COMP 220: Data Structures **December 08, 2017**

Exam 7

Instructions:

- This exam is NOT open book or notes.
- Partial answers will receive partial credit. Make every effort to at least put something for each question. If you don't know the answer, you can get plenty of partial credit by giving a partial answer and explaining where it is wrong or where you got stuck.
- If you believe a problem is incorrectly or incompletely specified, make a reasonable assumption and solve the problem. The assumption should not result in a trivial solution.
- In all cases, clearly state any assumptions you make in your answers.

Name	

Part	Description	**Points Possible**	**Grade**
1	Multiple Choice	15	
2	Short Answer	10	
3	Trees	30	
4	Linked Lists	15	
5	Hash Tables	30	
6	Bonus	20	
Total		100	

Multiple Choice

1. [3 points] What is the minimum height for a binary tree of n nodes?

   (a) 1

   (b) $\log_2 n$

   (c) n

   (d) n^2

   Your Answer B

2. [3 points] What is the maximum height for a binary tree of n nodes?

   (a) 1

   (b) $\log_2 n$

   (c) n

   (d) n^2

   Your Answer C

3. [3 points] How many different ways can you draw a binary tree of n nodes at its maximum allowable height?

   (a) 1

   (b) 2

   (c) $n!$

   (d) 2^n

   Your Answer B

4. [3 points] Which of the following sorting algorithms take $O(n \log n)$ time *in the worst case*?

   (a) Quicksort

   (b) Mergesort

   (c) Both A and B

   (d) Neither A nor B

   Your Answer B

5. [3 points] Your company needs to store many key-value pairs. During the course of the day you need to perform many inserts and lookups, so they must be fast. At the end of each day you need to iterate over all pairs to generate a report, in order of the keys in the key-value pair. Which data structure should you use?

   (a) An AVL tree

   (b) A hash table

   (c) A vector with binary search for lookups

   (d) A linked-list with linear search for lookup

   (e) A plain binary search tree

   Your Answer A

Short Answer

6. [6 points] Rank each of the following functions from least to greatest complexity class order such that rank 1 is the least complex. If two functions are equally complex then rank them the same. For example, if two functions are found to be in the same class and that class is the least complex of all the examples, then rank them both with 1.

$f(n)$	Rank
$\frac{3}{15}n + 1.5E16$	$O(n) \rightarrow 2$
$\frac{n \log n}{25} - 13n + 5n^3$	$O(n^3) \rightarrow 4$
$\frac{(n+1)(n+2)}{n}$	$O(n) \rightarrow 2$
1.3×10^{19}	$O(1) \rightarrow 1$
$\log n + 135 + \frac{n}{2}$	$O(n) \rightarrow 2$
$n \log n + 42n$	$O(n \log n) \rightarrow 3$

7. [4 points] Consider the following code:

```
for (int i{0}; i < n; ++i) {
  for (int j{0}; j < n; ++j) {
    foo(i,j);
  }
}
```

What is the time complexity class of this code, in terms of n? Why? You may assume that foo(i,j) takes $O(1)$ time.

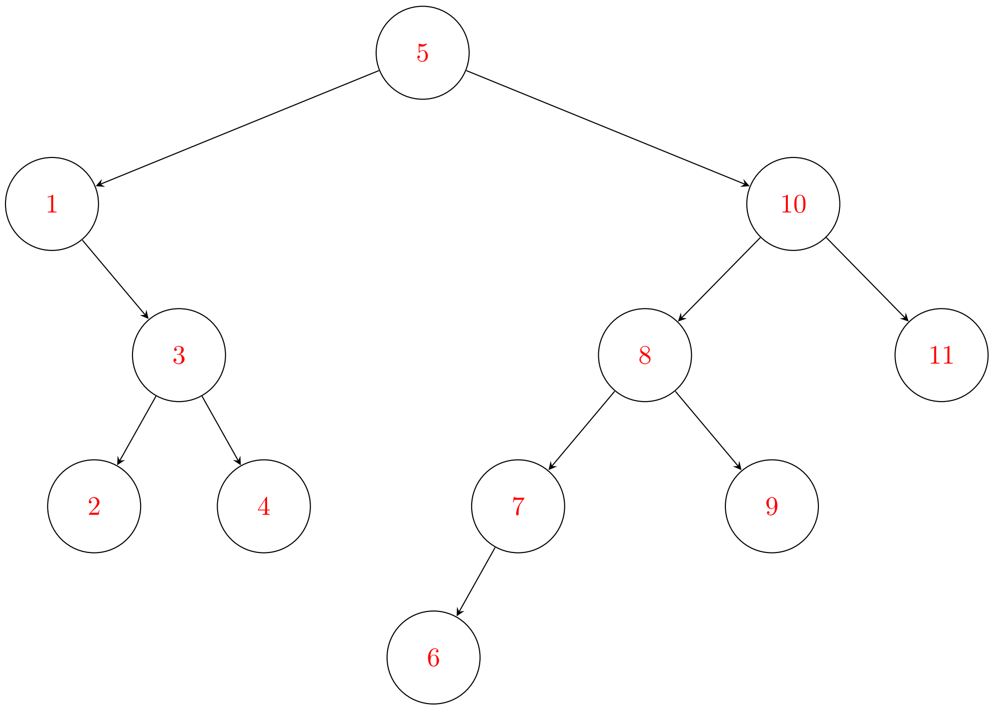
$O(n^2)$

Trees

8. Do the following for the tree given below:

   (a) [2 points] Draw a rectangle around all the leaf nodes.

   (b) [5 points] Label the nodes with 1 to 11 such that an inorder traversal would visit them in least to greatest numerical order. By "visit" I mean the processing step of the traversal.



Nodes 2, 4, 6, 9, and 11 are leaves.

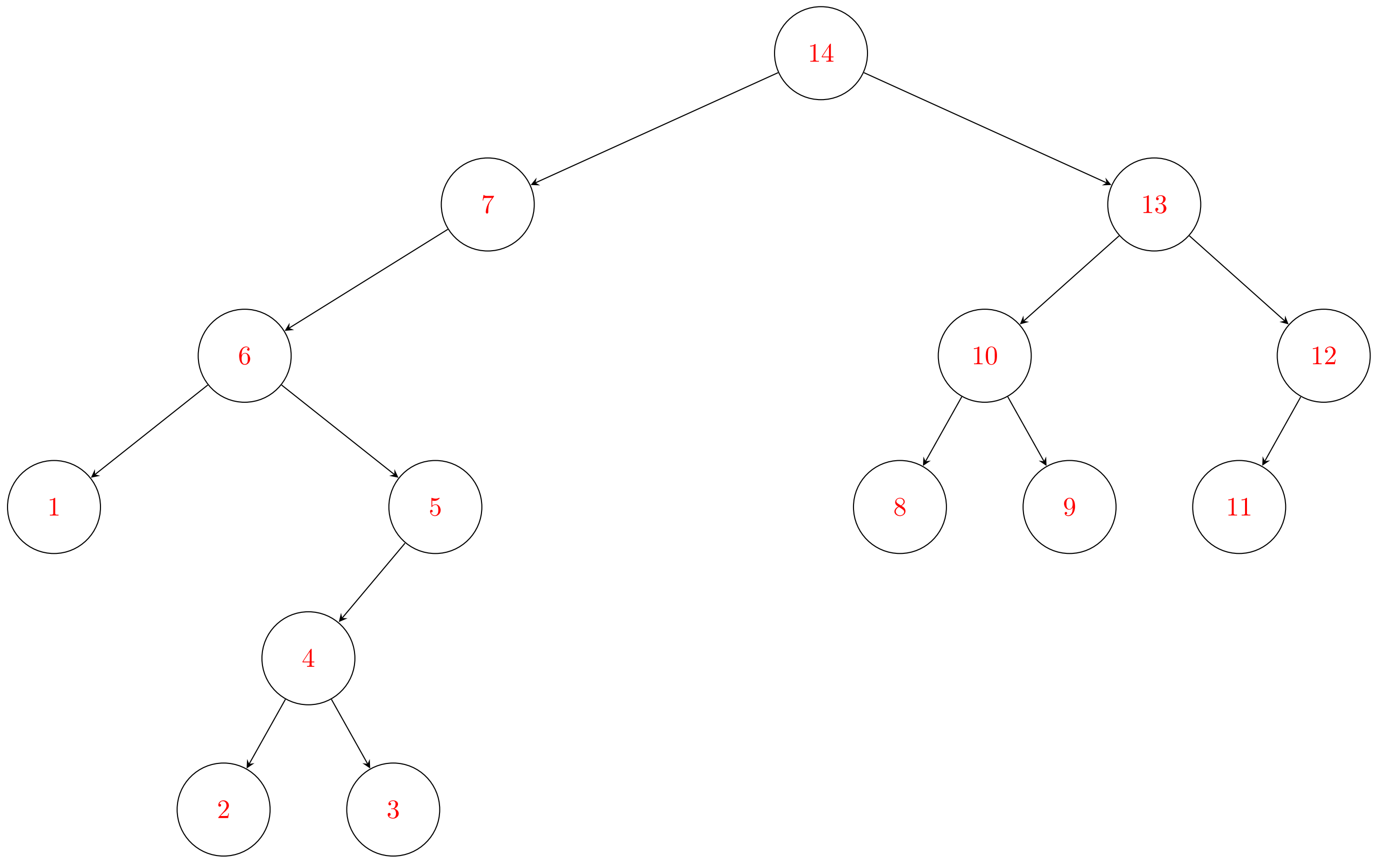
9. Do the following for the tree given below:

   (a) [2 points] The height of the tree is 5______________ .

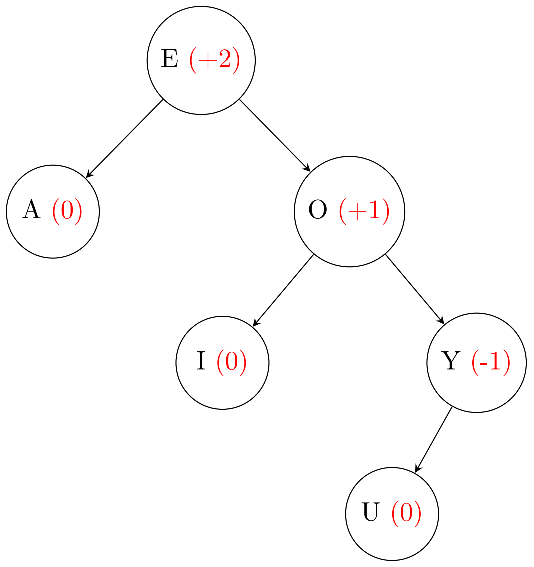
   (b) [5 points] Label the nodes with 1 to 14 such that a postorder traversal would visit them in least to greatest numerical order.

   (c) [3 points] **True** or **False**: This tree is balanced. If you selected false, then add a minimal number of nodes required to make it balanced.

   False



10. Consider the following tree:

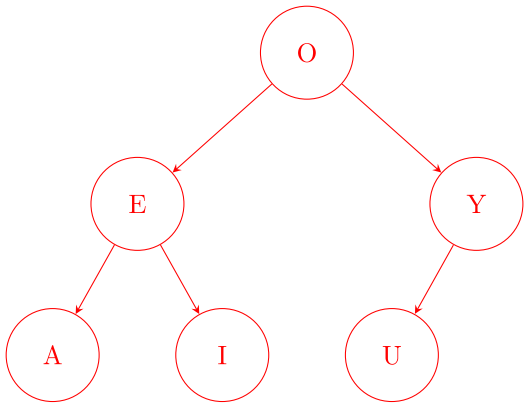


(a) [5 points] For each node, write its balance factor next to it.

(b) [2 points] Describe the rotation necessary to balance the tree.

Left rotation at E, i.e. rotation on the E-O axis.

(c) [6 points] Draw the tree after the rotation. You do not need to calculate the balance factors.



Linked Lists

11. Suppose we define a class

```
class StringContainer {
  public:
  // ...

  private:
  struct Node { /* ... */ };
  Node *head;
};
```

which uses a linked list to store strings. Consider the following task:

> Modify the list by finding the first string of length less than some given length n and moving that node to the front (e.g. head) of the list. If all strings have length at least n, do nothing.

(a) [2 points] Provide the Node definition necessary for this class.

```
struct Node {
  std::string data;
  Node *next;
};
```

(b) [3 points] Provide the documentation and declaration for this method, which you should call moveToFront.

```
/** Moves the first string with length less than n to the
*  front of the list .
*  @param n the length limit of the string to move.
*  @pre none
*  @post If there exists a string with length less than n in
*  the list , the first such string will now be at the head of
*  the list . All strings that were in the old list remain in
*  the new list , though their relative order may have changed.
*/
void moveToFront(int n);
```

(c) [10 points] Write the complete implementation of this procedure.

```
void StringContainer::moveToFront(int n) {
  Node* current = head;
  Node* prev = nullptr;
  while (current) {
    if (current->data.length() < n)
    break;
    prev = current;
    current = current->next;
  }
  if (current == nullptr) return; // No valid string found
  if (prev == nullptr) return // already at head
  prev->next = current->next;
  current->next = head;
  head = current;
}
```

Hash Tables

12. Consider the following start to the declaration of a bucket-based HashMap class template.

```
template <typename K, typename V>
class HashMap {
  public:
  // ...
  V& operator[](const K& key);

  private:
  struct Node {
    K key;
    V value;
    Node∗ next;
  };

  Node∗∗ buckets;
  int numBuckets;

  static int hashCode( ... );

  static Node∗ find(Node∗ head, const K& key);

};
```

(a) [2 points] What should replace the ... in the hashCode function declaration?

```
const K& key
```

(b) [3 points] Why is Node∗∗ the type of the instance variable buckets?

The instance variable buckets buckets is an array of buckets, i.e. it is a pointer to a block of memory. Each bucket itself is a linked list, so in each "slot" of the bucket array we need a pointer to the head of the list. In other words, buckets is a pointer to a pointer to a Node.

(c) [4 points] Provide the declaration of the copy constructor as it would appear in the public section of the class template and then write a stub for the constructor as it would appear in the implementation.

```
/** Constructs a HashMap by copying from another HashMap.
*  @param src The HashMap to copy from.
*  @pre src is a valid HashMap
*  @post The constructed HashMap contains the same
*  entries as src, but in completely disjoint memory
*  locations.
*/
HashMap(const HashMap<K,V>& src);

template <typename K, typename V>
HashMap<K,V>::HashMap(const HashMap<K,V>& src) {
  buckets = nullptr;
  numBuckets = 0;
}
```

(d) [4 points] Provide the declaration of the destructor as it would appear in the public section of the class template and then write a stub for the destructor as it would appear in the implementation.

```
/** Destroys a HashMap by recycling all allocated memory.
*  @pre The object is a valid HashMap.
*  @post All allocated memory has been returned to the heap.
*/
~HashMap();

template <typename K, typename V>
HashMap<K,V>::~HashMap() { }
```

(e) [7 points] The helper procedure find searches a linked list starting at head for the Node containing the key value key. If the list contains no such Node, then find returns nullptr. Give a complete implementation of this procedure.

```
template <typename K, typename V>
struct HashMap<K,V>::Node* HashMap<K,V>::find(Node* head, const K& key)
     {
  while (head != nullptr) {
     if (head->key == key) break;
     head = head->next;
  }
  return head;
}
```

(f) [10 points] Assume that a complete implementation of the helper hashCode is available. Write the complete implementation for operator[]. (Hint: use find.)

```
template <typename K, typename V>
V& HashMap<K,V>::operator[](const K& key) {
  int index = hashCode(key) % numBuckets;
  Node *bucket = buckets[index];
  Node *n = find(bucket, key);
  if (!n) {
    n = new Node;
    n->key = key;
    n->next = bucket;
    buckets[index] = n;
  }
  return n->value;
}
```

Bonus

13. [5 bonus points] Consider the following implementation of the HashMap destructor:

```
template <typename K, typename V>
HashMap<K,V>::~HashMap() {
  for (int i{0}; i < numBuckets; ++i) {
    Node *current = buckets[i];
    while (current) {
      delete current;
      current = current->next;
    }
  }
  delete buckets;
}
```

Describe the bug(s) in the code.

The variable current is deleted and then immediately accessed. C++ does not allow this.

14. [5 bonus points] Rewrite the destructor to fix the bug.

```
template <typename K, typename V>
HashMap<K,V>::~HashMap() {
  for (int i{0}; i < numBuckets; ++i) {
    Node *current = buckets[i];
    while (current) {
      Node* tmp = current;
      current = current->next;
      delete tmp;
    }
  }
  delete buckets;
}
```

15. [10 bonus points] Write the implementation of the copy constructor for HashMap. You may **NOT** assume any kind of deepCopy method is available!

```
template <typename K, typename V>
HashMap<K,V>::HashMap(const HashMap<K,V>& src) {
  numBuckets = src.numBuckets;
  buckets = new Node*[numBuckets];
  for (int i{0}; i < numBuckets; ++i) {
    Node *current = src.buckets[i];
    while (current) {
      Node *tmp = new Node;
      tmp->key = current->key;
      tmp->value = current->value;
      tmp->next = buckets[i];
      buckets[i] = tmp;
      current = current->next;
    }
  }

}
```