CSC 212 - Spring 2017

College of Computer and Information Sciences, King Saud University
Exam Duration: 3 Hours

13/05/2017

(1) All answers must be written on the answer sheet. (2) Calculators are not allowed.

Question 1 [16 points]

1. Consider the following code:

```
1 for (int i = 0; i < n; i++) {
2         System.out.println("first");
3         for (int j = 0; j <i; j++)
4                 System.out.println("second"); }
5 System.out.println("good bye");
```

Choose the correct answer:

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

2. Consider the following code:

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

n-1 -2 +1

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
3	(a) $(n+1)(n-3)$	(b) $n(n-2)$	(c) $(n+1)(n-2)$	(d) $n^2(n+1)$	(e) $n(n+1)$
4	(a) $(n+1)(n-2)$	(b) $n(n-2)$	(c) $(n+1)(n-1)$	(d) $n^2(n+1)$	(e) $n(n+1)$
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) $O(1)$

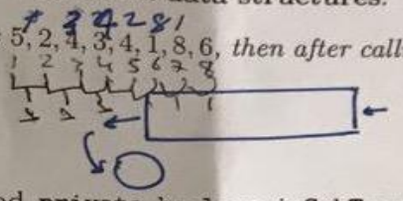
Question 2 [16 points]

1. Write a method `static Stack <T> removeFirst(Stack<T> s, T e)`, user of Stack ADT. It accepts a stack s and an element e. It creates and returns a new stack with all elements except the first instance of e from the top if exists. The stack s becomes empty at the end. Use the method equals to test for equality.

   Example 2.1. *Suppose s contains A, O, M, K, O, A (from top to bottom). When calling* removeFirst (s, O), *it will return A, M, K, O, A and when calling* removeFirst (s, B), *it will return A, O, M, K, O, A.*

2. As a user of ADT Queue, write the method `public static void bubble(Queue<Integer> q)`, which compares consecutive items and exchanges those that are out of order (assume the order must be decreasing). **Do not use any extra data structures.**

   Example 2.2. *If q contains* 5, 2, 4, 3, 4, 1, 8, 6, *then after calling the method* bubble, *q should become* 5, 4, 3, 4, 2, 8, 6, 1.



Question 3 [16 points]

1. Write the **recursive** method `private boolean inSubTree(BTNode<T> t, T e)`, member of the class *BT* which returns true if the data *e* exists in the subtree *t*, false otherwise.

2. Write the method `public boolean insertUnique(T e)` (member of *LinkedList*), that inserts *e* if it is not already in the list. If *e* does not exist, the method behaves in the same way as *insert*. The method returns true if *e* is inserted, false otherwise. **Do not call any methods of the class LinkedList.**

   Example 3.1. *If the list l contains A, B, C, D and* current *is on C, then calling* l.insertUnique("A") *does not change the list. Calling* l.insertUnique("E") *results in A, B, C, E, D with* current *on E.*

Question 4 [12 points]

1. Consider the following heap represented as an array: **3, 5, 6, 6, 8, 9**. Choose the correct answer for every operation (all operations are done on the above heap).

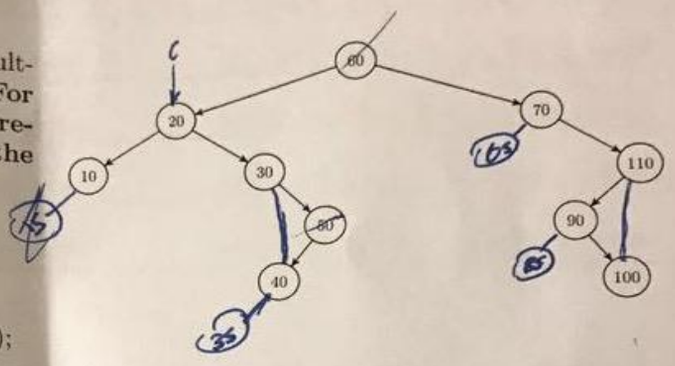
1. Heap after inserting 3:				
(a) 3,5,3,6,8,6,9	(b) 3,5,6,6,8,9,3	(c) 3,3,5,6,8,9,6	(d) 3,5,3,6,8,9,6	(e) 3,5,3,9,8,6,6
2. Heap after inserting 7:				
(a) 7,5,6,6,8,9,3	(b) 3,5,7,6,8,9,6	(c) 3,5,6,6,8,9,7	(d) 3,6,5,6,8,9,7	(e) 3,5,7,6,8,9,6
3. Heap after inserting 1:				
(a) 6,5,3,6,8,9,1	(b) 3,5,1,6,8,9,6	(c) 1,3,5,6,8,9,6	(d) 3,5,6,6,8,9,1	(e) 1,5,3,6,8,9,6
4. Heap after inserting 5:				
(a) 5,5,3,6,8,9,6	(b) 3,5,5,6,8,9,6	(c) 3,5,6,6,8,9,5	(d) 3,5,5,6,8,6,9	(e) 3,5,5,6,8,6,9
5. Heap after inserting 2:				
(a) 2,5,3,6,8,9,6	(b) 3,5,2,6,8,9,6	(c) 3,5,6,6,8,9,2	(d) 2,3,5,6,8,9,6	(e) 2,5,6,6,8,9,3

2. Consider the following heap represented as an array: **9, 8, 7, 8, 6, 5, 4**. Choose the correct answer for every operation (all operations are done on the above heap).

1. Heap after deleting one key:				
(a) 8,7,8,4,6,5	(b) 4,8,7,8,6,5	(c) 9,8,7,8,6,5	(d) 8,4,7,8,6,5	(e) 8,8,7,4,6,5
2. Heap after deleting two keys:				
(a) 8,6,7,4,5	(b) 8,7,4,6,5	(c) 8,8,7,4,6	(d) 8,6,7,5,4	(e) 7,8,5,4,6
3. Heap after deleting three keys:				
(a) 6,7,4,5	(b) 7,6,5,4	(c) 8,6,7,4	(d) 5,6,7,4	(e) 7,5,6,4

Question 5 [12 points]

Given the initial BST to the right, draw the resulting BST for each of the following sequences. **For each part, you should have one final tree result. Each part should be applied on the original tree.**



1. insert(35); insert(65); insert(85); insert(15);
2. insert(15); removeKey(50); findKey(20); insert(19);
3. removeKey(60); removeKey(70); removeKey(20); removeKey(10);
4. removeKey(70); removeKey(60); removeKey(110); removeKey(90);

Question 6 [12 points]

1. Trace the evaluation of the following postx expression using a stack. Draw the stack after every push or pop operation (you have to draw the stack 13 times in total):
   `9 8 - 7 8 - / 9 2 + 7 + *`
2. Trace the evaluation of the following expression (draw the stacks after every push operation):
   $7 + 8 - 2 \leq 2 * 5 + 3/2$

Question 7 [12 points]

Use the hash function $H(key) = key\%11$ to store the sequence of keys **24,27,19,13,35,16,30,38,57,8** in the hash table. Use the following collision resolution strategies:

1. Linear rehashing (c=1), indicate the number of probes.
2. External chaining.
3. Coalesced chaining with cellar size 3 (do not change the hash function).

Question 8 [4 points]

Choose the most appropriate data structure:

1) Check that the parentheses in an expression are balanced				
(a) Hash table	(b) array list	(c) linked priority queue	(d) heap	(e) linked stack
2) Evaluate a postfix expression				
(a) linked list	(b) linked queue	(c) binary tree	(d) AVL tree	(e) array stack
3) Implement a phone directory				
(a) AVL tree	(b) array list	(c) heap	(d) Hash table	(e) linked list
4) Undo/redo in a word processor				
(a) linked stack	(b) array list	(c) linked priority queue	(d) heap	(e) linked list
5) Evaluate an infix expression				
(a) linked list	(b) linked queue	(c) binary tree	(d) linked stack	(e) array queue
6) Manage clients' orders at an online store				
(a) linked stack	(b) heap	(c) array queue	(d) priority queue	(e) Hash table
7) Check that HTML tags are balanced				
(a) linked list	(b) linked queue	(c) binary tree	(d) linked stack	(e) array queue
8) Manage patients in an emergency service				
(a) linked stack	(b) heap	(c) array queue	(d) priority queue	(e) Hash table

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.