Simon Fraser University
Computing Science 225
Midterm Examination
Spring 2008

Time: **60 minutes**

Last/Family Name: _____

(Please, PRINT)

First Name: _____

(Please, PRINT)

Student Number: _____

Signature: _____

Instructor: Alireza Khodabakhshi

This examination has 8 pages inclusive.

Verify that you have a complete paper.

- ✓ No books, cheat sheets, calculators, or other materials may be used.
- ✓ Read each question carefully before answering it.
- ✓ Answer all the questions on this paper.
- ✓ List any assumptions you make when answering a question.
- ✓ The marks for each question are given in the table in front. Use it to manage your time:

Good luck!

Q #	Marks
1	/ 1
2	/ 4
3	/ 4
4	/ 4
5	/ 4
6	/ 3
Total	**/ 20**

1. [1 mark] Give one line description of the following terms.

a) Abstraction: **separation of the purpose of the module from its implementation details.**

b) Object in a program: **a collection of data and methods wrapped in a variable.**

c) Stack ADT: **is an ADT that items can be inserted or removed only from one end.**

d) Loosely coupled objects: **objects that have low degree of dependency to each other.**

2. a) [2 marks] Approximate the following running time functions by providing their closest upper bound expressed using the **Big O notation**. Note that n represents the size of the input.

i. $t(n) = n + 6\sqrt{n}$ **O(n)**

ii. $t(n) = (\log n + n)*(n - \text{long } n)$ **$O(n^2)$**

iii. $t(n) = n^2 \log n + n^3 + 7n$ **$O(n^3)$**

iv. $t(n) = n \log n + n^{1.1} + 3^{100}$ **$O(n^{1.1})$**

2. b) [2 mark] Analyze the time complexity of the following algorithms (give a Big O notation for each code).

I)

```
x = 0;
for( i = 0; i < n; i++ ) => O(n)
  for( j = 0; j < n * n; j++ ) => O(n^2)
    x++;
```

$O(n^3)$

II)

```
x = 0;
for( i = 0; i < n; i++ ) => O(n)
  for( j = n; j > 1; j=j/2) => O(log n)
    x++;
```

O(n log n)

3. a) [2] The following two methods recursively compute the power 2 of a positive integer n. Argue which of them is more efficient by analyzing the number of operations of each method.

```
int pow2(int n){
   if (n==0)
      return 1;
   else
      return 2*pow2(n-1);
 }
```

```
int pow2(int n){
   if (n==0)
      return 1;
   else
      return pow2(n-1)+pow2(n-1);
}
```

Left: **T(n)=T(n)+1, and T(0) = 1 => T(n) is O(n)**
Right: T(n) = T(n-1) + T(n-1) + 1 = 2T(n-1) + 1, and T(0) = 1 => T(n) is O(2^n)
Therefore, the left method is more efficient.

b) [1] Consider the following recursive method. Draw the execution tree for n=4 and k=2.

```
int choose(int n, int k){
     if (k==0 or k == n)
          return 1;
     if (k > n)
          return 0;
     return choose(n-1, k-1)+choose(n-1, k);
}
```

Page 142. text book.

c) [1] using an induction prove that the T(n) is O(n log n) where
T(n) = 2T(n/2) + n and T(2) = 1;

We need to prove T(n) < c.n.log n for some constant c.
The base of induction is true because: T(2) = 1 < 2c, for c > 1/2.
Induction hypothesis: assume T(m) < c.m.log m for all m < n and some constant c.
We will prove that T(n) < c.n.log n.
T(n) = 2T(n/2) + n < 2(c.n/2.log(n/2)) + n = n(c.log(n/2) + 1) < n.c.log(n)

4. [4] Write a method that gets the reference to the first element in a linked list (i.e head), reverses it and returns the reference to the head of the reversed list. The method should not use any temporary array or. The method should run in O(n) of space and time. Below is the prototype of the method.

```
Node reverseList(Node head);
```

Remarks: the class Node has three methods: Node getNext(), void setNext(Node n), Object getItem().

```
void reverseList(Node head){
     Node prev, curr, next;
     prev = null; curr=head; next=curr.getNext();
     while(curr != null){
          curr.setNext(prev);
          prev=curr;
          curr=next;
          next=next.getNext();
     }
     Return prev;  //the head of the reversed list is prev.
}
```

5.[4] **Using the ADT Stack methods**, write a method that takes two stacks of characters Stack1 and Stack2 as arguments, and "compresses" Stack1 by deleting consecutive copies of the same character from it. For example, the stack *(A,B,B,C,C,A,A,A,C,A)* should be compressed to *(A,B,C,A,C,A)*. Thus the same character can appear more than once in the compressed stack, only not successively. **Requirement**: your method cannot use any temporary array, list, or stack. Recall that the ADT Stack provides the following methods:

```
void push(char c); char pop();char peek();boolean isEmpty();
void popAll();
```

```
void compress(Stack s1, Stack s2){
     s2.popAll(); //make s2 empty.
     if (s1.isEmpty())
          return;
     s2.push(s1.pop()); //push the first character of s1 in s2.
     while(!s1.isEmpty()){
          char c=s1.pop();
          if(c != s2.peek())
               s2.push(c);
     }
     while(s2.isEmpty())
          s1.push(s2.pop());
}
```

6.[3] Write a method that gets an array of integers and returns true if the array has repeated elements and false otherwise.

Requirement: the method must run in O(n log n) where n is the size of the array.

```
boolean hasRepetition(int arr[]){
    sore(arr, "merge"); //O(n log n)
    for(int i=0; i<arr.length-1; i++){
          if(arr[i]==arr[i+1])
               return true;
    }
    return false;
}
```