A

CS3334 Data Structure

Mid-term Quiz (Sem. A, 16-17)

(50 min)

Student ID:

Tutorial Group: Mon (11:00)
Tue (9:00)
Tue (15:00)

Answer all questions in this paper

/80

Section A: Multiple Choices (40 marks)

(4 marks for each correct answer, write down your answer on the answer box on **page 4**)

1. Which of the following statements is/are true if $f(n)=O(g(n))$?

   i. $g(n) \geq f(n)$ for all n.
   ii. there exist a positive constant n_0 such that $g(n) \geq f(n)$ for all $n \geq n_0$.
   iii. there exist positive constants n_0 and c such that $cg(n) \geq f(n)$ for all $n \geq n_0$.
   iv. there may exist positive constants n and c such that $cg(n) \leq f(n)$.

   a. i and ii
   b. ii and iii
   c. iii and iv
   d. ii, iii and iv

2. Which of the following statements is/are true if $f(n)=\Theta(g(n))$?

   i. $c_1g(n) \leq f(n) \leq c_2g(n)$ for all n, *where* c_1 *and* c_2 *are two positive constants*
   ii. $c_1g(n) \leq f(n) \leq c_2g(n)$ for all $n \geq n_0$, *where* n_0, c_1 *and* c_2 *are positive constants*
   iii. $f(n)=O(g(n))$
   iv. $f(n)=\Omega(g(n))$

   a. i and ii
   b. ii and iii
   c. iii and iv
   d. ii, iii and iv

3. Suppose $f(n) = 2n^3+4n^2+1$, and $f(n)=O(g(n))$, $g(n)$ could be:

   a. n^3
   b. $4n^2$
   c. n^2
   d. 1

4. Suppose $f(n)=\Theta(g(n))$ and $g(n)=n^2$, $f(n)$ could be:

   a. Log(n)
   b. n
   c. n^2
   d. n^3

5. The O-notation of function $f(n)=2n^2-3n^3+2n+10$ should be:

    a. $O(n^2)$
    b. $O(2n^2-3n^{3)}$
    c. $O(n^3)$
    d. O(n)

6. Which of the following is NOT the Ω-notation of function $f(n)= 3n^3+2n+10$?

    a. $\Omega(n^3)$
    b. $\Omega(n^2)$
    c. $\Omega(n)$
    d. $\Omega(2^n)$

7. Which of the following function is not a valid operation of stack?

    a. pop(): pop an element from stack
    b. push(): push an element to stack
    c. length(): return number of element currently stored in the stack
    d. getAt(int i): return the i^{th} element of stack

8. Which of the functions have the same order of growth rate of $g(n)=2^n+n^2-10n$?

    i. $f(n)=log_2(n)+2n-10$
    ii. $f(n)=2^n+n^2$
    iii. $f(n)=2^n$
    iv. $f(n)=n^2$

    a. i
    b. iii
    c. ii and iii
    d. iii and iv

9. Which of the following statements about sorted arrays is TRUE?

    a. It has better performance in element insertion in compared with ordinary array
    b. It has better performance in element searching in compared with ordinary array.
    c. Elements stored in the array must be in an ascending order
    d. Elements stored in the array must be in a descending order

10. Consider the following function that search an element in an array

```
int find(int a[], int n, int key){
     for (int i=0;i<n;i++){
          if (a[i]==key)
               return i;
     }
     return -1;
}
```

Assume:

- Variable declaration and function return take 0 unit of time
- For operations (=, +, <, ++, [],==), each takes 1 unit of time.
- Ignore the cost of parameters passing and initialization

What is the worst-case computation cost?

a. 4n+2
b. 4n+1
c. 3n+2
d. 3n+1

Answer

Section B: Long Questions (40 marks)

1. Consider the following class, Node, which is designed to implement a singly-linked list. The header file "Node.h" and implementation "Node.cpp" are listed below. (20 marks)

```
//Node.h
#include <iostream>
class Node
{
private:
	int data;
	Node* next;
	/* (1) */

public:
	Node();
	Node(int data, Node* node);
	//getter and setter
	int getData();
	void setData(int data);
	Node * getNext();
	void setNext(Node* next);
	~Node();
	/* (2) */
};
```

```
//Node.cpp
#include "Node.h"

Node::Node()
{
	data = -1;
	next = NULL;
}
Node::Node(int data, Node* node){
	this->data = data;
	this->next = node;
}
int Node::getData(){
	return data;
}
void Node::setData(int data){
	this->data = data;
}
Node * Node::getNext(){
	return next;
}
void Node::setNext(Node* next){
	this->next = next;
}
Node::~Node()
{
}
/* (3) */
```

Modify the Node.h and Node.cpp so that the class Node can be used to implement a **doubly-linked list**. Your code must be in good programming style, i.e. efficient and readable.

a. Provide the code to be **inserted** after placeholder (1) and (2) of "Node.h", write your code in the space below. (6 marks)

```
/* (1) */

Node* prev;  //reference of the previous node
                // (2 marks)


/* (2) */
// getter and setter (4 marks)
Node* getPrev();
void setPrev(Node *n);
```

b. Provide the code to be **inserted** after placeholder (3) of "Node.cpp", write your code in the space below. (4 marks)

```
/* (3) */

// Implementation 4 marks
Node* Node::getPrev(){
      return prev;
}
void Node::setPrev(Node *n){
      prev=n;
}
```

c. Rewrite the constructors of Node so that they work well with your code in part (a) and (b). All pointers should be properly set within the constructors. Write down the revised constructors in the space below: (10 marks)

```
//code to be placed in Node.h

Node(int data, Node* prev, Node* next);
//(2 marks)




//code to be placed in Node.cpp

Node::Node()
{
      data = -1;
      next = NULL;
      prev = NULL;           (2 marks)
}
Node::Node(int data, Node* prev, Node* next){
      this->data = data;
      this->next = next;
      this->prev = prev;          (2 marks)
      prev->setNext(this);        (2 marks)
      next->setPrev(this);        (2 marks)

      -1 marks for accessing private attribute
      e.g. prev->next=this;

}
```

2. In UNO game, there are three types of card: normal cards with number 0-9, action cards (skip, reverse, draw 2) and wild cards (wild / wild draw 4). Except wild cards, cards also in one of four colors (red, green, blue and yellow). You may assume the color of wild cards is black. Consider a modified UNO game with the following rules:

   Game setup:
   - Each player collects 5 cards from the deck and builds a list.
   - Card in the list can be re-arranged in any order that fits the player's strategy.
   - Cards' order cannot be changed after the game is started.
   - To start the game, a card is drawn from the deck randomly and flipped over as the last disposed card.

   Gameplay:
   - Player takes turn to play a card from his/her list.
   - Exception action/wild cards, player can only select the first card from his/her list.
   - Except wild cards, player can only play card with the same color or number or action as the last disposed card.
   - If no card can be played, the player must draw a card from the deck.
   - Player can insert the newly drawn card into any position of the list. Once inserted, the card cannot be rearranged.

   Winning condition:
   - The first player who disposed all his/her card will win the game.

   A programmer has created a class "Card" to model a UNO card. The class has two attributes, color and number, both in integer. Card with number greater than 9 indicates it is either an action card or wild card. The mapping of color, action cards and wild cards to integer are defined in the header file below:

```
//Card.h
//#include <iostream>
//Color of Card
#define RED       0
#define BLUE      1
#define GREEN     2
#define YELLOW    3
#define BLACK     4 //color of wild and wild draw 4 cards

//Card Number for action cards
#define REVERSE   10
#define SKIP      11
#define DRAW2     12
#define WILD      13
#define WILDDRAW4 14
class Card
{
```

```
private:
      int number; // 0-9 normal card,
                  // 10-14 action/wild card
      int color;  // 0-3,
      /* (1) */

public:
      Card(int number,int color);

      //getter and setter
      int getNumber();
      int getColor();

      ~Card();
      /* (2) */
};
```

a. Add codes to placeholder (1) and (2) so that the class Card can be used to build a singly-linked list of Card. (6 marks)

```
/* (1) */

      Card* next;

/* (2) */

      Card * getNext();
      void setNext(Card* next);
```

b. The following shows the header file and partial implementation of class CardList, which is a data structure similar to stack and designed to keep the cards of a player. The class has two main functions, "insert" and "play".
Function "insert" is designed for inserting new card, c, into the list.
Function "play" is designed for finding the card that can be played in current round with last disposed card "lastCard". Cards is stored in a singly-linked list within CardList.

```
//CardList.h
#include "Card.h"
class CardList{
private:
      Card * head;
public:
      CardList();
      void insert(Card *c);
      Card* play(Card* lastCard);
};
```

```
//CardList.cpp
#include "CardList.h"

CardList::CardList(){
      head=new Card(-1,-1);   //dummy head node
}
void CardList:: insert(Card* n){




}
```

i. Provide the implementation of function "insert" which insert the new card, n, at the heading position of the list. Do not modify other code. (4 marks)

```
void CardList:: insert(Card* n){

      n->setNext(head->getNext());   (2 marks)
      head->setNext(n);              (2 marks)


      -1 mark if delete n









}
```

ii. Suppose a player want to group cards with same color, with the newest card placed at the beginning of the set. If the color of the newly drawn card cannot be found, append the card at the end of the list. Provide another version of "insert" that implements this approach. (10 marks)

```
void CardList:: insert(Card* n){

      Card *c=head->getNext();
      Card *p=head;
      while (c!=NULL){
            if (c->getColor()==n->getColor()){
                  n->setNext(c);
                  p->setNext(n);
                  return;
            }
            p=c;
            c=c->getNext();
      }

      p->setNext(n);



      //Correct insert when color found (4 marks)
      //    if new card is appended to the end instead of
      //    insert to the heading position ( -2 marks)


      //Traversing the list correctly (2 marks)
      //Insert Card in case color not found (2 marks)
      //Insert card when the list is empty (2 marks)


}
```