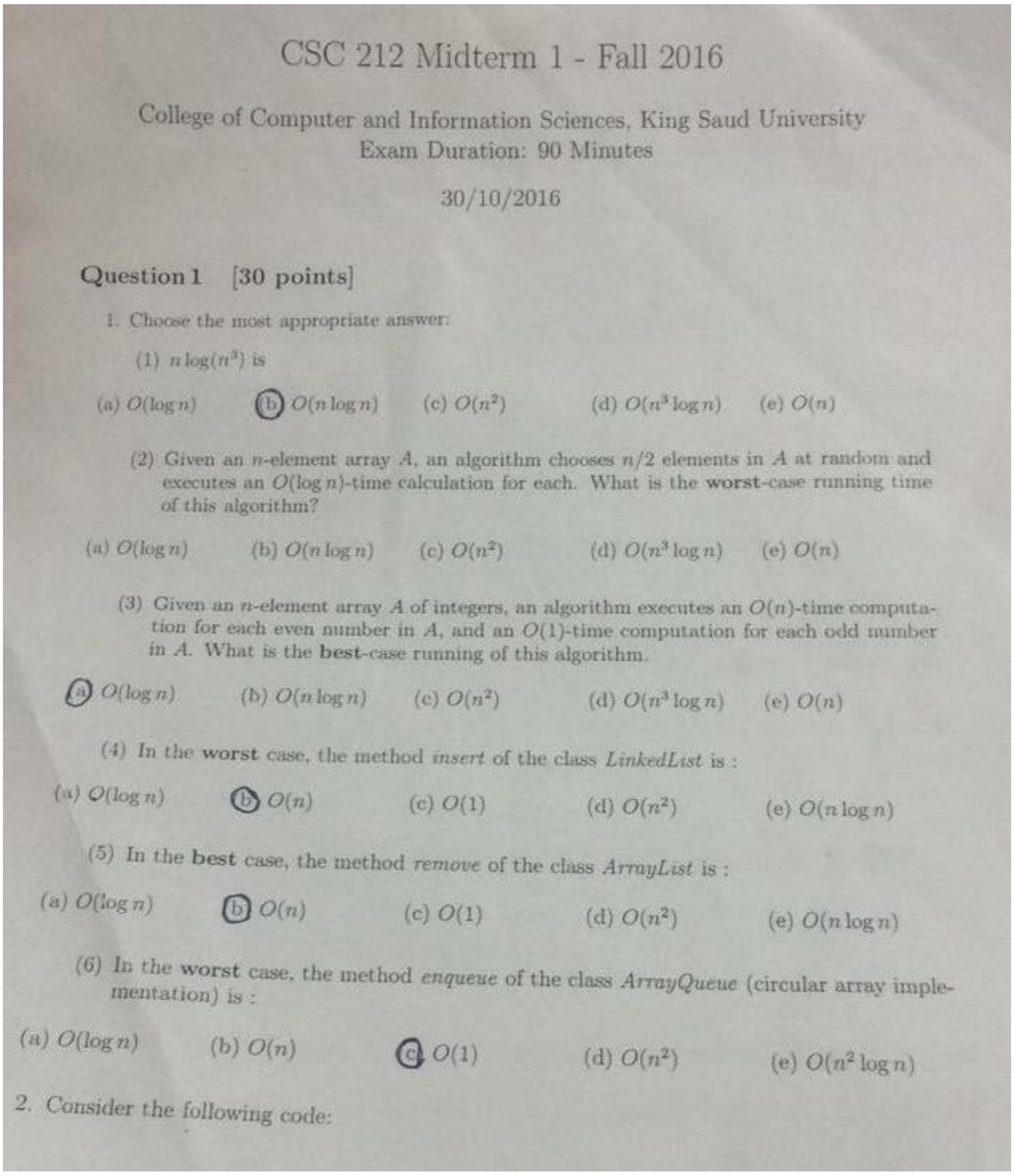
CSC212-Mid 1 – Fall2016



CSC 212 Midterm 1 - Fall 2016

College of Computer and Information Sciences, King Saud University
Exam Duration: 90 Minutes

30/10/2016

Question 1 [30 points]

1. Choose the most appropriate answer:

(1) $n \log(n^3)$ is

(a) $O(\log n)$ [✓ (b) $O(n \log n)$] (c) $O(n^2)$ (d) $O(n^3 \log n)$ (e) $O(n)$

(2) Given an n-element array A, an algorithm chooses $n/2$ elements in A at random and executes an $O(\log n)$-time calculation for each. What is the **worst**-case running time of this algorithm?

(a) $O(\log n)$ (b) $O(n \log n)$ (c) $O(n^2)$ (d) $O(n^3 \log n)$ (e) $O(n)$

(3) Given an n-element array A of integers, an algorithm executes an $O(n)$-time computation for each even number in A, and an $O(1)$-time computation for each odd number in A. What is the **best**-case running of this algorithm.

[✓ (a) $O(\log n)$] (b) $O(n \log n)$ (c) $O(n^2)$ (d) $O(n^3 \log n)$ (e) $O(n)$

(4) In the **worst** case, the method *insert* of the class *LinkedList* is :

(a) $O(\log n)$ [✓ (b) $O(n)$] (c) $O(1)$ (d) $O(n^2)$ (e) $O(n \log n)$

(5) In the **best** case, the method *remove* of the class *ArrayList* is :

(a) $O(\log n)$ [✓ (b) $O(n)$] (c) $O(1)$ (d) $O(n^2)$ (e) $O(n \log n)$

(6) In the **worst** case, the method *enqueue* of the class *ArrayQueue* (circular array implementation) is :

(a) $O(\log n)$ (b) $O(n)$ [✓ (c) $O(1)$] (d) $O(n^2)$ (e) $O(n^2 \log n)$

2. Consider the following code:

```
1 int sum = 0;
2 for (int i = 0; i < n * n; i++)
3         for (int j = n; j < 2 * n; j++)
4                 sum += j;
5 return sum;
```

n^2-0+1 n^2+1

n^2

n

$2n-n+1$ $n+1$

n^3+n^2

Choose the correct answer:

Line			Frequency		
1	(a) n	[✓ (b) 1]	(c) n^2	(d) 0	(e) i
2	(a) n	(b) $n+1$	(c) n^2	(d) $n-1$	[✓ (e) n^2+1]
3	(a) $i \times n^2$	(b) n^3	(c) n^2+1	[✓ (d) n^3+n^2]	(e) $n(n+1)/2$
4	(a) n^4	(b) n^3-1	[✓ (c) n^2]	(d) n^3-n	(e) n^3
5	(a) n	(b) n^3	(c) n^2	[✓ (d) 1]	(e) n
Total O	(a) $O(n)$	(b) $O(n^2)$	[✓ (c) $O(n^3)$]	(d) $O(n^4)$	(e) $O(1)$

Question 2 [35 points]

L.

1. As a user of ADT List, write the method *removeDuplicate* that keeps only the first occurrence of an element k and removes all its remaining occurrences. Assume the list l is not empty. The signature is *public static <T>void removeDuplicate(List<T>l, T k).* **Do not use any auxiliary data structures.**

   Example 2.1. *If list l is $l : A \to B \to C \to B \to B$ and the method accepts* removeDuplicate(l, "B"), *then l will be changed to $l : A \to B \to C$.*

2. Write the method *copy*, user of the ADT Queue, to copy the i-th element to the j-th element in a queue, assume the positions i and j are valid and that $i < j$ (numbering starts at 0 at the head). Signature: *public static <T>void copy(Queue<T>q, int i, int j).* **Do not use any auxiliary data structures.**

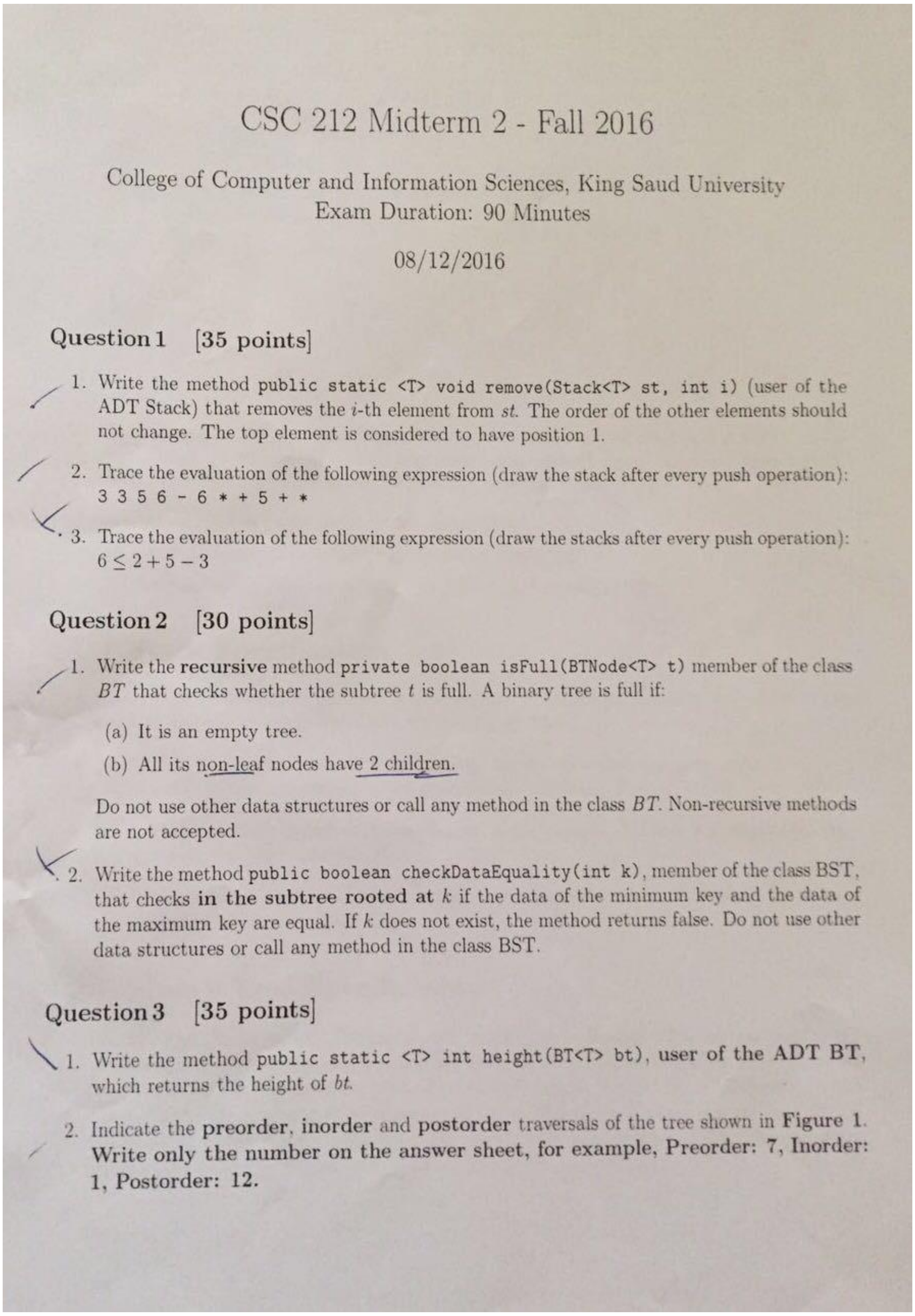
   Example 2.2. *If q is $A \to B \to C \to D \to E \to F$, then after calling* copy(q, 1, 4), *then q becomes $A \to B \to C \to D \to B \to F$.*

Question 3 [35 points]

Suppose we want to add to the specification of the the ADT List the method *insertAll* that inserts an array of elements into the list:

- *insert(T e[], int n)*:
  requires: list L is not full. $n \leq$ length of e.
  input: e: array of elements, n: number of elements to add.
  results:
  - The elements of e are inserted one by one until n elements are inserted or the list becomes full.
  - If the list is not empty, the elements of e are inserted after *current*, otherwise they are inserted at the beginning of the list.
  - The last element added to the list is made *current*.

CSC212-Mid 2 – Fall2016



CSC 212 Midterm 2 - Fall 2016

College of Computer and Information Sciences, King Saud University
Exam Duration: 90 Minutes

08/12/2016

Question 1 [35 points]

1. Write the method `public static <T> void remove(Stack<T> st, int i)` (user of the ADT Stack) that removes the *i*-th element from *st*. The order of the other elements should not change. The top element is considered to have position 1.
2. Trace the evaluation of the following expression (draw the stack after every push operation): `3 3 5 6 - 6 * + 5 + *`
3. Trace the evaluation of the following expression (draw the stacks after every push operation): $6 \leq 2 + 5 - 3$

Question 2 [30 points]

1. Write the **recursive** method `private boolean isFull(BTNode<T> t)` member of the class *BT* that checks whether the subtree *t* is full. A binary tree is full if:
   (a) It is an empty tree.
   (b) All its non-leaf nodes have 2 children.

   Do not use other data structures or call any method in the class *BT*. Non-recursive methods are not accepted.
2. Write the method `public boolean checkDataEquality(int k)`, member of the class BST, that checks **in the subtree rooted at** *k* if the data of the minimum key and the data of the maximum key are equal. If *k* does not exist, the method returns false. Do not use other data structures or call any method in the class BST.

Question 3 [35 points]

1. Write the method `public static <T> int height(BT<T> bt)`, **user of the ADT BT**, which returns the height of *bt*.
2. Indicate the **preorder**, **inorder** and **postorder** traversals of the tree shown in **Figure 1**. **Write only the number on the answer sheet, for example, Preorder: 7, Inorder: 1, Postorder: 12.**

1.	A C M S Q D E I K J	2.	A C M S D Q E I J K	3.	A M C S D Q E I K J
4.	A C M S D Q I E K J	5.	[✓ A C M S D Q E I K J]	6.	M C A S D Q E I K J
7.	M C Q S D A I K E J	8.	M C D S Q A K I E J	9.	M C D S Q A I K E J Pre in
10.	C S R Q X J I K Z M	11.	M D Q S C K I J E A	12.	M D Q S C K I E J A

Pre: M C S D S Q A

in2. Pre: A C M S D Q E I K J   M D Q S C K I J E A

L-R-PN

Figure 1: Binary Tree.

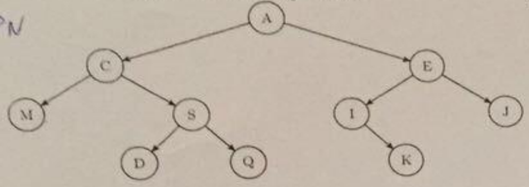
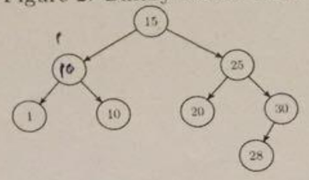


Figure 2: Binary Search Tree.



3. Given the initial BST shown in **Figure 2**, draw the resulting BST after each of the following sequences of operations. **For each sequence, you should draw one final tree result. Each sequence should be applied on the original tree.**

   (a) insert(12); insert(23); findKey(30); insert(30);

   (b) insert(0); insert(29); removeKey(7); removeKey(15);

   (c) removeKey(5); removeKey(10); removeKey(1); removeKey(15);

ADT Stack Specification

- push (T e): **requires**: Stack S is not full. **input**: T e. **results**: Element e is added to the stack as its most recently added elements. **output**: none.
- pop (T 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**: T 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.

ADT Binary Tree Specification

- boolean **find** (Relative rel): **Requires**: BT is not empty. **Results**: the current node of BT is determined by Relative and the current node prior to the operation as follows (always return true unless indicated so): (1) rel = Root: current = root (2) rel = Parent: if the current node has a parent then parent is the current node; otherwise returns false (3) rel = LeftChild: if the current node has a leftchild then it will be the current node; otherwise returns false (4) rel = RightChild: same as above but for rightchild.