**CSC 231 – Datastructures and Algorithms #**

**Final Study Guide**

**Part 1: Terms**

<http://xlinux.nist.gov/dads/>

<http://www.vuzs.info/extra-notes/53-cs301-data-structure-extranotes/5092-cs301-data-structure-glossary.html>

Unit 1: C++, Search, Sort, Big O

Algorithm: An ordered, executable set of steps to achieve a desired result.

Argument: A value passed to a called function by the calling function.

Array: In programming, a list of data values, all of the same type, any element of which can be referenced by an expression consisting of the array name followed by an indexing expression. Arrays are part of the fundamentals of data structures, which, in turn, are a major fundamental of computer programming

Asymptotic lower bound: An asymptotic bound, as a function of the size of the input, is the best (fastest, least amount of space used, etc.) an algorithm can possibly achieve to solve a problem. That is, no algorithm can use fewer resources than the bound.

Average case: Having to do with the mathematical average or median of all cases, the typical case.

Big-O notation: A theoretical measure of the execution of an algorithm, usually the time or memory needed, given the problem size n, which is usually the number of items. Big O notations indicates how the work of the algorithm increases as the size of the data set increases.

Binary Search: A type of search algorithm that seeks an item, with a known name, in an ordered list by first comparing the sought item to the item at the middle of the list's order. The search then divides the list in two, determines in which half of the order the item should be, and repeats this process, until the sought item is found. It’s complexity is O(log2 n).

C++: A compiled, general-purpose programming language that “has imperative, object-oriented and generic programming features, while also providing facilities for low-level memory manipulation. It was designed with a bias toward system programming and embedded, resource-constrained and large systems, with performance, efficiency and flexibility of use as its design highlights.” (Wikipedia).

Linear complexity: O(n) – linear search is an example.

Linear Search: A simple, though inefficient, search algorithm that operates by sequentially examining each element in a list until the target element is found or the last has been completely processed. Linear searches are primarily used only with very short lists. Also called sequential search.

Logarithmic complexity: O(log n) – binary search is an example.

Matrix: A two-dimensional array. By convention, the first index is the row, and the second index is the column.

Parameter: A value received by a called function from a calling function.

Quadratic complexity: O(n2) –examples include selection sort, bubble sort, and insertion sort.

Run time: The amount of time needed to execute an algorithm.

Selection Sort: A sort algorithm that repeatedly looks through remaining items to find the least one and moving it to its final location. The run time is O(n2), where n is the number of comparisons and the number of swaps is O(n).

Sort: Arrange items in a predetermined order. There are dozens of algorithms, the choice of which depends on factors such as the number of items relative to working memory, knowledge of the orderliness of the items or the range of the keys, the cost of comparing keys vs. the cost of moving items, etc.

String: A list of characters, usually implemented as an array. Informally a word, phrase, sentence, etc. Since text processing is so common, a special type with substring operations is often available.

Structure: A mechanism which allows objects of different types to be grouped together as a single compound type.

Unit 2: OOP, Recursion, Sort, and Big O

Bubble Sort: Sort by comparing each adjacent pair of items in a list in turn, swapping the items if necessary, and repeating the pass through the list until no swaps are done.

Insertion Sort: Sort by repeatedly taking the next item and inserting it into the final data structure in its proper order with respect to items already inserted.

Key: The part of a group of data by which it is sorted, indexed, cross referenced, etc.

Merge: Combine two or more sorted sequences of data into a single sorted sequence.

Merge sort: A sort algorithm that splits the items to be sorted into two groups, recursively sorts each group, and merges them into a final, sorted sequence.

Quicksort: An in-place sort algorithm that uses the divide and conquer paradigm. It picks an element from the array (the pivot), partitions the remaining elements into those greater than and less than this pivot, and recursively sorts the partitions

Recursion: An algorithmic technique where a function, in order to accomplish a task, calls itself with some part of the task.

Uniform matrix: A matrix having the same number of items in each row.

Worst case: The situation or input that forces an algorithm or data structure to take the most time or resources.

Unit 3: Lists, Pointers, and Object-Oriented Programming

Abstract Data Type: A set of data values and associated operations that are precisely specified independent of any particular implementation. Also known as ADT

Access operators: Operators that indicate that the data on the right of the operator belongs to the data on the left side of the operator. In c++, access operators are the . and the ->. The -> operator is used when data on the left is a pointer which is being implicitly dereferenced, thus **x->y** is equivalent to **\*(x).y** .

Circular List: A linked list in which the rear item refers back to the head item

Class declaration file: A type of header file where the interface or declaration of a class is written. The code for the definition of the class is written in the class implementation file.

Class implementation file: A file where the definition of a class is written – in a very simple class this is sometimes done in a .h file – other times the interface or declaration for the class is written in a .h file and the class implementation file is written in a .cpp file.

Data Structure: The term data structure refers to the way data is organized for use within a program. Correct organization of data can lead to simpler and more efficient algorithms. Common data structures are linked-lists, stacks, queues and trees.

Doubly Linked List: A data structure in which each element contains pointers to the next and previous elements in the list, thus forming a bidirectional linear list.

Driver file or class: Generally an application which uses (drives) various other classes or programs.

FIFO: First in first out is a policy that items are processed in order of arrival. A queue implements this.

Head: The first item of a list.

Header file: A programmer defined file, usually with a .h extension, that is included in other files also containing c++ code.

Instance: A class is a definition of a set of data and member functions. When space for the data is actually allocated, we say that a member of the class has been instantiated. The instantiation is called an instance of the class. Each instance has its own set of data (there is also a mechanism in C++ to define data that is only allocated once per class, and shared amongst all instances of the class).

LIFO: Last in first out is a policy that the most recently arrived item is processed first. A stack implements this.

Linked List: A data structure in which a list of nodes or elements of a data structure connected by pointers. A singly linked list has one pointer in each node pointing to the next node in the list; a doubly linked list has two pointers in each node pointing to the next and previous nodes. In a circular list, the first and last nodes of the list are linked together.

Node: A unit of reference in a data structure. Also called a vertex in graphs and trees.

Object: Any program entity which uses physical memory in the computer

Object Oriented Programming: A concept of programming in which elements of the program are coded as stand-alone objects. Each object is completely self-contained in that it incorporates methods whereby the object can manipulate its own characteristics. A "Door" object, for instance would know how to open and close itself. It would also be able to respond to interrogation and advise the enquirer whether it is currently open or closed.

Overload: A term used to refer to the use of one symbol for more than one purpose. For instance, in mathematics the "-" symbol is used both as a negation symbol and as a subtraction symbol. In C++ the "<".

Queue: A data structure with first-in first-out behavior, supporting the operations enqueue (to insert) and dequeue (to remove)

Scope resolution operator: Sometimes called the resolution operator for short. It’s an operator, **::**, that is used to define a function outside a class from which it was declared. It can also be used with a global variable that has a local variable with same name. Thus the operator is used to define the scope of either data or functions – sometimes its use is required and sometimes it simply makes the code more clear.

Singly Linked List: A data structure in which a list of nodes or elements of a data structure connected by pointers. A singly linked list has one pointer in each node pointing to the next node in the list

Stack: A collection of items in which only the most recently added item may be removed. The latest added item is at the top. Basic operations are push and pop. Often top and isEmpty are available, too. Also known as "last-in, first-out" or LIFO.

Tail: The last item of a list.

Template: “… is a feature of the C++ programming language that allows functions and classes to operate with generic types. This allows a function or class to work on many different data types without being rewritten for each one. Templates are of great utility to programmers in C++, especially when combined with multiple inheritance and operator overloading.” – Wikipedia

Textfile: A file containing only printable text and white space. Textfiles may be given structure through the use of delimiters.

Vector: A data collection similar to arrays but can change their size dynamically. Like list classes, vectors generally have methods that facilitate the management of their data.

Unit 4 Trees

Ancestor: A parent ... of a node in a tree, the parent of the parent, etc.

AVL tree: A balanced binary search tree where the height of the two subtrees (children) of a node differs by at most one.

Balanced Binary Tree: A binary tree where no leaf is more than a certain amount farther from the root than any other. After inserting or deleting a node, the tree may rebalanced with "rotations."

Binary Search Tree: A data structure within which every node refers to a left subtree and a right subtree such that all values in the left subtree are smaller than the value in the node and all elements in the right subtree are greater than (or equal to) the value in the node. The top node is called the root. The nodes with no children (left and right subtrees empty) are called leaves.

Binary Tree: A specific type of tree data structure in which each node has at most two subtrees, one left and one right.

Child: An item or node of a tree referred to by a parent item. Every item, except the root, is the child of some parent.

Complete Binary Tree: A complete binary tree of depth d whose leaves are at level d

Degree: The degree of a node is the number of that node’s children.

Depth: The depth of a node is the length of the path from that node to the root. The depth of the root is 0.

Descendant: A child of a node in a tree, any of the children of the children, etc.

Forest: A forest is a set of 0 or more disjoint trees.

Full binary tree: A binary tree in which each node has exactly zero or two children.

Height: The height of a tree is the length of the longest path from the root to the furthest tree node.

Internal node: A node with at least one child. An internal node is any non-leaf node including the root if it is not a leaf.

In-order Traversal: Process all nodes of a tree by recursively processing the left subtree, then processing the root, and finally the right subtree.

Internal Node: A node of a tree that has one or more child nodes, equivalently, one that is not a leaf

Key: The part of a group of data by which it is sorted, indexed, cross referenced, etc.

Leaf: Any node (location) in a tree structure that is at the farthest distance from the root (primary node), no matter which path is followed. Thus, in any tree, a leaf is a node at the end of a branch—one that has no descendants.

Left rotation: In a binary search tree, pushing a node N down and to the left to balance the tree. N's right child replaces N, and the right child's left child becomes N's right child.

Level: A level on a tree is all the nodes sharing the same depth.

Level-order Traversal: Process all nodes of a tree by depth: first the root, then the children of the root, etc.

Perfect binary tree: A binary tree with all leaf nodes at the same depth. All internal nodes have degree 2.

Post-order traversal: Process all nodes of a tree by recursively processing the left subtree, then processing the right subtree, and finally the root.

Preorder traversal: Process all nodes of a tree by recursively processing the root, then processing the left subtree, and finally the right subtree.

Right rotation: In a binary search tree, pushing a node N down and to the right to balance the tree. N's left child replaces N, and the left child's right child becomes N's left child.

Root: The distinguished initial or fundamental item of a tree. The only item which has no parent.  The ancestor of all a tree’s nodes.

Rotation: To switch children and parents among two or three adjacent nodes to restore balance to a tree

Sibling: A node in a tree that has the same parent as another node is its sibling.

Subtree: A node and all its descendants.

Tree: A data structure containing zero or more nodes that are linked together in a hierarchical fashion

Tree Traversal: A technique for processing the nodes of a tree in some order.

Unit 5: Hash Tables

Binary File: A series of bytes in a file that can be accessed according to their data types without string processing and delimiters.

Chaining: A collision resolution scheme where the data is stored in a linked format such as in a linked list connected to each position in the table.

Clustering: A non-uniform distribution of data in a hash table where large segments of the table are filled while others are empty.

Collision: When two or more different keys hash to the same value or position in a hash table.

Collision Resolution Scheme: A way of handling collisions, that is, when two or more items should be kept in the same location, especially in a hash table. The general ways are keeping subsequent items within the table (open addressing), keeping a list for items which collide (direct chaining hashing or separate chaining hashing), keeping a special overflow area, etc. Perfect hashes avoid collisions, but may be time-consuming to create.

Hash Function: A formula that provides the home position in the hash table for data.

Hash Table: A dictionary in which keys are mapped to array positions by a hash function. It provides O(1) look-up for searches based on keys and may be used to implement associative arrays.

Home position: The position originally designated by the hash function for a particular data item.

Index File: A small file that contains search values and the location of the larger data – these files are often used to speed up non-primary key searches.

Linear Probing: A form of collision resolution that looks for the desired position by looking at the next adjacent position until the desired position has been found. This form of probing is generally not used because of clustering.

Load: An important density measure for a hash table that is computed by the number of data items over then number of positions in the hash table.

Open addressing: Any collision resolution scheme that places all the data in the hash table rather than relying on some way of storing some of the data outside the table. Probing strategies are a form of open addressing. Chaining is not a form of open addressing.

Perfect hash function: A hash function that places all the data in their home positions without any collisions.

Quadratic Probing: A collision resolution scheme designed to avoid clustering. Instead of probing simply to the next available position, quadratic probing checks first for the next position (1 2), but then checks for the 4 th position (2 2), then the 9 th positon (3 2), ...

Uniform distribution: A good hash function will provide a uniform distribution of the data within the hash table.

Unit 6: Heaps and Priority Queues

Binary Heap: A complete binary tree where every node has a key more extreme (greater for max heaps or less for min heaps) than or equal to the key of its parent.

Heap: Generally a binary tree where every level of the tree is filled except for the lowest level which is filled from left to right. Heaps are often used to implement a priority queue.

Heap sort: In a min or max heap, the progressive removal of nodes from the top of the heap will result in a sorted traversal of the data.

Max-heap property: Each node in a tree has a key which is less than or equal to the key of its parent.

Order property : Each node in the heap has an equal or more important in relation to it’s descendants.

Perculate down: When an item is removed from a priority queue, the item in the last position of the heap is placed at the root, and then it switches positions with its appropriate child until the heap order position is fulfilled.

Perculate up: When an item is added to a priority queue implemented as a heap, it is added to the next available position in the heap, and then it switches positions with its parent until the heap order property is fulfilled.

Priority queue: A priority queue is a data structure in which added items are dequeued or released from the collection according to their importance or priority.

Structure property: a heap has all levels filled except possibly the last one (deepest) this is filled fully from left to right.

Unit 7: Graphs

Acyclic graph: A graph without cycles

Adjacency matrix: an implementation for a graph which uses a two dimensional array where the vertices of the array provide the indices that define a value for each edge between the vertices.

Adjacency lists: an implementation for a graph which uses a list of vertices where each vertex also has a list of its edges.

Adjacent: Two nodes are adjacent if they are connected through an edge

Arc: An edge of a directed graph

Bridge: An edge whose removal would create a disconnected graph

Complete: A complete graph is one in which every vertex shares an edge with every other vertex in the graph

Cycle: A walk or tour that begins at one vertex and returns to that same vertex without repeating any vertex in the walk.

Degree of a vertex: The degree of vertex is equal to the number of its edges.

Dense graph: a graph where the number of edges approaches the number of vertices (not a rigid definition and often depends on context). In a dense graph, E may approach V2.

Digraph or directed graph: A graph where the arcs (edges) have a specific direction.

Disjoint Set: A set whose members do not overlap.

Edge: A connection between two vertices.

Finite graph: A graph with a finite number of edges and vertices.

Hypergraph is a graph in which an edge can join any number of vertices.

Infinite graph: A graph with an infinite number of edges and/or vertices.

Isolated vertex: Refers to vertex without any edges.

Leaf: A vertex with a degree of 1.

Loop: An edge whose source and destination is the same vertex.

Matrix: A two-dimensional array. By convention, the first index is the row, and the second index is the column.

Minimum spanning tree: A subgraph without cycles that includes all the graphs vertices and that has the least cost among all spanning trees

Order: The order of a graph is the number of its vertices.

Path: A sequence of edges originating at one vertex and leading to a destination vertex.

Peek: A stack function whereby the value of the top is retrieved without popping the stack.

Ragged matrix: A matrix having irregular numbers of items in each row.

Ray: A path originating at one vertex within an infinite graph

Rectangular matrix: An n × m matrix, or, one whose size may not be the same in both dimensions.

Sink: A vertex in a digraph that only has incoming arcs.

Size: The size of a graph is the number of its edges.

Source: A vertex in a digraph that only has outgoing arcs.

Spanning tree: A subgraph without cycles that includes all the graphs vertices. In a spanning tree, E always equals V - 1.

Sparse graph: A graph with only a “few” edges for most vertices (“few” may depend on context). In a very sparse graph E approaches V.

Standard Template Library (STL): A set of c++ template classes to provide common data structures and functions. These classes included vector, list, queue, stack, set, priority\_queue, arrays, map, and set; all of which can be useful when one implements graphs and graph algorithms.

Subgraph: A subset of a graph’s edges and vertices.

Tree: In the context of graph theory, a tree is an acyclic graph,

Undirected: A graph is undirected if all its edges are bi-directional.

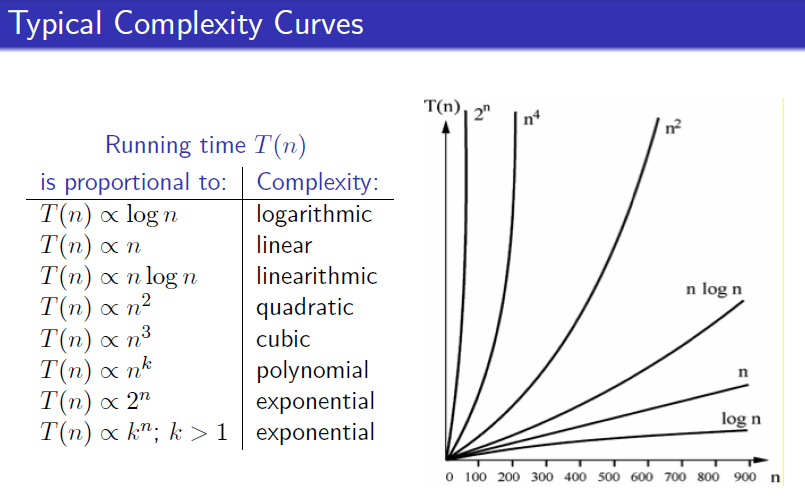
Union: The union of two sets is a set having all members in either set.

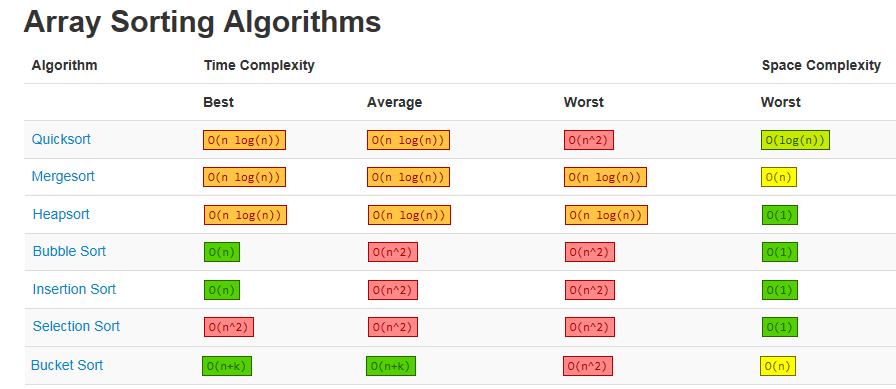
Unweighted: A graph where all the edges have the same cost.

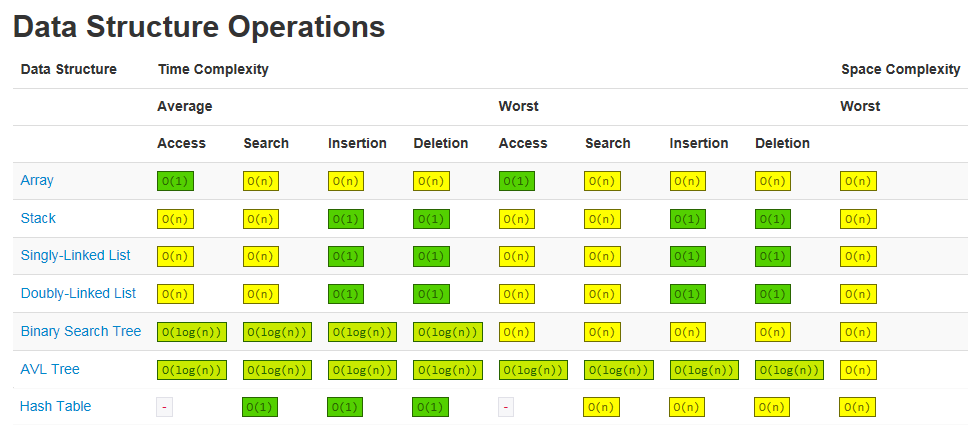
Vertex: A data item or node in a graph.

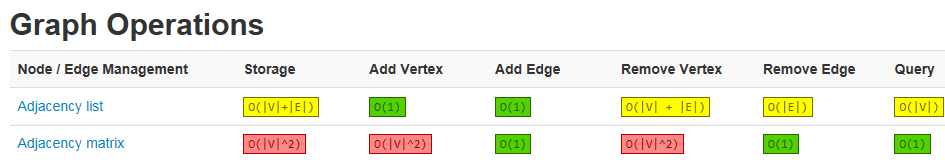
Weight: The cost associated with processing or traversing an edge.

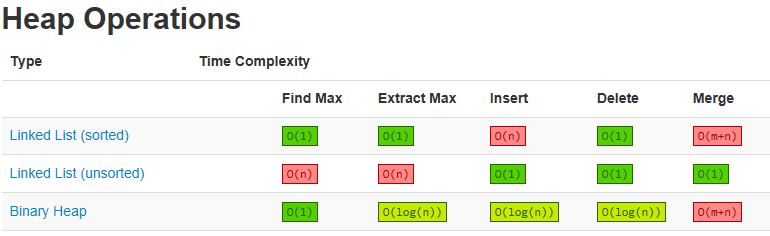
**Complexity** (very nice resource: <http://bigocheatsheet.com/> )











**Part 2: Exercises**

1. What is the Big O complexity for each of the following sorts (average case)?

Selection sort \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Insertion sort \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Quick sort \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Bubble sort \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Merge sort \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Describe the purpose of the find function in the code below. bookTitle is a string array.

string output = "";

for (int i = 0; i < 3; i++)

{

if (bookTitle[i].find(searchString) != string::npos)

{ output += "\"" + searchString + "\" is found in " + bookTitle[i]

+ "\n";

}

else

{

output += "\"" + searchString + "\" is not found in " + bookTitle[i]

+ "\n";

}

}

cout << output << endl;

1. Show or represent the output of this code.

int int1;

int \*pointer1;

pointer1 = &int1;

\*pointer1 = 42;

cout << "Demo 1:" << endl;

cout << "int1 = " << int1 << endl;

cout << "&int1 = " << &int1 << endl;

cout << "\*&int1 = " << \*&int1 << endl;

cout << "pointer1 = " << pointer1 << endl;

cout << "\*pointer1= " << \*pointer1 << endl;

int number = 11;

int \*digit = &number;

\*digit = number + 7 + \*digit;

cout << "number = " << number << endl;

cout << "digit = " << digit << endl;

cout << "\*digit = " << \*digit << endl;

1. Fill in the blanks for (a) merge sort and (b) quick sort.

(a)

void merge\_sort(int array[], int low, int high)

{

int mid;

if ( \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ < \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

{

mid = (\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ + \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_) / \_\_\_\_\_;

merge\_sort(\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ , \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ );

merge\_sort(\_\_\_\_\_\_\_\_\_\_\_\_\_, \_\_\_\_\_\_\_\_\_\_\_\_\_\_ + \_\_\_\_\_\_\_\_\_\_\_\_\_ , \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ );

merge(\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, mid,\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_);

}

}

(b)

void quickSort(int array[], int left, int right)

{

if ( \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ < \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

{

int p = partition(array, left, right);

quickSort(\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_,\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_,\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_);

quickSort(\_\_\_\_\_\_\_\_\_\_\_\_\_\_,\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_,\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_);

}

}

1. Complete the following function to display the square root of each element of the array. Use the sqrt function of the cmath include.

void displaySquareRoots(int array[], int length)

{

1. Write a code segment that swaps the values of an integer array called myArray at index i and index j.
2. In a singly-linked list, with a node pointer called head, code the body of the isEmpty function.
   1. return ( (head == null) && (tail == null) );
   2. return (head == NULL);
   3. return (head != null)
   4. return ( (head == NULL) || (tail == NULL) )
3. A node’s data attributes consist of a(n) \_\_\_\_\_\_\_\_\_\_\_\_\_\_ and a(n) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.
   1. data element pointer
   2. string pointer
   3. head tail
   4. head data element
4. A member function whose name begins with a tilde (~) is called a(n) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.
   1. constructor
   2. node
   3. destructor
   4. linker
   5. include
5. Fill in the blanks for this class declaration (see Book.h).

#include <iostream> // Used for input and output.

#include <string>

using namespace std;

// Preprocessing directives that prevents multiple definitions.

#ifndef \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#define \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

// Class declaration for Book class.

class Book

{

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ostream &operator << (ostream& out, const Book& theBook);

/////////////////////////////////////

public:

// Default constructor.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

// Other constructor.

\_\_\_\_\_\_\_\_\_\_\_\_\_(\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_);

// Declare get and set member functions.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

// Utility member functions.

void print();

/////////////////////////////////////////

// Member attributes (generally private).

private:

string title;

string author;

int year;

};

#\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Fill in the following implementation of the Book class in Book.cpp.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ "Book.h"

// Default constructor.

Book::Book()

{

setTitle(\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_);

setAuthor(\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_);

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(0);

}

// Another constructor. This is an example of overloading.

Book::Book(string theTitle, string theAuthor, int theYear)

{

setTitle(\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_);

setAuthor(\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_);

setYear(\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_);

}

///////////////////////

// Get and set methods.

string \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_::getTitle() const

{

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ title;

}

string Book \_\_\_\_ getAuthor() const

{

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ author;

}

int Book::getYear() const

{

return \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

}

void Book::setTitle(string theTitle)

{

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

}

void Book::setAuthor(string theAuthor)

{

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

}

// Control data and flag bad data with default value.

void Book::setYear(int theYear)

{

if (theYear < 0)

{

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ = 0;

}

else

{

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ = theYear;

}

}

// Utility member functions.

void Book::print()

{

cout << title << " by " << author << " published in " << year;

}

ostream &operator << (ostream& out, const Book& theBook)

{

out << \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ << " by "

<< theBook.author << " published in " << \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

return \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

}

1. Complete the code in the singly linked list’s default constructor and destructor. [2 points]

SinglyLinkedList::SinglyLinkedList()

: head(\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_) { }

SinglyLinkedList::~SinglyLinkedList()

{

while (\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

{

removeFront();

}

}

1. Complete this function that returns the book of the first node of the list. Use meaningful identifiers.

const Book& SinglyLinkedList::front() const

{

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

}

1. Complete the isEmpty function below. Use meaningful identifiers.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ SinglyLinkedList::empty() \_\_\_\_\_\_\_\_\_\_\_\_\_\_

{

return \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

}

1. Fill in the blanks for the addFront member function. Use meaningful identifiers.

void SinglyLinkedList::addFront(const Book& e)

{

Node\* temp = \_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_->book = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_->next = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

head = temp;

}

1. Fill in the blanks for the removeFront member function.

void SinglyLinkedList::removeFront()

{

if ( !\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ )

{

Node\* temp = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

head = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_->next;

delete \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

}

}

1. Complete the following global declarations for the bookList that holds objects of the Book class and for the songList that holds objects of the Song class. bookList and songList are of the TemplateLinkedList template class.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ TemplateLinkedList<\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_> bookList;

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_<\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_> songList;

1. What are two principle advantages in using a language provided collection class such as the c++ Vector class instead of an array?
2. Fill in the blanks for this implementation of a binary search tree (adding).

//Definition of Node for Binary search tree

struct BstNode

{

int data;

BstNode\* left;

BstNode\* right;

};

// Function to create a new Node in tree

BstNode\* GetNewNode(int theData)

{

BstNode\* newNode = new BstNode();

newNode->data = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

newNode->\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ = newNode-> \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

return \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

}

// To insert data in BST, returns address of root node.

BstNode\* Insert(BstNode\* root,int theData)

{

if(root == NULL)

{ // Empty tree or sub-tree.

root = GetNewNode(theData);

}

// If data to be inserted is lesser ...

else if(theData <= root->data)

{

root->left = Insert(root->\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_);

}

// Else theData is greater.

else

{

root->right = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

}

return \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_; // Returns new added node (address).

}

1. Fill in the blanks for this getDepth function in a first-child next-sibling tree (with parent pointer as well). Use meaningful identifiers for blanks with unknown variable names.

int Node::getDepth()

{

int theDepth = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

Node \*node = this;

while ( ! \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_->isRoot() )

{

node = node->\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

}

return theDepth;

}

1. Fill in the blanks for these display functions for this first-child next-sibling tree implementation of an organization tree of employees with the president being the root. Use meaningful identifiers for blanks with unknown variable names.

void Organization::displayTraversal(Node\* node)

{

if (node)

{

cout << \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_->name << endl;

displayTraversal(node->\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_);

displayTraversal(node->\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_);

}

}

void Organization::display()

{

if (\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

{

displayTraversal(president);

}

}

1. Fill in the blanks for writing Book objects to a binary file.

void writeBinary(int position, Book theBook)

{

hashFile.seekp( \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\* sizeof(\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_) );

hashFile.write( (char \*) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, sizeof(\_\_\_\_\_\_\_\_\_\_\_\_\_\_) );

hashFile.flush();

}

1. Fill in the blanks for reading Book objects from a binary file.

void readBinary(int position, Book& theBook)

{

hashFile.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \* sizeof(\_\_\_\_\_\_\_\_\_\_\_) );

hashFile.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ( (char \*) \_\_\_\_\_\_\_\_\_\_\_\_\_\_, sizeof(\_\_\_\_\_\_\_\_) );

}

1. Explain how you would write a hash function. What are some important characteristics of a good hash function?

int hashFunction(int key)

{

int position = key % FILE\_LENGTH;

return position;

}

1. Fill in the blanks for this linear probing functions for a hash table for book objects (using previous readBinary function.

bool addable(int position)

{

Book book;

readBinary(\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_);

return ( book.getISBN() == EMPTY );

}

int linearProbe(int position)

{

do // Increment position until an addable position is reached.

{

if (position < \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ )

{

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

}

else // If the end of the file is reached

{ // restart in the first position.

position = 0;

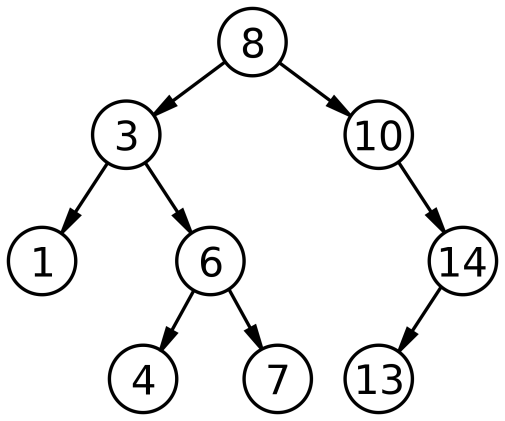
}

} while ( !addable(position) );

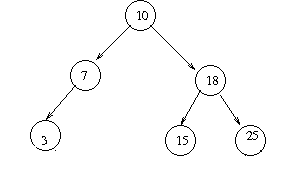
return \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

}

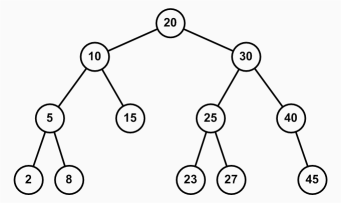
1. Add a node with the key of 5 to this binary search tree.

[](http://www.google.com/url?sa=i&rct=j&q=binary+search+tree&source=images&cd=&cad=rja&uact=8&ved=0ahUKEwi5m_iWvLvJAhVMRiYKHdviBcEQjRwIBw&url=https://en.wikipedia.org/wiki/Binary_search_tree&psig=AFQjCNHUn6SEGasZJZT7iu7IpzRGVvkcmw&ust=1449087048960093)

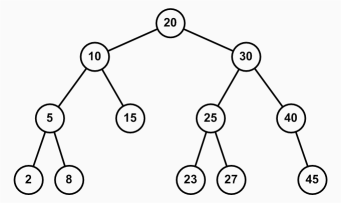
1. Add a node with a key of 1 to this binary search tree. Rotate the nodes to keep the tree balanced.



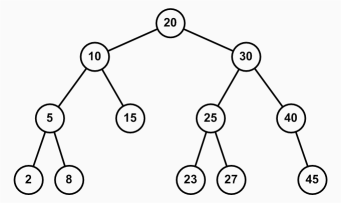
1. In this BST, Remove the node with the key of 20 showing the transformation and the end result.

[](http://www.google.com/url?sa=i&rct=j&q=binary+search+tree&source=images&cd=&cad=rja&uact=8&ved=0ahUKEwjr_7mCv7vJAhXBOCYKHZlxA8IQjRwIBw&url=http://www.jade-cheng.com/uh/ta/ics-211-spring-2010/faq/&psig=AFQjCNHUn6SEGasZJZT7iu7IpzRGVvkcmw&ust=1449087048960093)

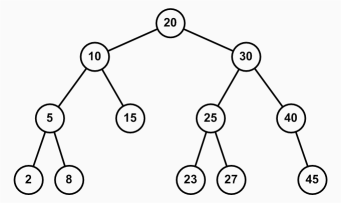
1. In this BST, remove the node with the key of 30 showing the transformation and the end result.

[](http://www.google.com/url?sa=i&rct=j&q=binary+search+tree&source=images&cd=&cad=rja&uact=8&ved=0ahUKEwjr_7mCv7vJAhXBOCYKHZlxA8IQjRwIBw&url=http://www.jade-cheng.com/uh/ta/ics-211-spring-2010/faq/&psig=AFQjCNHUn6SEGasZJZT7iu7IpzRGVvkcmw&ust=1449087048960093)

1. In this BST, add a node of 48 and rotate.

[](http://www.google.com/url?sa=i&rct=j&q=binary+search+tree&source=images&cd=&cad=rja&uact=8&ved=0ahUKEwjr_7mCv7vJAhXBOCYKHZlxA8IQjRwIBw&url=http://www.jade-cheng.com/uh/ta/ics-211-spring-2010/faq/&psig=AFQjCNHUn6SEGasZJZT7iu7IpzRGVvkcmw&ust=1449087048960093)

1. Redraw this tree as a first child next sibling tree.

[](http://www.google.com/url?sa=i&rct=j&q=binary+search+tree&source=images&cd=&cad=rja&uact=8&ved=0ahUKEwjr_7mCv7vJAhXBOCYKHZlxA8IQjRwIBw&url=http://www.jade-cheng.com/uh/ta/ics-211-spring-2010/faq/&psig=AFQjCNHUn6SEGasZJZT7iu7IpzRGVvkcmw&ust=1449087048960093)

1. Here’s some code for an in order traversal for a binary tree. Show code for post-order and pre-order traversals.

void inOrder(Node\* node)

{

if (node == NULL)

return;

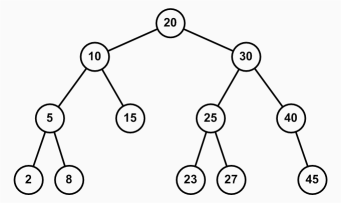
inOrder(node->left);

cout << node->data;

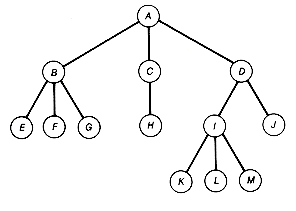
inOrder(node->right);

}

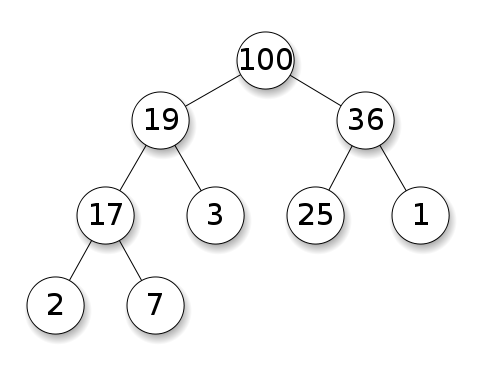
1. Give the pre-order, post-order, and in-order traversals of this binary search tree.

[](http://www.google.com/url?sa=i&rct=j&q=binary+search+tree&source=images&cd=&cad=rja&uact=8&ved=0ahUKEwjr_7mCv7vJAhXBOCYKHZlxA8IQjRwIBw&url=http://www.jade-cheng.com/uh/ta/ics-211-spring-2010/faq/&psig=AFQjCNHUn6SEGasZJZT7iu7IpzRGVvkcmw&ust=1449087048960093)

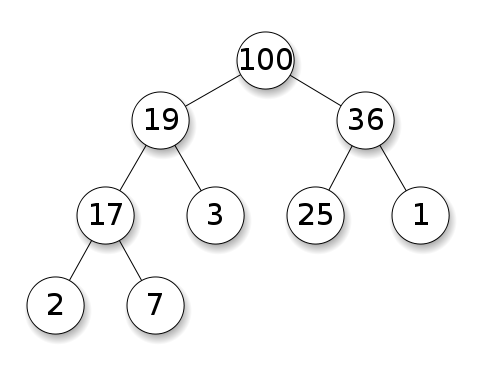
1. Explain how would you use a queue to create a breadth-first traversal of the above tree.
2. Would Dijkstra’s shortest path algorithm work on a min heap? Would it work on an AVL tree? Why or why not for each?
3. Redraw this tree as a first child next sibling tree.

[](http://www.google.com/url?sa=i&rct=j&q=general+tree&source=images&cd=&cad=rja&uact=8&ved=0ahUKEwiH85-OwLvJAhVBNSYKHS4TAcEQjRwIBw&url=http://mindfulintegrations.com/books/Technology/computer_science/algo/books/book2/algo02b5.htm&bvm=bv.108194040,d.eWE&psig=AFQjCNGmfGzDa38IJYPA91eL-gm7wwIF-w&ust=1449088114285671)

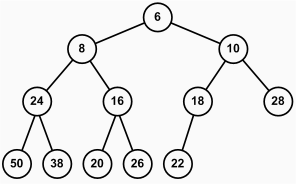
1. Is this a valid max heap? OK, now let’s add a 49 and then add a 31. Show end result.

[](http://www.google.com/url?sa=i&rct=j&q=max+heap&source=images&cd=&cad=rja&uact=8&ved=0ahUKEwjkh7SlwLvJAhWHWCYKHV9pD8IQjRwIBw&url=http://cs.stackexchange.com/questions/27860/whats-the-difference-between-a-binary-search-tree-and-a-binary-heap&psig=AFQjCNFppbuQ1jCwoesytvhxO-RnwomD_Q&ust=1449088166445363)

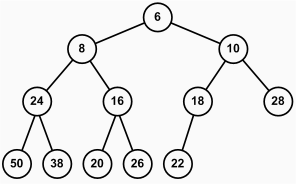
1. Remove the 100. Show work and end result.

[](http://www.google.com/url?sa=i&rct=j&q=max+heap&source=images&cd=&cad=rja&uact=8&ved=0ahUKEwjkh7SlwLvJAhWHWCYKHV9pD8IQjRwIBw&url=http://cs.stackexchange.com/questions/27860/whats-the-difference-between-a-binary-search-tree-and-a-binary-heap&psig=AFQjCNFppbuQ1jCwoesytvhxO-RnwomD_Q&ust=1449088166445363)

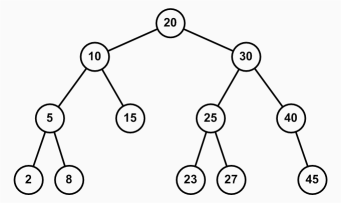
1. What kind of heap is the following? Remove the 6. Show work and end result.

[](http://www.google.com/url?sa=i&rct=j&q=min+heap&source=images&cd=&cad=rja&uact=8&ved=0ahUKEwj1tdX4wbvJAhWKTSYKHeMEBXEQjRwIBw&url=http://www.jade-cheng.com/uh/ta/ics-211-spring-2010/faq/&psig=AFQjCNGR6fxTJcEWlxtfVWj-q5K9rnrR0Q&ust=1449088591169764)

1. Remove a node from this priority queue.

[](http://www.google.com/url?sa=i&rct=j&q=min+heap&source=images&cd=&cad=rja&uact=8&ved=0ahUKEwj1tdX4wbvJAhWKTSYKHeMEBXEQjRwIBw&url=http://www.jade-cheng.com/uh/ta/ics-211-spring-2010/faq/&psig=AFQjCNGR6fxTJcEWlxtfVWj-q5K9rnrR0Q&ust=1449088591169764)

1. Show how the following keys would be stored in an array.

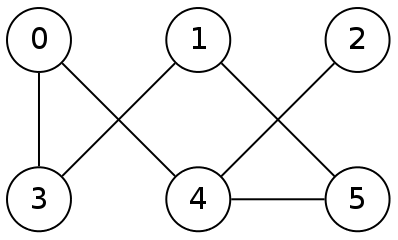
[](http://www.google.com/url?sa=i&rct=j&q=binary+search+tree&source=images&cd=&cad=rja&uact=8&ved=0ahUKEwjr_7mCv7vJAhXBOCYKHZlxA8IQjRwIBw&url=http://www.jade-cheng.com/uh/ta/ics-211-spring-2010/faq/&psig=AFQjCNHUn6SEGasZJZT7iu7IpzRGVvkcmw&ust=1449087048960093)

Also show formulas for the following indices from a current node *i*:

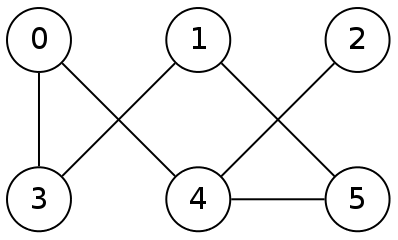
Left child index =

Right child index =

Parent index =

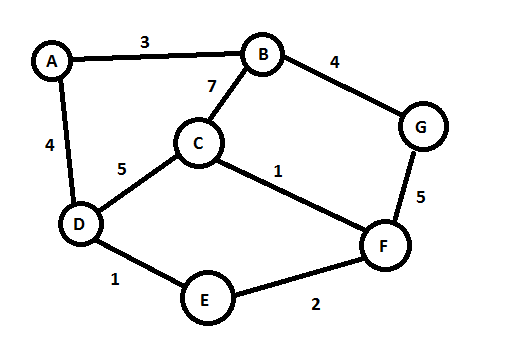
1. Trace the depth-first traversal beginning with vertex 1. Use the chart below to show your work as we did in class.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Action (1)** | **Action (2)** | **Stack** | **Unvisited Vertices** | **Visited Vertices** |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

1.  Trace the breadth-first traversal beginning with vertex 3. Use the chart below to show your work as we did in class.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Dequeue** | **Enqueue** | **Queue** | **Unvisited Vertices** | **Visited Vertices** |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

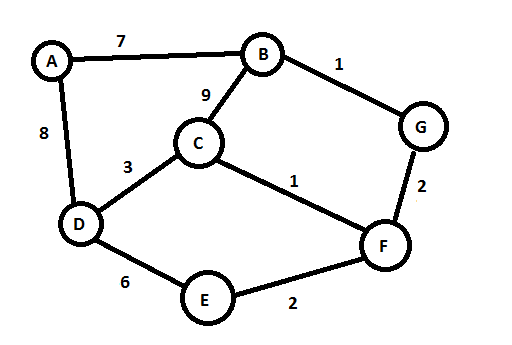
1. Trace Dijkstra’s Shortest Path Algorithm using the following chart with A as the starting vertex.



|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Selected Vertex |  | Destination Vertex | | | | | | |
|  | A | B | C | D | E | F | G |
| Initial | **0-A** |  |  |  |  |  |  |
| A (start vertex) |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Shortest Path From A | B | C | D | E | F | G |
| Cost |  |  |  |  |  |  |
| Through vertex |  |  |  |  |  |  |
| Path |  |  |  |  |  |  |

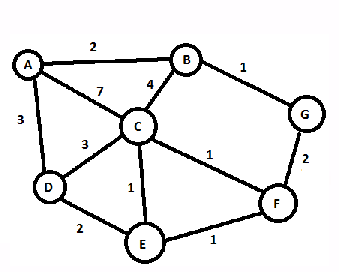
1. Use Prim’s algorithm to construct a minimum spanning tree. Show your work in the charts provided. Draw the minimum spanning tree.



|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Sorted Edges | T1 | T2 | T3 | T4 | T5 | T6 |
| Tree: | | | | | |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

Draw Minimum Spanning Tree:

1. Use Kruskal's algorithm to construct a minimum spanning tree. Show your work in the charts provided. Draw the minimum spanning tree.



|  |  |  |  |
| --- | --- | --- | --- |
| Sorted Edges | Same  Set? | Action | Result |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Draw Minimum Spanning Tree:

1. List the pros and cons of an adjacency matrix versus an adjacency list representations for a graph.

|  |  |  |
| --- | --- | --- |
|  | Pros | Cons |
| Adjacency matrix | * Good for dense graphs with stable vertices * Search for edge weight is O(1) * Easy to implement and understand | * Storage is O(n2) * Adding and deleting vertices is expensive O(n2) because of shifting rows and columns in the arrays |
| Adjacency lists | * Good for sparse graphs (which tend to be the most common) * Storage is O(V + E) * Adding and deleting vertices is O(V + E) (adding a vertex could be O(1) if vertices are unsorted) – better for graphs with vertices that need to be added and deleted often * Often quicker with algorithms using large number of vertices because the number of edges per vertex may be much smaller | * Search at unknown vertex is O(V + E) at known vertex is O(E) |

1. What data structures might a graph application use besides linked lists and 2D matrix?
2. List some common straight-forward applications for the following data structures: array, stack, queue, priority queue, hash table, tree, and graph.

Array/vector – 3D tic-tac-toe (3D array), chess board, game of battleship

Stack – collection of boxes of stock in a grocery store, a stack of items in a shipping container, system stack, pancakes, general graph implementation for the sets of edges and vertices

Queue – teller line at a bank, a collection of cars waiting to get on a ferry, printer queue, multi-tasking processing, cafeteria line

Linked list – general graph implementation for the sets of edges and vertices, changes in an IDE, ...

Tree – tournament brackets, a decision tree on purchasing the best computer, biological tree of life

Hash table – dictionary, student directory, collection of books or students

Heap – a priority queue such as a collection of items people that prioritizes the age of older people or for bills to be paid

Graph – neuron mapping, social network, computer network, GIS

1. Identify whether each algorithm is best implemented iteratively or recursively and briefly describe your rationale (no preference option possible too).

Quick sort

Merge sort

Bubble sort

Binary search

Adding to a sorted linked list

Pre-order binary tree traversal

Removing from a binary search tree

Removing from a heap implementation of a priority queue

Breadth-first traversal of an AVL tree

Dijkstra’s shortest path

1. Explain how a binary search tree or binary heap can be implemented as an array. What formulas can be used to link parent and child nodes? What is the principle limitation of the array implementation? What are the principle advantages of a linked implementation?
2. How does the balance or lack of balance affect the the worst, best, and average case scenarios for adding and deleting from a binary search tree by specifically list the worst, best, and average case Big Notation for each.