HW 2, Part I

Due 23:00 October 12, 2017

CS ID: **SOLUTION**

Section (circle one):

Monday	L1J 9–11a	L1N 11a–1p	L1C 4–6p
Tuesday	L1A 11a–1p	L1F 4–6p	
Wednesday	L1B 9–11a	L1E 2–4p	L1M 6–8p
Thursday	L1D 10:30a–12:30p	L1G 4–6p	
Friday	L1K 1–3p	L1H 3–5p	

Please print out this document and write your solutions into the spaces provided. Show your work where necessary for full credit. If you require additional space, please indicate in the question space that you are writing on the last blank page, and also indicate on the blank page which question the work solves.

You must scan and upload the completed document, including this page and the last page, to GradeScope, using your 4- or 5-character CS ID.

1. **[Choices, Choices! – 36 points].**

   In each of the multiple choice problems below, bubble the best answer. In a few cases, more than one response is correct. Mark them both! You are welcome to explore some of the coding problems using a compiler, but be careful–sometimes the system gives you an unreasonable, reasonable answer. A good way to solve these problems is to speculate on a response *without* using the compiler, and then check to see if the code behaves as you predicted.

MC1 (1.5pts)

```
class Coffee{
  public:
    bool awesome;
    void setSugar();
  private:
    int oz;
    bool sugar; };

void Coffee::setSugar() { // code code }
void serveCoffee() { // code code }
int main() {
    Coffee c;
    return 0;
}
```

Where could we assign `awesome = true;`?

○ In the `serveCoffee` function.

[✓ **In `main`, if we made it `c.awesome=true;`.**]

○ Only in a constructor for the class.

○ In another file that does not use `Coffee`.

○ None of the other options is correct.

MC2 (1.5pts)

```
class LegoMov{
  public:
    void setEvIsAwes(bool b);
  private:
    bool evIsAwes; };

void LegoMov::setEvIsAwes(bool b) { evIsAwes = b; }
int main() {
    LegoMov movie;
    LegoMov.setEvIsAwes(true);
    return 0;
}
```

What is the problem with this code?

○ `LegoMov` functions aren't scoped correctly.

[✓ **`main` calls `LegoMov`'s functions incorrectly.**]
○ `LegoMov` is missing a constructor.
○ `LegoMov` is missing a destructor.
○ None of the other answers is true.

MC3 (1.5pts)

```
flower * plantANew(flower & orig) {
    flower * seed = new flower(orig);
    return seed;
}
int main() {
    flower f1; flower * f2;
    f2 = plantANew(f1);
    return 0;
}
```

In the code above, assume the `flower` class has default and copy constructors defined. How many times is a `flower` constructor called?
○ Never, but the code executes with no errors.
○ Never, because this code has a compiler error.
○ One time.
[✓ **Twice.**]
○ Three times.

MC4 (1.5pts)

```
void foo(int *bar) {
  *bar = 7;
}
int main() {
    int *x = new int(3);
    foo(x);
    cout << *x << endl;
    //code for deleting memory
    return 1;
}
```

What is the result of compiling and running this code?
○ The number 3 is printed to the screen.
○ The number 1 is printed to the screen.
[✓ **The number 7 is printed to the screen.**]
○ Nothing is printed to the screen.
○ This code has a compilation error.

MC5 (1.5pts)

```
int foo (int &a, int &b) {
  b = b + a;
  a++;
  return a;}

int main() {
  int x=5, y=2;
  int z = foo(x,y);
  cout << z <<"\t" << y << "\t" << x;
  return 0;
}
```

Which of the following options below would be printed out by the program?

[✓ **6 7 6**]
○ 6 7 5
○ 5 2 5
○ 6 2 6
○ None of the other answers are correct.

MC6 (1.5pts)

Consider this simple code, and assume the `flower` class has default and copy constructors defined:

```
flower * plantANew(flower & orig) {
    flower * seedling = new flower(orig);
    return seedling;
}

int main() {
    flower f1; flower * f2;
    f2 = plantANew(f1);
    return 0;
}
```

How many times is the `flower` copy constructor called in the example above?

○ Never, but the code executes with no errors.
○ Never, because this code has a compiler error.
[✓ **One time.**]
○ Twice.
○ Three times.

MC7 (1.5pts)

Consider this simple example

```
class Bear {
   public:
      Bear() { cout << " Growl"; }
      ~Bear() { cout << " Stomp stomp stomp"; }
};

int main() {
   Bear beary;
   cout << " Run!";
   return 0;
}
```

What is the result of compiling and executing this code?

○ `Run!`
○ `Growl Run!`
[✓ `Growl Run!  Stomp stomp stomp`]
○ `Run!  Stomp stomp stomp`
○ This code does not compile.

MC8 (1.5pts)

```
int * p;
int i;
i = 37;
*p = i;
*p = 99;
cout << i << endl;
```

What is the result of executing these statements, assuming that `iostream` is included?

[✓ **This code results in undefined runtime behavior.**]
○ '37' is sent to standard out.
○ '99' is sent to standard out.
○ This code does not compile.
○ This code has a memory leak.
○ None of the other options describes the behavior of this code.

MC9 (1.5pts)

```
class textBlock{
   public:
      textBlock(const string & s):text(s) {}
      char & operator()(int position)
         { return text[position]; }
   private:
      string text;
};
int main() {
   textBlock t("code monkey");
   for (int i = 0; i < 11; i ++)
      // Your answer goes here!

   return 0;
}
```

Which of the following statements complete the code above so that the output is `code monkey`?

- [✓ `cout << t(i);`]
- ○ `cout << t;`
- ○ `cout << text[i];`
- ○ `cout << t[i];`
- ○ More than one of the other answers produces the correct output.

MC10 (1.5pts)

```
int * a;
int * b;
b = new int(5);
a = b;
*a = 9;
cout << *b << endl;
delete b;
a = NULL;
b = NULL;
```

What is the result of executing these statements if you assume the standard `iostream` library has been included?

- ○ 5 is sent to standard out.
- ○ This code results in undefined runtime behavior.
- ○ The memory address of `b` is sent to standard out.
- [✓ **9 is sent to standard out.**]
- ○ This code has a memory leak.
- ○ None of the other options describes the behavior of this code.

MC11 (1.5pts)

In a sorted singly linked list containing n nodes, the time required to find the maximum element is (**NOTE:** assuming there is only a pointer to the front of the list):

- [✓ $O(n)$]
- ○ $O(n \log n)$
- ○ $O(n^2)$
- ○ $O(\log n)$
- ○ $O(1)$

MC12 (1.5pts)

Consider the following function definition and suppose that 1) the `node` class consists of an integer `data` element, and a `node` pointer called `next`, and 2) variable `head` is the address of a linked list of such `node`s.

What does the function do?

```
void fun(node * curr) {
   if (curr != NULL) {
      cout << curr->data;
      if (curr->next != NULL) {
         fun(curr->next->next);
      }
   }
}

node * head = NULL;
// maybe insert data into the chain here
fun(head);
```

- ○ `fun` prints the elements of the list from `head` to the end.
- ○ `fun` prints the reverse of the list.
- ○ `fun` segfaults on lists of odd length.
- [✓ **`fun` prints every other element of the list.**]
- ○ None of the other options is correct.

MC13 (1.5pts)

Which of the following List ADT implementations gives us an $O(1)$ time for `insertAtEnd`, i.e inserting an element at the end of the list?

○ A singly-linked list with only a `head` pointer.
[✓ **A singly-linked list with `head` and `tail` pointers.**]
○ A doubly-linked list with only a `head` pointer.
[✓ **A doubly-linked list with `head` and `tail` pointers.**]
○ None of the other options is correct.

MC14 (1.5pts)

Consider a class `List` that is implemented using a singly linked list with a `head` and `tail` pointer (i.e. pointers to the first and last nodes in the list). Given that representation, which of the following operations could be implemented in $O(1)$ time?

[✓ **Insert item at the front of the list**]
[✓ **Insert item at the rear of the list**]
[✓ **Delete front item from list**]
○ Delete rear item from list
○ All of them

MC15 (1.5pts)

In a doubly linked list, what will be the time required to insert at the middle position of the list?

○ $O(1)$
[✓ $O(n)$]
○ $O(n \log n)$
○ $O(\log \log n)$
○ $O(\log n)$

MC16 (1.5pts)

We have implemented the Stack ADT as an array. Every time the array is full, you resize the array creating a new array that can hold four times as many elements as the previous array and copy values over from the old array. What is the total running time for n pushes to the stack?

○ $O(1)$
○ $O(\log n)$
[✓ $O(n)$]
○ $O(n \log n)$
○ $O(n^2)$

MC17 (1.5pts)

Suppose a `std::stack<int> s` contains 6 elements 1, 2, 3, 4, 5, 6 (pushed in that order). What is the result of executing the following code snippet, assuming all required libraries are included?

```
for(int i = 0; i<6; i++){
   if(i%2==0) {
      s.push(s.top());
      s.pop();
   }
}
```

[✓ **`s` remains the same.**]
○ The elements of `s` are reversed.
○ The odd numbers in `s` are reversed.
○ The even numbers in `s` are reversed.
○ The top half of `s` contains even elements and the bottom half of `s` contains odd elements

MC18 (1.5pts)

In implementing Queue ADT, using which of the following data structure gives best asymptotic runtime for `enqueue` and `dequeue`? (Assume best possible implementation for queue using provided data structure)

[✓ **Doubly linked list with head and tail pointer.**]
○ Singly linked list with head pointer only.
[✓ **Singly linked list with head and tail pointer.**]
○ Doubly linked list with head pointer only.

MC19 (1.5pts)

Suppose we have implemented the Queue ADT as a singly-linked-list with `head` and `tail` pointers and no sentinels. Which of the following best describe the tightest running times for the functions `enqueue` and `dequeue`, assuming there are $O(n)$ items in the list, and that the front of the queue is at the `head` of the list?

[✓ $O(1)$ **for both.**]
○ $O(n)$ for both.
○ $O(1)$ for `enqueue` and $O(n)$ for `dequeue`.
○ $O(n)$ for `enqueue` and $O(1)$ for `dequeue`.
○ None of the options is correct

MC20 (1.5pts)

Fill in the blanks so that the following sentence is true: If you have a complete tree with 17 nodes, the maximum height (h) of the tree is $\boxed{}$ and there are $\boxed{}$ nodes on level h.

○ First blank is 4, second is 1.
○ First blank is 5, second is 2.
○ First blank is 8, second is 2.
○ First blank is 8, second is 9.
[✓ **None of the other options makes the sentence true.**]

MC21 (1.5pts)

Consider the binary tree class with 1) variable `root` that is the `treeNode` representing the root of the binary tree and 2) each `treeNode` consists of an integer `data` element, and two `treeNode` pointers called `left` and `right`.

What does fun(`root`) return?

```
int fun(treeNode * curr) {
   if (curr != null) {
      int ret1 = fun(curr->left);
      int ret2 = fun(curr->right);
      return 1 + max(ret1, ret2);
   }
   else return -1;
}
```

[✓ **`fun` returns the height of the tree.**]
○ `fun` returns the number of elements in the tree.
○ `fun` returns the sum of all elements in the tree.
○ `fun` returns the shortest distance from root to leaf.
○ None of the other options is correct.

MC22 (1.5pts)

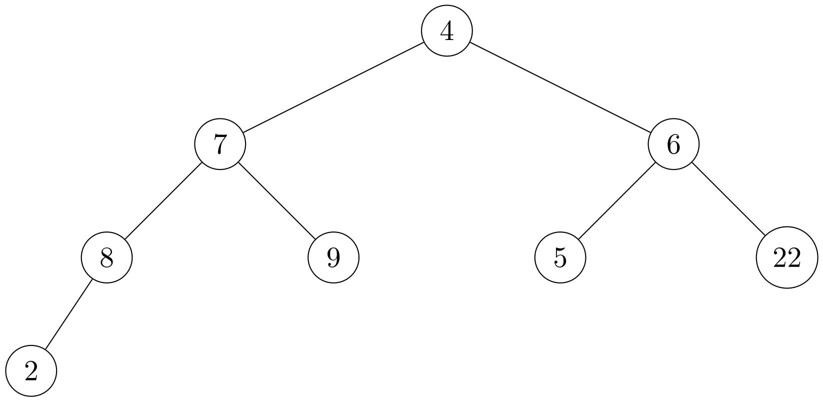
Choose the appropriate running time from the list below. The variable n represents the number of items (keys, data, or key/data pairs) in the structure and h represents the height of the tree. In answering this question you should assume the best possible implementation given the constraints, and also assume that every array is sufficiently large to handle all items (unless otherwise stated).

Determine if the Binary Tree is ordered. (i.e at every node k, all the elements in the k's left subtree are smaller or equal to k's key and all the elements in the k's right subtree are larger or equal to k's key.)
○ $O(1)$
○ $O(h)$

[✓ $O(n)$]
○ $O(n^2)$
○ None of the options is correct.

MC23 (1.5pts)

In the level order traversal of the following binary tree, which is the last node that will be dequeued before the node 9 is enqueued?



○ 4
[✓ **7**]
○ 6
○ 8
○ None of the options is correct

MC24 (1.5pts)

Choose the appropriate running time from the list below.

The variable n represents the number of items (keys, data, or key/data pairs) in the structure. In answering this question you should assume the best possible implementation given the constraints, and also assume that every array is sufficiently large to handle all items (unless otherwise stated).

Perform a level-order traversal of a Binary Tree.

○ $O(1)$
○ $O(\log n)$
[✓ $O(n)$]
○ $O(n \log n)$
○ $O(n^2)$

2. **[Polynomials – 28 points].**

The last time you thought about polynomials you were probably falling asleep in your grade 10 math course. They actually underlie a rich and important computational discussion! Polynomials can be used to estimate any continuous function (see the Weierstrass Approximation Theorem), and as such, they're broadly useful in applications, including motion planning, computer graphics, computer vision, physical system modeling, etc. In this problem we will dig into a few of the computational issues that arise in the subproblem of polynomial evaluation.

The *power form* of a polynomial is:

$$p(x) = a_0 + a_1x + a_2x^2 + \ldots + a_nx^n, a_n \neq 0.$$

(a) (4 points) Write pseudocode for a function called $pow(x, n)$ that returns the value of x^n. The runtime of your algorithm should be $\Theta(n)$. Your pseudocode may use the fundamental operations of addition and multiplication, but it may only invoke one of each within any assignment statement.

```
int pow(int x, int n) {
    int result = 1;
    for (int i = 0; i < n; i++) {
        result *= x;
    }
    return result;
}
```

(b) (4 points) Given a real valued array A of length $n+1$, and a real valued number x, write pseudo-code to define a function that computes the power form of a polynomial $p(x)$, where the coefficients a_i are the entries in A. Your pseudocode may use the fundamental operations of addition and multiplication, but it may only invoke one of each within any assignment statement. Please call $pow(x, n)$ to compute x^n.

```
int poly(std::vector<int> A, int x) {
    int result = 0;
    for (int n = 0; n < A.size(); n++) {
        result = result + A[n]*pow(x, n);
    }
    return result;
}
```

(c) (4 points) What is the asymptotic running time of your $p(x)$ function? $\boxed{O(n^2)}$

(d) (4 points) Write pseudocode for a *recursive* algorithm (called $recpow(x, n)$) to compute x^n. It should have strictly better asymptotic running time than the function you wrote in part a). Hints: i) $x^{13} = x \cdot x^6 \cdot x^6$; ii) you will need two base cases.

```
int recpow(int x, int n) {
    if (n <= 0) {
        return 1;
    }
    int temp = recpow(x, n/2);
    int result = temp*temp;
    if (n % 2 == 0) {
        return result;
    }
    return result*x;
}
```

(e) (4 points) What is the asymptotic running time of your $p(x)$ function if it calls $recpow(x, n)$ instead of $pow(x, n)$? $\boxed{O(n\,lg\,n)}$

(f) (4 points) Verify for yourself that the power form of the polynomial can be rewritten as

$$p(x) = a_0 + x(a_1 + x(a_2 + ... + x(a_{n-1} + a_n x))).$$

Write pseudocode to use this equivalent form to evaluate the polynomial.

```
int horner(std::vector<int> A, int x) {
    int n = A.size();
    int result = A[n-1];
    for (int i = n-2; i >= 0; i--) {
        result = result*x + A[i];
    }
    return result;
}
```

(g) (4 points) What is the asymptotic running time of this new $p(x)$ function? $\boxed{O(n)}$

This technique is called "Horner's Method", and it's the best known algorithm for exact evaluation of general polynomials.

Blank sheet for extra work.