CPSC 223: Exam 2 Study Guide

Problems 1 & 2: Problems 1 & 2 refer to the collection defined below.

The collection contains:

```
Collection();                          int popFront();
int getItem(int index);                void insertFront(int item);
int size();                            void insertBack(int item);
```

1. Write an algorithm that takes a collection and returns a collection containing the same items in the reverse order. For example, if the original collection was to contain 1,2,3,4, then the new collection would contain 4, 3, 2, 1. You can destroy 'c' in the process.

   ```
   Collection reversed(Collection c)
   ```

2. Analyze the Big-O for the functions of the collection if implemented as an array and as a Linked List.

	Array	Linked List
`getItem`		
`size`		
`popFront`		
`insertFront`		
`insertBack`		

3. Write an algorithm that returns the smallest (absolute) difference between any two members of the array. You can assume no difference will be bigger than 1000 and that no two elements will be the same. *Hint: the* `abs()` *function might be useful.*

   ```
   int smallestDifference(int data[], int n)
   ```

4. Complete the following summation identities.

$\sum_{i=l}^{u} 1 =$ $\sum_{i=1}^{n} i =$ $\sum_{i=1}^{n} \log i =$

Problems 5 & 6: Problems 5 & 6 refers to the Animal class defined below in the file Animal.h.

```
#pragma once

#include <string>

using namespace std;

class Animal {
public:
    virtual ~Animal() {}

    virtual void eat(string what) = 0;

    virtual void speak() = 0;


    // Whatever type of animal this is called in...it will create an
    // exact copy of this animal you can use like the original animal
    virtual Animal* clone() = 0;
};
```

5. Choose two different animals that you want to represent. Implement them as subclasses. You will be creating two .h files. Creativity is required for this section. Note: additional space is provided on the following page. You can assume the following code exists at the top of both files:

```
#pragma once
#include <iostream>
#include "Animal.h"
```

6. You will now create a Cage class in the file below Cage.h. You will store a pointer to an Animal in the cage (set via constructor). You will be able to access the Animal in the cage. When the Cage's destructor runs, the animal will be deleted.

```
#pragma once

#include "Animal.h"

class Cage {
public:
    Cage(Animal* initAnimal)
    {

    }

    ~Cage()
    {

    }

    Animal *getAnimal()
    {

    }

    Cage &operator=(const Cage& other)
    {









    }

    Cage(const Cage &other)
    {

    }

private:
    Animal *animal;
};
```

7. Following the Rule of Three, if a class has a non-trivial destructor, it should either have (or delete) the _____ _____ and _____ _____.

8. We implement the assignment operator as a _____ _____.

9. In regards to the assignment operator, for a class named Tree, the signature is

10. When is the assignment operator used? What is the result of an assignment operator?

11. In regards to the copy constructor, for a class named Board, the signature is:

12. When is the copy constructor used?

13. Recalling assignment 2 (a sorted Dictionary), write a copy constructor. Remember, there are two arrays, keys and values, of types K and V, respectively. *Hint: it may be helpful to use the function* `maxSize`.

14. Describe the difference between static and dynamic dispatch.

15. Dictionaries are an example of a/an _____ data structure.

16. With dictionaries, the _____ allows us to look up pieces of data associated with it called _____.

Problems 17-19: Problems 17-19 are a reference to assignment 2.

17. What is the solution to improve the speed to look up a key's index?

18. To implement this improvement, what three steps must be followed when adding a key/value pair?

19. To implement this improvement, what step must be followed when removing a key/value pair?

20. What is the binary search tree property?

21. Write the templated struct of type T for a Binary Search Tree with Node that holds data, a pointer to the left, a pointer to the right, and a pointer to the parent.

22. Write the pseudocode for adding an element to a binary search tree.

23. Does the order elements are added to a binary search tree affect the tree structure? Why or why not? It may be helpful to give an example. *Hint: How does the first element added affect the tree structure?*

24. Write the pseudocode for a function the returns true if an element exists and false if an element does not exist. *Hint: a helper function is almost absolutely needed.*

```
bool doesElementExist(itemType item)
```

25. Describe the process for removing an element from a binary search tree in the following cases.
    a. Leaf nodes:

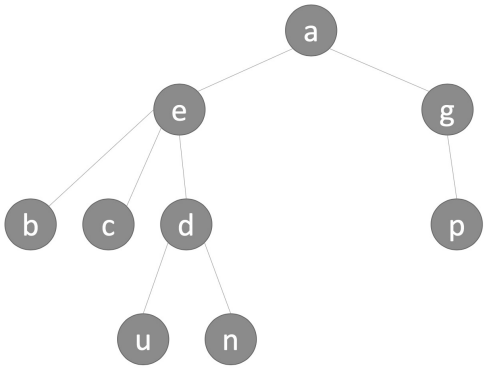
    b. Nodes with a single child:

    c. Nodes with two children:

26. What is the difference between a full binary tree and a complete binary tree?

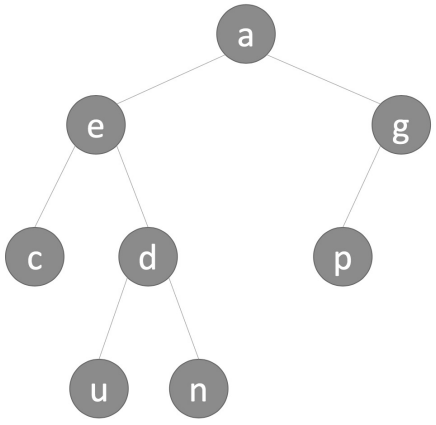
27. What does it mean if a tree is balanced?

28. Given the following binary tree, complete the statements.



a. **d** is the _____ of **u** and **n**.
b. **b** is a _____ of **e**.
c. **a** is the _____.
d. All of the nodes are _____.
e. The lines between the nodes are _____.
f. The relationship among **b**, **c**, and **d** is _____.
g. Unlike **c** and **d**, **b** is **e**'s _____.
h. **p** is an example of a _____.
i. All the nodes, except for **a**, are _____ of **a**.
j. **b**, **c**, and **d** are **e**'s _____.
k. The height of the tree is _____.
l. **u** and **n** are at level _____.
m. **a** is an _____ to all the nodes except for **a**.

29. Given the following binary tree, identify the results of a preorder, in-order, and postorder traversal.



a. preorder:
b. in-order:
c. postorder:

30. The _____ traversal of a binary search tree visits the tree's nodes in sorted order.

31. What is an exception? What do they indicate?

32. Describe the process when an exception is triggered. Also, provide the more proper term for triggered.

33. Heaps are data structures built off of _____ _____.

34. In this course, we deal with _____ (type of heap). This means (three things):

35. In the case of this course and the type of heap identified in problem 36, the root node is the _____ of all the elements in the heap.

36. Heaps have four main functions. Identify and describe them. *Hint: you should go into depth with the description of two of the functions. The other two descriptions should be very concise.*

37. How can heaps be stored?

38. When a heap is stored as an array, let C_L be the index that contains the left child, C_R be the index that contains the right child, and P be the index that contains the parent. Define the "formulas" to find C_L, C_R, and P.

39. Describe selection sort. Write the sorting algorithm's operation in sigma notation and then solve for the Big-O. Then write the code for the algorithm. *Hint: You should want to create an additional function that will help you solve the overall algorithm.*

```
int selectionSort(double data[], int n)
```

40. Describe insertion sort. Write the sorting algorithm's operation in sigma notation and then solve for the Big-O. Then write the code for the algorithm.

```
void insertionSort(double data[], int n)
```

41. Describe bubble sort. Write the sorting algorithm's operation in sigma notation and then solve for the Big-O.

42. Describe heap sort. Write the sorting algorithm's operation in sigma notation and then solve for the Big-O.

43. Complete the table below for the Big-O of the following sorting algorithms in their best and worst cases: selection, insertion, bubble, and heap sort.

	Best	Worst
Select		
Insert		
Bubble		
Heap		

44. For the two functions below, assume that n represents the number of elements in data. Identify the basic operation and state how many times that basic operation occurs in terms of n as an expression as well as the corresponding big-O values.

```
int f(double data, int n)
{
   double a = 0;
   for(int i=0; i<n; i++)
   {
      a += data[i];
   }
   return a/n;
}
```

```
int g(double data[], int n)
{
   double b = 0;
   for(int i=0; i<n; i++)
   {
      double c = f(data, n);
      double d = data[i] – c;
      b += d*d;
   }
   return sqrt(b/n);
}
```