	King Saud University	College of Computer and Information Sciences Department of Computer Science
	Data Structures CSC 212	**Final Exam - Fall 2017**
	Date: 30/12/2017	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 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) 3 (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) $count+2$ (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 answer:

1. $n^2 + n \log n^4$ is : (A) $O(n)$ (B) $O(n^2)$ (C) $O(n \log(n))$ (D) $O(n^4)$ (E) None
2. $n^2 + 1000n$ is : (A) $O(n)$ (B) $O(n^2)$ (C) $O(n \log(n))$ (D) $O(nn^2)$ (E) None
3. $n^4 \log n + n!$ is : (A) $O(n!)$ (B) $O(n^4)$ (C) $O(n^5)$ (D) $O(\log(n))$ (E) None
4. 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 ..... 10 points

(a) Given a queue of time intervals represented as pairs of integers, write the method **public static** `Queue<Pair<Integer, Integer>> getIntervals(Queue<Pair<Integer, Integer>> q, int start, int end)`, which returns all intervals of `q` intersecting the interval [start, end]. The input `q` must not change. Assume that all intervals in `q` are valid (that is `first <= second`), non-overlapping and ordered in chronological order. The class `Pair` is given below.

```
public class Pair<U, V> {
  public U first;
  public V second;
  Pair(U first, V second) {
    this.first = first;
    this.second = second;
  }
}
```

(b) Consider a stack of decreasing time intervals, that is, starting from the top, each interval contains the next. Write the method **public static** `Pair<Integer, Integer> smallest(Stack<Pair<Integer, Integer>> st,` **int** `t)`, which returns the smallest interval containing `t` if it exists, **null** otherwise. Assume that all intervals in `st` are valid (that is `first <= second`).

Example 0.1. *If* `st` *:* $\{[0,8],[1,6],[1,5],[2,4]\}$*, then* `smallest(st, 1)` *returns* $[1,5]$*,* `smallest(st, 3)` *returns* $[2,4]$*,* `smallest(st, 9)` *returns* **null***.*

Complete the code below by choosing the correct answer:

```
1  public static Pair<Integer, Integer>smallest(Stack<Pair<Integer, Integer>> st, int t){
2    ...
3    Pair<Integer, Integr> itm = null;
4    while (!st.empty()) {
5      Pair<Integer, Integr> it = st.pop();
6      ...
7      if (...)
8        itm = it;
9      else
10       ...
11   }
12   while (...) {
13     ...
14   }
15   return itm; }
```

1. Line 2:
   - (A) `Queue<Pair<Integer, Integer>> r = new LinkedQueue<Pair<Integer, Integer>>();`
   - (B) `Stack<Integer> r = new LinkedStack<Integer>();`
   - (C) `List<Pair<Integer, Integer>> r = new LinkedList<Pair<Integer, Integer>>();`
   - (D) `Stack<Pair<Integer, Integer>> r = new LinkedStack<Pair<Integer, Integer>>();`
   - (E) None
2. Line 6:
   - (A) `r.push(it.first);`
   - (B) `r.insert(it);`
   - (C) `r.enqueue(it);`
   - (D) `r.push(it);`
   - (E) None
3. Line 7:
   - (A) **if** `(it.first < t && t <= it.second)`
   - (B) **if** `(it.first <= t && t <= it.second)`
   - (C) **if** `(it.first < t || it.second > t)`
   - (D) **if** `(it.first <= t && it.second <= t)`
   - (E) None
4. Line 10:
   - (A) `r.serve();`
   - (B) **break**;
   - (C) `r.pop();`
   - (D) `r.findNext();`
   - (E) None
5. Line 12:
   - (A) **while** `(r.empty()){`
   - (B) **while** `(!r.empty()){`
   - (C) **while** `(r.pop()!= null){`
   - (D) **while** `(r.length()!= 0){`
   - (E) None
6. Line 13:
   - (A) `st.push(r.pop());`
   - (B) `st.push(r.serve());`
   - (C) `st.push(r.retrieve()); r.findNext();`
   - (D) `st.push(r.push());`
   - (E) None

Question 3 ..... 10 points

(a) The method **private** `BTNode<T> mirrorCopy(BTNode<T> t)` creates **recursively** a mirror copy of the subtree `t`. Choose the correct option to complete the code of this method:

```
1 private BTNode<T> mirrorCopy(BTNode<T> t) {
2   ...
3   ...
4   ...
5   ...
6   ...
7   ...
8 }
```

1. Line 2:
   - (A) **if** `(t.left == null || t.right == null)`
   - (B) **if** `(t.left == null && t.right == null)`
   - (C) **if** `(t == null)`
   - (D) **if** `(root != null)`
   - (E) None
2. Line 3:
   - (A) **return null**;
   - (B) **return** `root;`
   - (C) **return** `mirrorCopy(root);`
   - (D) **return** `mirrorCopy(t);`
   - (E) None
3. Line 4:
   - (A) `BTNode<T> p = new BTNode<T>(t.data);`
   - (B) `BTNode<T> p = new BTNode<T>(root);`
   - (C) `BTNode<T> p = new BTNode<T>(t);`
   - (D) `BTNode<T> p = new BTNode<T>(root.data);`
   - (E) None
4. Line 5:
   - (A) `p.right = mirrorCopy(t.left);`
   - (B) `t.left = mirrorCopy(t.left);`
   - (C) `p.right = mirrorCopy(t.right);`
   - (D) `t.left = mirrorCopy(t.right);`
   - (E) None
5. Line 6:
   - (A) `t.right = mirrorCopy(t.left);`
   - (B) `p.left = mirrorCopy(t.left);`
   - (C) `p.left = mirrorCopy(t.right);`
   - (D) `t.right = mirrorCopy(t.right);`
   - (E) None
6. Line 7:
   - (A) **return** `p;`
   - (B) **return** `mirrorCopy(t);`
   - (C) `mirrorCopy(t.left); mirrorCopy(t.right);`
   - (D) **return** `t;`
   - (E) None

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

```
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) {
    p.previous = 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, B Ⓑ D, E, A, B, C Ⓒ C, D, E, A, B Ⓓ A, D, E, B, C Ⓔ None
2. The list `l`: A, B, C, D, E, after calling `l.f(0)`, `l` becomes:
   Ⓐ *empty* Ⓑ A, B, C, D, E Ⓒ B, C, D, E, A Ⓓ B, C, D, E Ⓔ None
3. The list `l`: A, B, C, D, E, after calling `l.f(5)`, `l` becomes:
   Ⓐ *empty* Ⓑ A, B, C, D, E Ⓒ E, D, C, B, A Ⓓ A, D, E, B, C Ⓔ None
4. The list `l`: A, B, C, D, E, after calling `l.f(1)`, `l` becomes:
   Ⓐ A Ⓑ E, A, B, C, D Ⓒ C, D, E, A, B Ⓓ B, C, D, E, A Ⓔ None

Question 4 ..... 14 points

(a) Consider the following heap represented as an array: 3, 7, 9, 13, 8, 11. Choose the correct answer for every operation (all operations are done on the above heap).

1. Heap after inserting 5:
   Ⓐ 3,7,9,13,8,11,5 Ⓑ 3,5,7,13,8,11,9 Ⓒ 3,7,9,13,8,5,11 Ⓓ 5,7,3,13,8,11,9 Ⓔ None
2. Heap after inserting 10:
   Ⓐ 3,7,10,13,8,11,9 Ⓑ 3,7,9,13,8,10,11 Ⓒ 3,7,9,13,8,11,10 Ⓓ 3,7,9,10,8,11,13 Ⓔ None
3. Heap after inserting 2:
   Ⓐ 3,7,9,13,8,11,2 Ⓑ 3,7,2,13,8,11,9 Ⓒ 2,7,3,13,8,9,11 Ⓓ 2,7,3,13,8,11,9 Ⓔ None
4. Heap after deleting one key:
   Ⓐ 7,13,9,11,8 Ⓑ 7,8,9,13,11 Ⓒ 9,7,11,13,8 Ⓓ 7,9,8,11,13 Ⓔ None
5. Heap after deleting two keys:
   Ⓐ 7,13,9,11 Ⓑ 8,11,9,13 Ⓒ 7,8,9,13 Ⓓ 13,9,8,11 Ⓔ None

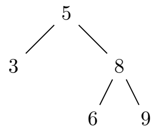
(b) What is the result of a bottom-up min-heap construction of the following array: 2,4,6,3,5,1?
Ⓐ 1,2,3,5,4,6 Ⓑ 2,1,3,4,5,6 Ⓒ 1,3,2,4,5,6 Ⓓ 1,3,2,5,6,4 Ⓔ None.

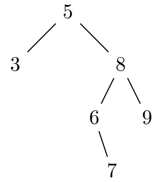
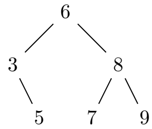
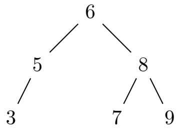
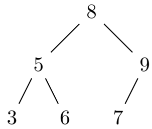
(c) Choose the correct answer:

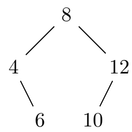
1. What is the height of a heap of size k?
   Ⓐ $\log\log k$ Ⓑ $k/2$ Ⓒ $k \log k$ Ⓓ $\log k$ Ⓔ None.
2. Bottom-up heap construction is:
   Ⓐ $O(n)$ Ⓑ $O(\log n)$ Ⓒ $O(n \log n)$ Ⓓ $O(n^2)$ Ⓔ None.
3. The enqueue operation in a heap priority queue is:
   Ⓐ $O(1)$ Ⓑ $O(\log n)$ Ⓒ $O(n)$ Ⓓ $O(n \log n)$ Ⓔ None.

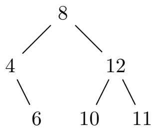
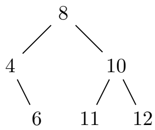
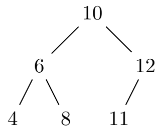
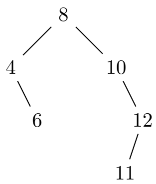
Question 5 .... 14 points

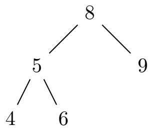
Choose the correct result in each of the following cases:

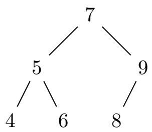
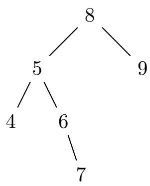
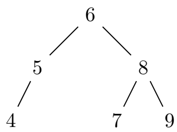
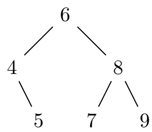
1. After inserting the key 7 in the AVL  , the tree becomes:

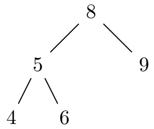
Ⓐ  Ⓑ  Ⓒ  Ⓓ  Ⓔ None

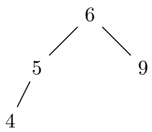
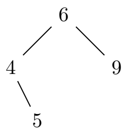
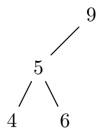
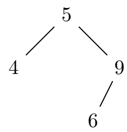
2. After inserting the key 11 in the AVL  , the tree becomes:

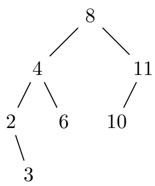
Ⓐ  Ⓑ  Ⓒ  Ⓓ  Ⓔ None

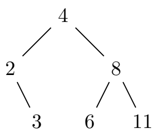
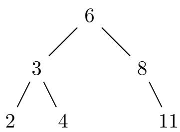
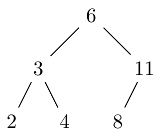
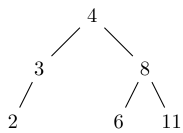
3. After inserting the key 7 in the AVL  , the tree becomes:

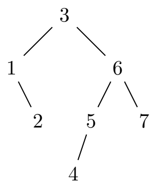
Ⓐ  Ⓑ  Ⓒ  Ⓓ  Ⓔ None

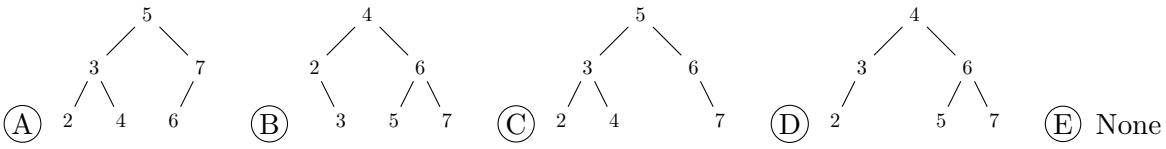
4. After deleting the key 8 from the AVL  , the tree becomes:

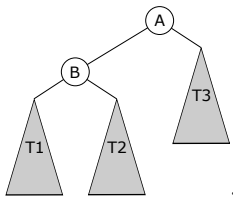
Ⓐ  Ⓑ  Ⓒ  Ⓓ  Ⓔ None

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

Ⓐ  Ⓑ  Ⓒ  Ⓓ  Ⓔ None

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 0, 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 $H(key) = key\%11$ to store the sequence of keys $16, 14, 27, 5, 21, 43, 10, 38, 19, 18, 20$ in the hash table. Use the following collision resolution strategies:

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

Key	16	14	27	5	21	43	10	38	19	18	20
Position											
Number of probes											

2. External chaining. Fill in the following table:

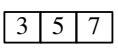
Key	16	14	27	5	21	43	10	38	19	18	20
List position											

3. Coalesced chaining with cellar size 2 (do not change the hash function). Fill in the following table (put -1 if there is no next element):

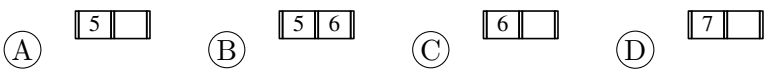
Key	16	14	27	5	21	43	10	38	19	18	20
Position											
Next											

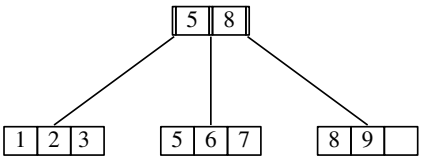
Question 7 ..... 14 points

Choose the correct result in each of the following cases:

1. After inserting the key 6 in the B+ tree 

, the **root** of tree becomes:



2. After inserting the key 4 in the B+ tree 

, the **root** of the tree becomes:

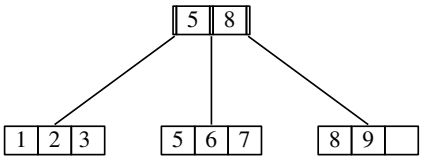
Ⓐ 

Ⓑ 

Ⓒ 

Ⓓ 

Ⓔ None

3. After inserting the key 10 in the B+ tree 

, the **root** of the tree becomes:

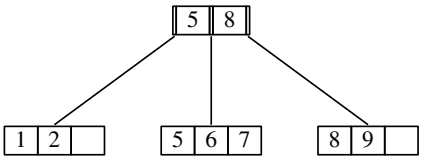
Ⓐ 

Ⓑ 

Ⓒ 

Ⓓ 

Ⓔ None

4. After deleting the key 2 from the B+ tree 

, the **root** of the tree becomes:

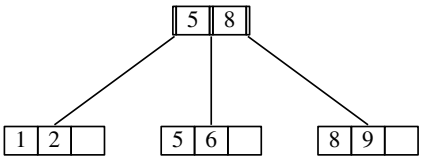
Ⓐ 

Ⓑ 

Ⓒ 

Ⓓ 

Ⓔ None

5. After deleting the key 5 from the B+ tree 

, the **root** of the tree becomes:

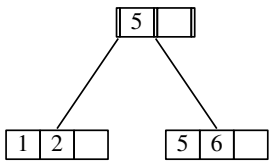
Ⓐ 

Ⓑ 

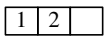
Ⓒ 

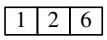
Ⓓ 

Ⓔ None

6. After deleting the key 5 from the B+ tree 

, the **root** of the tree becomes:

Ⓐ 

Ⓑ 

Ⓒ 

Ⓓ 

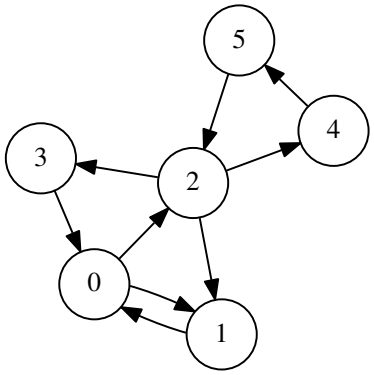
Ⓔ None

7. A B+ tree of order 3 containing n keys has a height that is:

Ⓐ $O(n)$ Ⓑ $O(n/3)$ Ⓒ $O(n^3)$ Ⓓ $O(\log n)$ Ⓔ None

Question 8 ..... 8 points

Consider the following graph.



1. Give the adjacency matrix of the graph.

   .....
   .....
   .....
   .....
   .....
   .....
   .....
   .....

2. Give the adjacency list representation of the graph.

   .....
   .....
   .....
   .....
   .....
   .....

3. What is the number of edges in the subgraph containing the nodes $\{1, 2, 3\}$.

   .....

4. What is the maximum number of edges in a directed graph with n nodes (loops, edges from a node to itself, are not allowed)?

   .....

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.