graph starting from A is (insert neighbors in the data structure in increasing alphabetic order):

   [✓ Ⓐ A, F, D, C, B.] Ⓑ A, F, C, D, B. Ⓒ A, B, C, F, D. Ⓓ A, F, B, C, D. Ⓔ A, C, B, F, D.