UCLA Computer Science Department TA: Kung-Hua Chang

Student Name and ID _____

CS 32, WINTER 2015, PRACTICE FINAL EXAM.

Problem #	Maximal Possible Points	Received
1	6	
2	9	
3	4	
4	12	
5	4	
6	12	
7	3	
8	6	
9	4	
10	3	
11	3	
12	14	
13	20	
Total	100	

***Every problem below has one or more answers. Please select all the correct ones.**

Problem #1: For the program below, which one or more of the following are correct?

(A) The output is: 3 5 -5 -3

(B) The output is: 3 5 -3 -5

(C) The output is: 3 5

(D) The output is: 2 4 -4 -2

(E) The output is: 2 4 -2 -4

(F) The output is: 2 4

(G) The output is: 2 4 -3 -5

(H) The output is: 2 4 -5 -3

(I) There is a memory leak in this program.

(J) There is no memory leak in this program.

(F)(I)

```
#include <iostream>
#include <vector>
using namespace std;
class myClass
{
public:
	myClass(int v): value( v + 1 )
	{
		cout << v << " ";
	}
	~myClass()
	{
		cout << -value << " ";
	}
private:
	int value;
};
int main()
{
    vector<myClass*> vec;
    vec.push_back( new myClass(2) );
    vec.push_back( new myClass(4) );
}
```

Problem #2: A Fibonacci number is defined as one of a sequence of numbers such that every number is the sum of the previous two numbers except for the first two numbers being 1. Thus, a Fibonacci number is one of: 1 1 2 3 5 8 13 … etc. Below are two implementations to calculate Fibonacci numbers. Which of the following descriptions are **TRUE**?

(A) Fib(5) = 8

(B) The Big-O complexity of Fib() is O(N)

(C) The Big-O complexity of Fib2() is O(N)

(D) Fib() runs faster than Fib2() when N > 10

(E) Fib2() runs faster than Fib() when N > 10

(A)(C)(E)

```
#include <iostream>
using namespace std;

int Fib(int N)
{
	if( N == 0 || N == 1)
		return 1;
	return Fib(N-1) + Fib(N-2);
}

int Fib2(int N)
{
	int *arr = new int[N+1];
	arr[0] = 1;
	arr[1] = 1;
	for(int i = 2; i <= N; i++)
		arr[i] = arr[i-1] + arr[i-2];
	int value = arr[N];
	delete [] arr;
	return value;
}
```

Problem #3: Suppose an array holds a sequence of N unique numbers that are sorted in decreasing order. If we move the last P numbers (P < N) to the beginning of this sequence of numbers, then what is the best possible Big-O time complexity you can achieve to search for a specific given number?
For example, suppose we have the following sequence of numbers:
{ 10 , 9 , 8 , 7 , 6 , 5 , 4 , 3 , 2 , 1 }
Suppose P were 4 and someone moved the last P numbers to the beginning:
{ 4 , 3 , 2 , 1 , 10 , 9 , 8 , 7 , 6 , 5 }
If we don't know P and want to search for a number (say, 7), what is the most efficient algorithm then? (B)

(A) O(N)
(B) O(log N)
(C) O(N log N)
(D) O(1)

Problem #4: Which of the following descriptions are **TRUE** about Linked List? (C)(D)(F)(G)

(A) The Big-O complexity to insert a node into a singly linked list is always O(1).
(B) The Big-O complexity to delete a node from a doubly linked list is always O(1).
(C) If there are both head and tail pointers in doubly linked list, then the insertion and deletion operation is always O(1) at the beginning and end of the list.
(D) If a stack is implemented using a singly linked list, then it is more efficient to use the head pointer as the top of the stack.
(E) If a queue is implemented using a singly linked list (with tail pointer), then it is more efficient to use the tail pointer as the front of the queue and the head pointer as the rear of the queue than vice versa.
(F) Given only a pointer pointing to one of the nodes to be deleted, it is more efficient to delete that node in a doubly linked list than in a singly linked list.
(G) Merging two sorted linked lists, each with N nodes, is O(N).

Problem #5: Which of the following are **TRUE** about Binary Search Trees (BST) with no special attention paid to balancing, and Balanced BSTs? (D)

(A) Search operations are always O(log N) in Binary Search Tree.
(B) The search operation in Binary Search Tree is O(log N) in the worst case.
(C) The insertion operation in Binary Search Tree is O(log N) in the worst case.
(D) The output from in-order traversal on Binary Search Tree is always sorted.
(E) After inserting N keys and deleting a random number of P keys (P < N) in a Binary Search Tree, the resulting Binary Search Tree is always perfectly balanced.
(F) Balanced Search Trees never ensure that all insertion, deletion, and search operations are O(log N).

Problem #6: Given the following insert() operations into an empty Binary Search Tree with no attention paid to balancing. Insert(2), Insert(0), Insert(1), Insert(3), Insert(5), Insert(7), Insert(9). Which of the following are true? (A)(B)(D)(E)

(A) The numbers that the pre-order traversal visited are in this order: 2 0 1 3 5 7 9

(B) The numbers that the in-order traversal visited are in this order: 0 1 2 3 5 7 9

(C) The numbers that the post-order traversal visited are in this order: 1 0 9 5 7 3 2

(D) After Delete(5), the numbers that the post-order traversal visited are in this order: 1 0 9 7 3 2.

(E) After Delete(5), the numbers that the level-order traversal visited are in this order: 2 0 3 1 7 9

Problem #7: Which of the following sorting algorithms is the fastest when applying it to sort this sequence of numbers: {5, 17, 50, 55, 65, 70, 80, 90, 100}? (D)

(A) Quick Sort

(B) Merge Sort

(C) Selection Sort

(D) Insertion Sort

(E) Heap Sort

Problem #8: Which of the followings are **TRUE**? (D)(E)

(A) If there are N numbers in a max heap, finding the 2nd largest number takes no better than O(log N) time.

(B) If there are N numbers in a max heap, using level order traversal on this max heap returns N numbers that are sorted in descending order.

(C) If there are N numbers in a min heap, using level order traversal on this min heap returns N numbers that are sorted in ascending order.

(D) There exists a max heap (with more than 1 node) that is a binary search tree.

(E) If there are N numbers in the array, the Big-O complexity of Heapsort is O(NlogN).

Problem #9: Here is a list of 10 numbers that are partially sorted. Which sorting algorithm(s) could have produced the following array after 2 iterations?
Original sequence: 30,20,40,50,60,10,70,90,80,0
Sequence after 2 iterations: 20 30 40 50 60 10 70 90 80 0

(A) Selection Sort

(B) Insertion Sort // (B)

(C) Bubble Sort

Problem #10: What is the Big-O complexity of the following function isV2inV1()?

(A) O(N1) // (A)

(B) O(N1 + N2)

(C) O(N2)

(D) O(N1*N2)

```
bool isV2inV1(int v1[], int N1,int v2[], int N2)
{
	int i = 0, j = 0;
	while( i < N1 && j < N2 ) {
		if(v1[ i ] == v2[ j ]) {
			i++;
			j++;
		}
		else
			i++;
	}
	if( j == N2)
		return true;
	return false;
}
```

Problem #11: What is the Big-O complexity of the following program?

(A) O(N^2)

(B) O(N^3)

(C) O(N^4) // (C)

(D) O(N^5)

```
#include <iostream>
using namespace std;
int main() {
	int N;
	cin >> N;
	for(int i=1; i <= N ; i++)
		for(int j=1; j <= i*i ; j++)
			if(j % i == 0)
				for(int k = 0 ; k < j ; k++ )
					cout << ".";
			else
				cout << endl;
}
```

Problem #12: What is the output of the following program?

O2 B 2 ~B ~O2 O4 B O3 D 3 ~D ~O3 ~B ~O4

```
#include <iostream>
#include <string>
using namespace std;
class Other
{
    int value;
public:
    Other(int n): value(n)      { cout << "O" << value; }
    ~Other()                    { cout << "~O" << value; }
};
class Base
{
public:
    Base(int i):o(i)                { cout << " B "; value = i; }
    virtual ~Base()                 { cout << " ~B "; }
    virtual void output_value()     { cout << value ; }
private:
    int value;
    Other o;
};
class Derived : public Base
{
public:
    Derived(int i): Base(i+1),value(i),o(i)     { cout << " D "; }
    ~Derived()                                  { cout << " ~D "; }
    virtual void output_value()                 { cout << value ; }
private:
    int value;
    Other o;
};

void processAll(Base *bp)
{
    bp->output_value();
    delete bp;
}
```

```
int main()
{
      processAll( new Base(2) );          cout << " ";
      processAll( new Derived(3) );       cout << endl;
}
```

Problem #13: You have had enough of this weird rainforest and decided to go back to the civilized world. Unfortunately, since you solved the water jug problem last time (see Problem #4 in Practice Midterm II), the cannibals believe that eating you will give them the wisdom you have. Thus, you are now being chased by 300 cannibals. The only way to return to the civilized world is to go through a maze. Fortunately, you stole a map of this maze from one of the cannibals, but you found that this maze is not a common maze. There are many rocks blocking the road here and there. Although you can use a bomb in your bag to smash the rocks to get through, you only have 2 bombs left. Thus, you decided to write a program to figure out whether you are able to use 2 bombs to smash 2 rocks and get to the exit. If so, then print out the moves you need to take in order to get to the exit.

Given the following map (S is the start, Q is the exit, R is a Rock, W is a wall)

```
SWWWWWWWWW
..WWWWWWWW
W...WWWWWW
W.WRWWWWWW
..W......W
.WWWRW.WRW
.WWWRW.W.W
RWW..W.RWW
WWWWWW.WRQ
WWWWWW...W
```

Your program should output: (Directions S/E/N/W indicate South/East/North/West)

SESEES Bomb!

SEEESSSSSEEN Bomb!

E Found Exit!

```
#include <iostream>
#include <string>
using namespace std;

int direction[4][2] = { {-1,0}, {0,-1}, {1,0}, {0,1} };
string dir_str[4] = {"N","W","S","E"};

void find_path(char maze[10][11], int bx, int by, int x,int y,
                    string path, int numBomb, bool &found );

int main()
{
      char maze[10][11] = {
           "SWWWWWWWWW",
           "..WWWWWWWW",
           "W...WWWWWW",
           "W.WRWWWWWW",
           "..W.....W",
           ".WWWRW.WRW",
           ".WWWRW.W.W",
           "RWW..W.RWW",
           "WWWWWW.WRQ",
           "WWWWWW...W"
      };

      bool found = false;
      string path;
      find_path(maze, 10,11, 0, 0, path, 2, found);
}
```

```
void find_path(char maze[10][11], int bx, int by, int x,int y,
                string path, int numBomb, bool &found )
{
    if(found) return ;

    for(int i=0;i<4;i++) {
        int next_x = x + direction[i][0];
        int next_y = y + direction[i][1];
        if( next_x >= 0 && next_y >= 0 && next_x < bx && next_y < by) {
            if( maze[ next_x ][ next_y ] == '.' ) {
                maze[ next_x ][ next_y ] = 'V';
                find_path(maze, bx , by , next_x, next_y,
                        path + dir_str[i] , numBomb, found );
                if ( ! found)
                    maze[ next_x ][ next_y ] = '.';
            }
            else if( maze[ next_x ][ next_y ] == 'R' && numBomb > 0 ) {
                // Smash the rocks with your bomb if you have any.
                maze[ next_x ][ next_y ] = 'V';
                find_path( maze, bx, by, next_x, next_y,
                        path + dir_str[i] + " Bomb!\n", numBomb-1, found );
                if ( ! found)
                    maze[ next_x ][ next_y ] = 'R';
            }
            else if(maze[ next_x ][ next_y ] == 'Q') {
                path += dir_str[i] + " Found Exit!\n";
                found = true;
                for(int j=0;j<path.size();j++)
                    cout << path[j];
            }
        }
    }
}
// If you can solve this problem, you will return back to the civilized world.
// If not, then …. (BAD END)
```