METU
Department of Computer Engineering
CENG 213 Data Structures
Midterm, November 30, 2016
120 min.

ID: SAMPLE solution

Name: _____

Q1 (12 pts)	
Q2 (12 pts)	
Q3 (20 pts)	
Q4 (16 pts)	
Q5 (25 pts)	
Q6 (15 pts)	
TOTAL	

READ FIRST BEFORE STARTING THE EXAM:

- This is a **closed book, closed notes** exam. The use of any reference material is strictly forbidden.
- No attempts of cheating will be tolerated.
- **Cell phones** and **other electronic devices** must remain **off** during the exam.
- Do not ask **unnecessary questions**. If you think that there is an error, clearly indicate that on the first page of the exam.
- Answer the questions briefly and clearly **only** in the boxes or underlined space provided.
- This exam consists of **8 pages** including this page. Check that you have them all!
- **GOOD LUCK!**

Q1. (12 Pts)
Describe the worst case running time of the following functions in Big-O notation in terms of the variable n.
(3 pts. each)

a)

```
void sunny (int n) {
   j = 0;
   while (j < n) {
      for (int i = 0; i < n; ++i) {
         cout << "i = " << i;
         for (int k = 0; k < i; ++k)
            cout << "k = " << k;
      }
      j = j + 1;
   }
}
```

$O(n^3)$

b)

```
void smiley (int n) {
   for (int i = 0; i < n * n; ++i) {
      for (int k = 0; k < i; ++k)
         cout << "k = " << k;
      for (int j = n; j > 0; j--)
         cout << "j = " << j;
   }
}
```

$O(n^4)$

c)

```
void funny (int n, int x) {
   for (int i = 0; i < n; ++i) {
      j = n;
      while (j > 0) {
         cout << "j = " << j;
         j = j - 2;
      }
}
```

$O(n^2)$

d)

```
int silly(int n, int m) {
   if (n < 1) return m;
   else if (n < 10)
      return silly(n / 2, m);
   else
      return silly(n - 2, m);
}
```

$O(n)$

Q2. (12 Pts)
Assume that growth rate of an algorithm is given by a function f(N). For each of the functions f(N) given below, indicate the complexity of the algorithm in Big-O notation. Unless otherwise specified, all logs are base 2. (2 pts each)

a) $f(N) = N^3 (4 \log N - \log N) + (N^2)^2$ $O(N^4)$

b) $f(N) = \log_8 (2^{4N})$ $O(N)$

c) $f(N) = 500\ N \log N + N^2 \log N$ $O(N^2 \log N)$

d) $f(N) = \log N^4 + \log^2 N$ $O(\log^2 N)$

e) $f(N) = N^3\ N^2 + (N + 2N)^2$ $O(N^5)$

f) $f(N) = N^2 \log N^2 + 2N \log^2 N$ $O(N^2 \log N)$

Q3. (20 Pts.)
Give the time complexity in Big-O notation for the worst case running time for each operation in terms of N. Assume you can implement the operation as a member function of the class – with access to the underlying data structure, including knowing the number of values currently stored (N). (2 pts. each)

a) Given a FIFO queue of integers implemented as a linked list, find and remove all of the values greater than 10, leaving the rest of the queue in its original order. $O(N)$

b) Given a binary search tree, print all of the odd values from largest to smallest. $O(N)$

c) Given a binary search tree, find which value is the minimum and delete it. $O(\log N)$

d) Given a ~~FIFO~~ LIFO stack ~~queue~~ implemented as a linked list currently containing N values, enqueue N more values so that when you are finished the queue will contain 2N values – Give the total time for enqueueing N more values. $O(N)$

e) Given a ~~FIFO queue~~ LIFO stack implemented as a linked list, find and remove all of the values greater than 10, leaving the rest of the ~~queue~~ stack in its original order. $O(N)$

f) Enqueue a value onto a queue containing N values implemented as a circular array (as described in class). Assume the array is size N+5. $O(1)$

g) Given a linked list of integers where all elements are equal, cost of sorting it using insertion sort. $O(N)$

h) Given a linked list of integers where all elements are already sorted, cost of sorting it using selection sort. $O(N^2)$

i) Given a linked list of integers where all elements are already sorted, find the middle element. $O(N)$

j) Pushing a value onto a stack implemented as an array. Assume the array is of size 2N. $O(1)$

Q4. (16 Pts)

a) (4 pts) What is the minimum and maximum number of leaf nodes in a complete tree of height 6? (Hint: the height of a tree consisting of a single node is 0) Give an exact number (with no exponents) for both of your answers – not a formula.

Minimum: $2^6 = 64$

Maximum: $2^{6+1} - 1 = 2^7 - 1 = 128 - 1 = 127$

b) (2 pts) What is the maximum number of leaf nodes in a full tree of height 5? Give an exact number (with no exponents) for your answer – not a formula.

Maximum: $2^5 = 32$

c) (1 pt) What is the depth of node E in the tree shown below? 2

d) (1 pt) What is the height of node C in the tree shown below? 2

e) (1 pt) Is the tree shown below height balanced binary tree? Yes

f) (2 pts) Give a PREORDER traversal of the tree shown below.

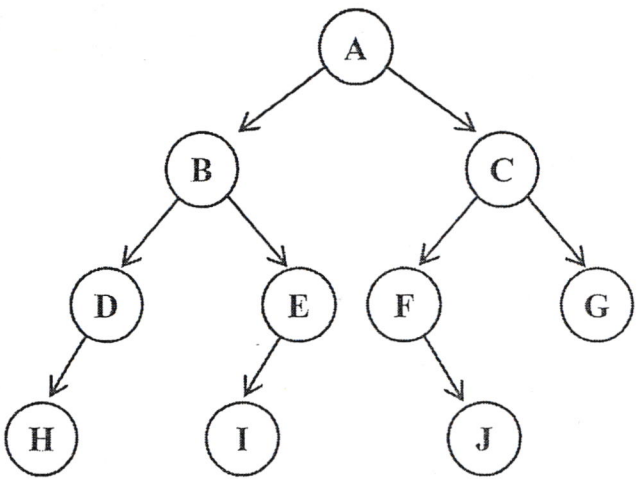
A B D H E I C F J G

g) (2 pts) Give an INORDER traversal of the tree shown below.

H D B I E A F J C G

h) (2 pts) Give a POSTORDER traversal of the tree shown below.

H D I E B J F G C A



Q6. (25 Pts)

Given the interface below for a singly linked list class

```
#include "ListException.h"

template <class ListItemType>
class Node {
 public:
  Node(const ListItemType& e=ListItemType(), Node *n=NULL);
  ListItemType element;
  Node *next;
};

template <class ListItemType>
class List {
 private:
  Node<ListItemType> *dummyHead;
 public:
  List();
  ~List();
  List(const List& rhs);
  List& operator=(const List& rhs);
  Node<ListItemType>* zeroth() const;
  Node<ListItemType>* first() const;
  bool isEmpty() const;
  void insert(const ListItemType& data, Node<ListItemType>* p);
  Node<ListItemType>* find(const ListItemType& data) const;
  Node<ListItemType>* findPrevious(const ListItemType& data);
  void remove(const ListItemType& data);
  void makeEmpty();
  ...
};
```

a) Write C++ code for **`operator[](i)`** that will return a reference to the `i`-th element of the linked list if exists, where `i`≥1; throw a `ListException` otherwise. Do not declare any other local variables.

```
template <class T>
T& List<T>::operator[](int i) const {
  Node<T> *p=dummyHead;
  if (i<1) throw ListException("No such element on operator[]");
  while(i>0 && p) {
    i--;
    p=p->next;
  }
  if (p)
    return p->element;
  else
    throw ListException("No such element on operator[]");
}
```

b) Write C++ code for `operator*(rhs)` that will return the resulting linked list `r=lhs*rhs` such that `r` will have all the common elements of `lhs` and `rhs`, assuming no duplicates exist. The resulting list will keep the order of `lhs`. For example; `[a,b,c,d]*[c,b,e]=[b,c]` and `[a,b]-[]=[]`. Do not declare any other local variables.

```
List<T> List<T>::operator*(const List<T> &rhs) const {
  List<T> result;
  Node<T> *p=first();
  Node<T> *r=result.zeroth();
  while(p) {
    if (rhs.find(p->element)!=NULL) {
      result.insert(p->element,r);
      r=r->next;
    }
    p=p->next;
  }
  return result;
}
```

Q6. (15 Pts)

Consider the following public `Stack` interface. Note that the given interface is slightly different from the interface we discussed in class.

```
#include "StackException.h"
template <class StackItemType>
class Stack {
public:
   Stack();
   Stack(const Stack& aStack);
   ~Stack();
   bool isEmpty() const;
   void push(StackItemType newItem);
   StackItemType pop() throw(StackException);
   StackItemType getTop() const throw(StackException);
...
}
```

Write a C++ function `findAndRemoveLargest(Stack)` that will determine which value in a stack (of integers) is the largest and remove and return that value while otherwise leaving the remainder of the stack in its original order. Thus if the stack1 contains values: (bottom 4 6 8 3 2 5 top) then a call to `findAndRemoveLargest`(stack1) would return the value 8 and leave stack1 as follows: [bottom: 4 6 3 2 5 :top]. In your solution, you may only declare extra stacks or scalar variables (e.g. ints, doubles – not collections like arrays or linked lists) if needed.

You can assume that any stack you use will never become full and that the provided stack will not contain duplicate values. Rather than throwing an exception, if the provided stack is empty `findAndRemoveLargest(Stack)` will return the value -409.

```
int findAndRemoveLargest(Stack<int> & S) {
    // Declare another Stack
(1)   Stack<int> S1;
    // Declare variables to store the top item and current maximum
(1)   int top, max, temp;
    // Process all items in Stack S
(2)   if (S.isEmpty()) return -409;
      else {
(1)     max = S.getTop();   initialize max
(1)     while (!S.isEmpty()) {   check S
(1)       temp = S.pop();
(2)       if (temp > max) max = temp;
(1)       S1.push(temp);
        }
(1)     while (!S1.isEmpty()) {
(1)       temp = S1.pop();
(2)       if (temp != max) S.push(temp);
        }
}
(1)     return max;
```

Finding max:
S must be emptied
S1 must be filled
update max if necessary

Empty S1 & fill S without max