**CS010C Final Study Guide**

*Note: This study guide is much longer than the actual final to provide you with various problems for you to practice and understand.*

**Definitions**

*Please write a concise definition for the following data structures.*

1. Linked List
2. Doubly Linked List
3. Stack
4. Queue
5. Heap
6. Binary Search Tree
7. B-Tree
8. AVL Tree
9. Red Black Trees
10. Hash Tables
11. Big O

**Runtime Analysis (Data Structures)**

*Please fill out the runtime chart below for the various data structures and operations. An explanation is not required, but you should still know the reason for the appropriate runtime.*

|  | **Insert / Push** | **Search / Front / Top** | **Remove / Pop** | **Count** |
| --- | --- | --- | --- | --- |
| Linked List |  |  |  |  |
| Doubly Linked List |  |  |  |  |
| Stack (Array) |  |  |  |  |
| Stack (Linked List) |  |  |  |  |
| Queue (Array) |  |  |  |  |
| Queue (Linked List) |  |  |  |  |
| Max Heap |  |  |  |  |
| Binary Search Tree |  |  |  |  |
| B-Tree |  |  |  |  |
| AVL Tree |  |  |  |  |
| Red Black Tree |  |  |  |  |
| Hash Table |  |  |  |  |

**Runtime Analysis (Algorithms)**

*Please fill out the runtime chart below for the various algorithms, along with an explanation reasoning why the runtime is correct.*

|  | **Runtime + Explanation** |
| --- | --- |
| Selection Sort |  |
| Insertion Sort |  |
| Radix Sort |  |
| Quicksort |  |
| Mergesort |  |
| Heapsort |  |
| Binary Search |  |
| Finding Tree Height |  |

**Free Response Questions**

1. What are the **required** properties of a hash function?
2. What are the **desired** properties of a hash function?
3. How is an AVL tree different from a Binary Search Tree? Why is that additional property useful?
4. When printing key value pairs in the std::map data structure which is implemented via a Red Black tree, why do the keys appear in sorted order?
5. How and why does your Linux system not store passwords as plain text?
6. If a hash function only returns prime bucket numbers, how does this affect the hash table?
7. What is an example in which a Binary Search Tree would have O(n) runtimes for insert, delete, and search?
8. What are the different parts of a Hash Table?
9. What is the Rule of Three? Why should we manually define the rule of three?
10. What are some ways to check for input/user validation?

**Coding Questions**

1. Write the function headers that will compute the sum of all the nodes in a binary search tree. This function will be a part of the BSTree class. We want to compute this recursively, therefore we should expect 2 function headers.
2. Write a function header to accept 2 vectors and return the total number of elements in both vectors.
3. Write a function header to accept a vector and conduct an element-wise multiplication. This function should not return anything.
4. Write a function header to accept 2 vectors and merge the vectors and return the new array.
5. Write a function header for enqueuing elements into a max heap and should not return anything. The max heap will consist of string values.
6. Write a class with inline functions to implement **stack**. Ensure the implementation is memory safe, utilizes const and pass by reference if applicable. The included functions should be:

* push()
* pop()
* top()
* empty()

1. Write a class with inline functions to implement a **queue**. Ensure the implementation is memory safe, utilizes const and pass by reference if applicable. The included functions should be:

* push()
* pop()
* front()
* empty()

For the following questions, implement the functions, given the provided classes. Functions in **bold** are the functions to complete.

class MaxHeap {  
 private:

int MAX\_SIZE = 10;

int data[MAX\_SIZE];

int size = 0;

public:

**void enqueue(int val);**

**void dequeue();**

**int highest() const;**

**int smallest() const;**

**bool empty() const;**

**void resize();** // resize to double the capacity

};

struct Node {  
 int small;

int large;

Node\* left;

Node\* middle;

Node\* right;

int children; // either 2 or 3 to indicate 2 vs 3 node

}

class BTree {  
 private:

Node\* root;

**void preorder(Node\*) const;**

**void inorder(Node\*) const;**

**void postorder(Node\*) const;**

**bool contains(Node\*, int) const;**

public:

**void preorder() const;**

**void inorder() const;**

**void postorder() const;**

**bool contains(int) const;**

};

struct Node {  
 int data;

Node\* left;

Node\* right;

}

class BST {  
 private:

Node\* root;

**int height(Node\*) const;**

**int count(Node\*) const;**

**void insert(Node\*, int);**

bool contains(Node\*, int);

public:

**int height() const;**

**int count() const;**

**void insert(int);**

**BST& intersection(const BST&);**

bool contains(int);

};

**Drawing Diagrams / Data Structures**

1. Insert the following elements into a BST:

14 58 22 53 88 62 97 52

1. Insert the following elements into an AVL:

28 37 21 57 87 70 10 3

1. Insert the following elements into a Red Black Tree. Depict Black Nodes with the letter B:

72 92 94 41 48 34 42 32

Use the following hash functions for questions 28 - 30:

int hash(int x) { return x % 10; }

1. Insert the following elements in a Hash Table with **closed addressing**

82 1 40 67 7 17 11 86 44 65

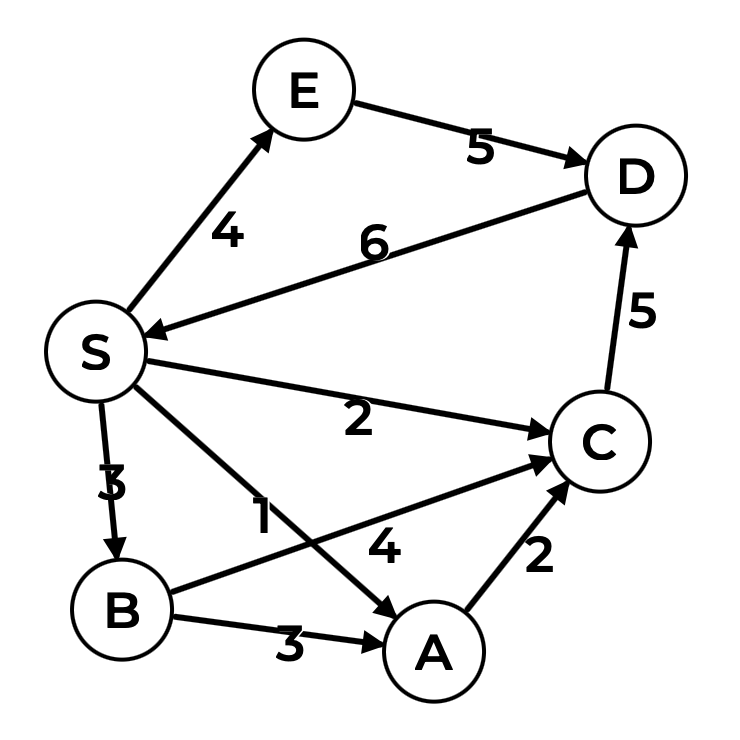
1. Insert the following elements in a Hash Table with **linear probing**

83 38 26 84 93 63 49 74 69 31

1. Insert the following elements in a Hash Table with **quadratic probing** with C1 = 1 and C2 = 2

56 33 61 91 27 19 66 6 39 41

1. List out the various steps involved in Dijkstra's algorithm.
2. Given the below graph, what would running Dijkstra's algorithm result, place the correct values into the table below. Start from symbol start node S. I want to see the history of your choices, so cross out prior entries (don’t erase). V is just there if you want to track visited nodes - it is not a graded component.



| **Label** | **Visited** | **Shortest Distance** | **Previous** |
| --- | --- | --- | --- |
| S |  |  |  |
| A |  |  |  |
| B |  |  |  |
| C |  |  |  |
| D |  |  |  |
| E |  |  |  |