	King Saud University	College of Computer and Information Sciences Department of Computer Science
	Data Structures CSC 212	**Final Exam - Spring 2018**
	Date: 05/05/2018	Duration: 3 hours

Guidelines

- No calculators or any other electronic devices are allowed in this exam.
- Use a pencil in choice questions.

Student ID:	Name:
Section:	Instructor:

1	2.1	2.2	3.1	3.2	4	5	6	7	8	Total

Question 1 ..... 16 points

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

```
1  i = 1;
2  while(i < n)
3      i = i * 2;
```

1. Line 1: (A) 0 **(B) 1** (C) 2 (D) n (E) n^2
2. Line 2: (A) n (B) $n+1$ (C) $\log(n)$ **(D) $\log(n)+1$** (E) 2^n
3. Line 3: (A) n (B) $n+1$ **(C) $\log(n)$** (D) $\log(n)+1$ (E) 2^n
4. Total O: (A) 1 (B) n (C) n^2 **(D) $\log(n)$** (E) 2^n

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

```
1  c = 10;
2  for (i = 1; i <= c; i++)
3    for (j = 0; j < n; j++)
4      count++;
```

1. Line 1: (A) 0 **(B) 1** (C) 2 (D) n (E) n^2
2. Line 2: (A) n (B) c **(C) 11** (D) 10 (E) 9
3. Line 3: (A) n (B) $10n$ **(C) $10(n+1)$** (D) c (E) n^2
4. Line 4: (A) $count+2$ **(B) $10n$** (C) $11n$ (D) n^2 (E) $n(n+1)/2$
5. Total O: (A) 1 **(B) n** (C) n^2 (D) $n\log(n)$ (E) n^3

(c) Choose the correct answer:

1. $n^3+n^2\log n$ is : **(A) $O(n^3)$** (B) $O(n^2)$ (C) $O(n^2\log(n))$ (D) $O(n^5)$ (E) None
2. 2^n+n^n is : (A) $O(n)$ (B) $O(n^2)$ (C) $O(2^n)$ **(D) $O(n^n)$** (E) None
3. $n^4\log n+2^n$ is : (A) $O(n)$ (B) $O(n^4)$ (C) $O(n^5)$ (D) $O(\log(n))$ **(E) None**

4. When traversing all nodes in a binary tree of depth d. The complexity would be:

   (A) $O(d)$ (B) $O(d^2)$ (C) $O(2^d)$ (D) $O(\log(d))$ (E) None

Question 2 ..... 10 points

(a) Given a map of queues of call records, we want to find out if there was call from a given number to another. Write the method **boolean** `callFrom(Map<String, Queue<Record>> m, String nb1, String nb2)`, which checks if there was a call from `nb1` to `nb2` without changing `m`. The map `m` is indexed by the **caller** number.

```
public interface Map<K extends Comparable<
    K>, T> {
  boolean empty();
  boolean full();
  T retrieve();
  void update(T e);
  boolean find(K key);
  boolean insert(K key, T data);
  boolean remove(K key);
}
```

```
public class Record {
  public String from, to;
  public Date start, end;
  ...
}
```

Complete the code below by choosing the correct answer:

```
1  boolean callFrom(Map<String, Queue<Record>> m, String nb1, String nb2) {
2    if (...)
3      ...;
4    Queue<Record> q = ...;
5    boolean found = ...;
6    ... {
7      Record r  = ...;
8      ...;
9      if (...)
10       ...;
11   }
12   ...;
13 }
```

- Line 2:
  - (A) `!m.find(nb2)`
  - (B) `m.find(nb1)`
  - (C) `!m.find(nb1)`
  - (D) `m.find(nb2)`
  - (E) None

- Line 3:
  - (A) **return true**
  - (B) **return** `m.find(nb1)`
  - (C) **return false**
  - (D) **return** `!m.find(nb1)`
  - (E) None

- Line 4:
  - (A) `m.enqueue(nb2)`
  - (B) `m.retrieve()`
  - (C) `m.retrieve(nb1)`
  - (D) `m.find(nb2)`
  - (E) None

- Line 5:
  - (A) **false**
  - (B) `m.retrieve().serve()`
  - (C) **true**
  - (D) `m.find(nb1)`
  - (E) None

- Line 6:
  - (A) `for (int i = q.length(); i >=0 ; i--)`
  - **(B) `for (int i = 1; i <= q.length(); i++)`**
  - (C) `while (q.length()> 0)`
  - (D) `for (int i = 0; i <= q.length(); i++)`
  - (E) None
- Line 7:
  - **(A) `q.serve()`**
  - (B) `q.serve(nb2)`
  - (C) `m.retrieve()`
  - (D) `q.head.data`
  - (E) None
- Line 8:
  - **(A) `q.enqueue(r)`**
  - (B) `m.enqueue(r)`
  - (C) `q.enqueue()`
  - (D) `q.serve(r)`
  - (E) None
- Line 9:
  - (A) `nb2 == r.to`
  - (B) `nb1.equals(r.to)`
  - (C) `nb1 == r.from`
  - (D) `nb2.equals(r.from)`
  - **(E) None**
- Line 10:
  - (A) `found = (nb2 == r.to)`
  - (B) `found = false`
  - **(C) `found = true`**
  - (D) `found = !found`
  - (E) None
- Line 12:
  - **(A) `return found`**
  - (B) `return found || (q.length()== 0)`
  - (C) `return false`
  - (D) `return true`
  - (E) None

(b) Given a queue of stack of call records, we want to find out if there was any call to a given number. Write the method **`boolean anyCallTo(Queue<Stack<Record>> q, String nb)`**, which checks if there was any call to `nb` without changing `q`.

Complete the code below by choosing the correct answer:

```
1  boolean anyCallTo(Queue<Stack<Record>> q, String nb) {
2    boolean found = false;
3    ... {
4      Stack<Record> st = q.serve();
5      ...;
6      if (...) {
7        Stack<Record> ts = new LinkedStack<Record>();
8        while (...) {
9          Record r = ...;
10         ...;
11         if (...)
12           ...;
13       }
14       while (...)
15         ...;
16     }
17   }
18   return found;
19 }
```

- Line 3:
  - (A) `while (i < q.length())`
  - (B) `while (q.length()> 0)`
  - (C) `for (int i = 1; i <= q.length(); i--)`
  - **(D) `for (int i = 0; i < q.length(); i++)`**
  - (E) None
- Line 5:
  - (A) `q.serve()`
  - (B) `st.push(q)`
  - (C) `q.enqueue()`
  - (D) `st.pop()`
  - **(E) None**
- Line 6:
  - (A) `found && st.empty()`
  - (B) `found`
  - **(C) `!found`**
  - (D) `found && !st.empty()`
  - (E) None
- Line 8:
  - **(A) `!st.empty()&& !found`**
  - (B) `!found`
  - (C) `st.empty()|| found`
  - (D) `st.empty()&& !found`
  - (E) None
- Line 9:
  - (A) `st.push()`
  - **(B) `st.pop()`**
  - (C) `q.serve()`
  - (D) `st.serve()`
  - (E) None
- Line 10:
  - **(A) `ts.push(r)`**
  - (B) `ts.push(st.pop())`
  - (C) `st.push(r)`
  - (D) `ts.pop()`
  - (E) None
- Line 11:
  - (A) `nb.equals(r.from)`
  - (B) `nb == r.from`
  - (C) `nb == r.to`
  - **(D) `nb.equals(r.to)`**
  - (E) None
- Line 12:
  - (A) `found = (nb == r.to)`
  - (B) **`return true`**
  - **(C) `found = true`**
  - (D) `found = false`
  - (E) None
- Line 14:
  - (A) `ts.empty()`
  - (B) `!st.empty()`
  - **(C) `!ts.empty()`**
  - (D) `st.empty()`
  - (E) None
- Line 15:
  - (A) `st.push(q.serve().pop())`
  - (B) `st.push(st.pop())`
  - (C) `ts.push(st.pop())`
  - **(D) `st.push(ts.pop())`**
  - (E) None

Question 3 ..... 10 points

(a)
- Line 2:
  - (A) **`return ((isPTRec(root.left))&&(isPTRec(root.right)));`**
  - (B) **`return ((isPathTree(root.left))&&(isPathTree(root.right)));`**
  - (C) **`return ((isPTRec(current.left))&&(isPTRec(current.right)));`**
  - **(D) `return isPTRec(root);`**

(E) None

- Line 5:
  - (A) `if (p != null)return true;`
  - **(B) `if (p == null)return true;`**
  - (C) `if (root == null)return true;`
  - (D) `if (p != null)return false;`
  - (E) None
- Line 6:
  - (A) `if ((p.left==null)||(p.right==null))return false;`
  - (B) `if ((p.left!=null)||(p.right!=null))return false;`
  - (C) `if ((p.left==null)&&(p.right==null))return true;`
  - **(D) `if ((p.left!=null)&&(p.right!=null))return false;`**
  - (E) None
- Line 7:
  - (A) `return true;`
  - **(B) `return isPTRec(p.left)&& isPTRec(p.right);`**
  - (C) `return isPTRec(p.left)|| !isPTRec(p.right);`
  - (D) `return !isPTRec(p.left)|| !isPTRec(p.right);`
  - (E) None

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

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

  if(p != null && p.next != null){
    q = p;
    while(q.next != null)
      q = q.next;
    q.previous.next = null;
    q.previous = null;
    q.next = p;
    p.previous = q;
    head = q;
  }
}
```

Choose the correct result in each of the following cases:

1. The list `l`: A, B, C, D, E, after calling `l.f(1)`, `l` becomes:
   (A) B, C, D, E (B) A, B, E, C, D **(C) E, B, C, D** (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) E, A, B, C, D** (C) B, C, D, E, A (D) A, B, C, D, E (E) None
3. The list `l`: A, B, C, D, E, after calling `l.f(2)`, `l` becomes:
   (A) *empty* (B) E, D, C, B, A (C) A, D, E, B, C **(D) E, C, D** (E) None
4. The list `l`: A, B, C, D, E, after calling `l.f(5)`, `l` becomes:
   (A) A (B) E, A, B, C, D (C) C, D, E, A, B **(D) A, B, C, D, E** (E) None

Question 4 ..... 14 points

(a) Consider the following heap represented as an array: 4, 16, 14, 22, 20, 18. Choose the correct answer for every operation (all operations are done on the above heap).

1. Heap after inserting 6: (A) 4,16,14,22,20,18,6 (B) 4,6,16,22,20,18,14 (C) 4,16,14,22,20,6,18 (D) 6,16,4,22,20,18,14 **(E) None**
2. Heap after inserting 16: **(A) 4,16,14,22,20,18,16** (B) 4,16,16,22,20,18,14 (C) 4,16,14,22,20,16,18 (D) 4,16,14,16,20,18,22 (E) None
3. Heap after inserting 0: (A) 4,16,14,22,20,18,0 **(B) 0,16,4,22,20,18,14** (C) 4,16,0,22,20,18,14 (D) 0,16,4,22,20,14,18 (E) None
4. Heap after deleting one key: (A) 16,22,14,18,20 **(B) 14,16,18,22,20** (C) 16,14,20,18,22 (D) 16,20,14,22,18 (E) None
5. Heap after deleting two keys: (A) 16,22,14,18 (B) 20,18,14,22 (C) 16,20,14,22 (D) 22,14,20,18 **(E) None**

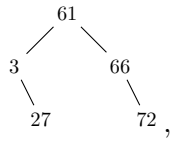
(b) What is the result of a bottom-up min-heap construction of the array: 1,5,11,4,6,0? (A) 0,1,4,6,5,11 (B) 1,0,4,5,6,11 **(C) 0,4,1,5,6,11** (D) 0,4,1,6,11,5 (E) None.

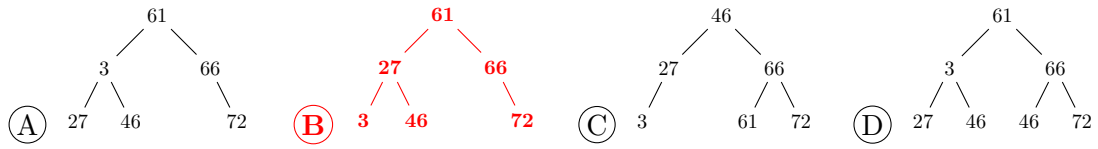
(c) Choose the correct answer:

1. Bottom-up heap construction is: **(A)** $O(n)$ (B) $O(\log n)$ (C) $O(n^2 \log n)$ (D) $O(n^2)$ (E) None.
2. The serve operation in a heap priority queue is: (A) $O(1)$ **(B)** $O(\log n)$ (C) $O(n)$ (D) $O(n \log n)$ (E) None.
3. What is the minimum number of nodes in a heap of height k? (A) $2^k - 1$ (B) $\log k$ **(C)** 2^k (D) 2^{k-1} (E) None.

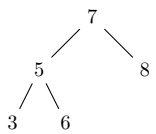
Question 5 ..... 14 points

Choose the correct result in each of the following cases (follow the the convention of replacing with the smallest key in the right sub-tree when necessary):

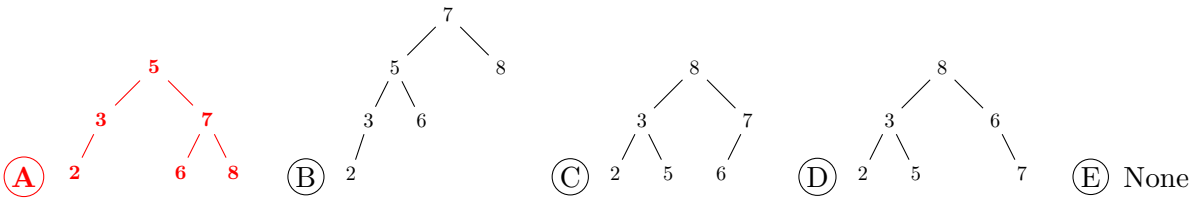
1. After inserting the key 46 in the AVL 

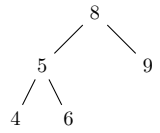
, the tree becomes:


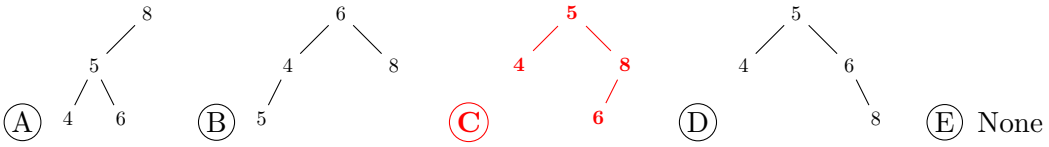
(E) None

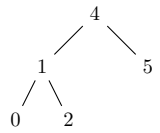
2. After inserting the key 2 in the AVL 

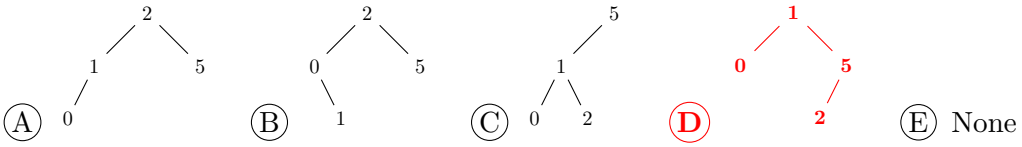
, the tree becomes:

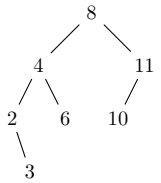


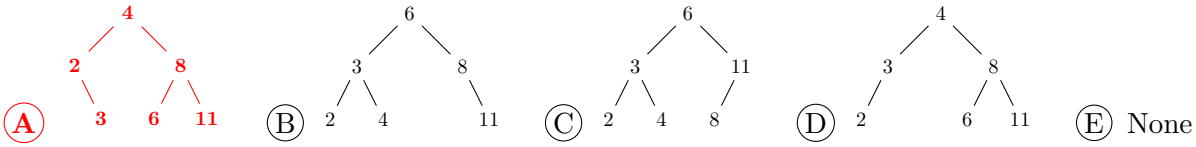
3. After deleting the key 9 from the AVL  , the tree becomes:

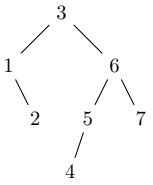


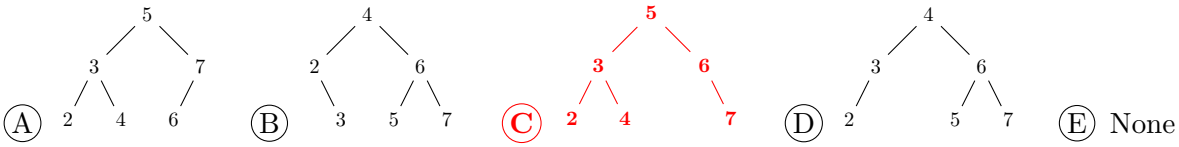
4. After deleting the key 4 from the AVL  , the tree becomes:

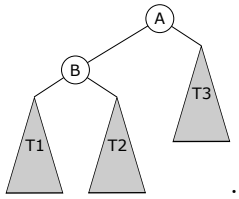


5. After deleting the key 10 from the AVL  , the tree becomes:



6. After deleting the key 1 from the AVL  , the tree becomes:



7. Consider the following tree  . If the balance of A is -2 and that of B is -1, then after performing a single right rotation at A, then:

   (a) The balance of A becomes: ☐

   (b) The balance of B becomes: ☐

Question 6 ..... 14 points

Use the hash function $H(key) = key\%10$ to store the sequence of keys $14, 15, 4, 16, 27, 20, 35, 47, 10, 7$ in a hash table of size 10. Use the following collision resolution strategies:

1. Linear rehashing (c=2). Fill in the following table:

Key	14	15	4	16	27	20	35	47	10	7
Position										
Number of probes										

2. External chaining. Fill in the following table:

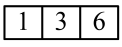
Key	14	15	4	16	27	20	35	47	10	7
Position of the list										

3. Coalesced chaining with cellar size 3 and address region size 7 (you must change the hash function to $H(key) = key\%7$.) Fill in the following table (put -1 if there is no next element):

Key	14	15	4	16	27	20	35	47	10	7
Position										
Index of next element										

Question 7 ..... 14 points

Choose the correct result in each of the following cases (when possible, always borrow and transfer to the left):



1. After inserting the key 7 in the B+ tree , the **root** of tree becomes:

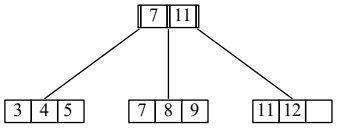
(A) 

(B) 

(C) 

(D) 

(E) None



2. After inserting the key 14 in the B+ tree , the **root** of the tree becomes:

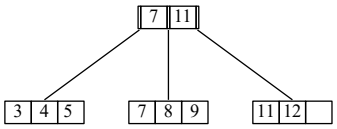
(A) 

(B) 

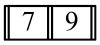
(C) 

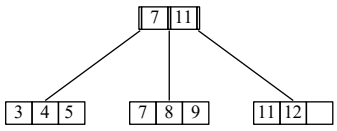
(D) 

(E) None



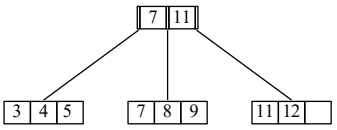
3. After inserting the key 10 in the B+ tree , the **root** of the tree becomes:

(A)  (B)  (C)  (D)  (E) None



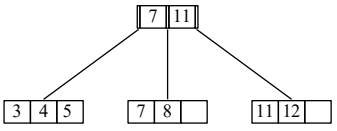
4. After deleting the key 9 from the B+ tree , the **root** of the tree becomes:

(A)  (B)  (C)  (D)  (E) None

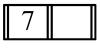
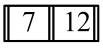
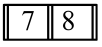


5. After deleting the key 12 from the B+ tree , the **root** of the tree becomes:

(A)  (B)  (C)  (D)  (E) None



6. After deleting the key 11 from the B+ tree , the **root** of the tree becomes:

(A)  (B)  (C)  (D) (E) None

7. A B+ tree of order 4 leaves can contain the following number of data elements:

(A) 3 to 4 elements (B) **2 to 4 elements** (C) 1 to 4 elements (D) 4 to 4 elements (E) None

Question 8 ..... 8 points

1. Given the following adjacency matrix, draw the weighted graph it represents.

	0	1	2	3	4	5
0		1	2			
1	1		1	3		
2	2	1		4		
3		3	4			
4						3
5					3	

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

2. Give the adjacency list representation of the graph.

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

3. What is the cycle with the largest number nodes in the graph? What is its total weight?

. . . . . .

. . . . . .

. . . . . .

ADT Queue Specification

- enqueue (Type e): **requires**: Queue Q is not full. **input**: Type e. **results**: Element e is added to the queue at its tail. **output**: none.
- serve (Type e): **requires**: Queue Q is not empty. **input**: none. **results**: the element at the head of Q is removed and its value assigned to e. **output**: Type e.
- length (int length): **requires**: none. **input**: none. **results**: The number of elements in the Queue Q is returned. **output**: length.
- full (boolean flag): **requires**: none. **input**: none. **results**: If Q is full then flag is set to true, otherwise flag is set to false. **output**: flag.

ADT Stack Specification

- push(Type e): **requires**: Stack S is not full. **input**: Type e. **results**: Element e is added to the stack as its most recently added elements. **output**: none.
- pop(Type e): **requires**: Stack S is not empty. **input**: **results**: the most recently arrived element in S is removed and its value assigned to e. **output**: Type e.
- empty(boolean flag): **requires**: none. **input**: none. **results**: If Stack S is empty then flag is true, otherwise false. **output**: flag.
- full(boolean flag): **requires**: none. **input**: none. **results**: If S is full then Full is true, otherwise Full is false. **output**: flag.