**CS315 Practice Questions for Midterm 2 Questions about bigO efficiency (sorting an array).**

1. What is the bigO efficiency of Merge Sort?

O(nlog(n))

1. What is the bigO efficiency of Quicksort?

O(nlog(n))

1. What is the bigO efficiency of Insertion Sort?

O(n2)

1. What is the bigO efficiency of Selection Sort?

O(n2)

1. What is the bigO efficiency of Bubble Sort?

O(n2)

1. What is the bigO efficiency of Radix Sort?

O(n)

1. Which three of the above sorting algorithms cannot be performed “in place” and must use space of a second array?

Mergesort, Radix Sort, Quicksort

**Questions about bigO efficiency (operations on data structures).**

1. What is the bigO efficiency of adding an element to the end of an array?

O(1)

1. What is the bigO efficiency of adding an element to the front of an array?

O(n)

1. What is the bigO efficiency of searching for a value in an unsorted array?

O(n)

1. What is the bigO efficiency of searching for a value in a sorted array using binary search?

O(log(n))

1. What is the bigO efficiency of searching for a value in a linked list?

O(n)

1. What is the bigO efficiency of searching for a value in a balanced binary search tree?

O(log(n))

1. What is the bigO efficiency of adding a link to the tail of a doubly linked list?

O(1)

1. What is the bigO efficiency of removing a value from an ordered array?

O(n)

1. What is the bigO efficiency of removing a node from a doubly linked list?

O(1)

**Questions about tree traversal.**

Each of the following questions refer to the following tree.



1. Print the values in the above tree as per pre-order depth-first traversal.

root, left, right

Root left right

4,2,3,1,6,8

1. Print the values in the above tree as per in-order depth-first traversal.

left root right

1, 2, 3, 4, 6, 8

1. Print the values in the following tree as per post-order depth-first traversal.

Left right root

1,3,2,6,8,4

1. Print the values in the following tree as per breadth-first traversal.

Top-Down, left to right

4, 2, 6, 1, 3, 8

**Questions about array sorting algorithm operations.**

Each of the following 6 questions refer to the following list of values in the order as shown.

555 288 633 666 444 422 399 477

1. Print the values in the order they would have after the first round of bubble sort’s bubbling operation low to high.

288 555 633 444 422 399 477 666

1. Print the values in the order they would have after the first round of selection sort for sorting low to high, assuming it selects the max.

555 277 633 477 444 422 399 666

1. Print the values in the order they would have after the first round of selection sort for sorting low to high, assuming it selects the min.

288 555 633 666 444 422 399 477

1. Print the values in the order they would have after the first round of a binary radix sort for sorting low to high.

288, 666, 444, 422, 555, 633, 399, 477

1. Print the values in the order they would have after the first round of a decimal radix sort for sorting low to high.

422 633 444 555 666 477 288 399

1. Print the values in the order they would have after the first round of quicksort’s 3-way partition operation. Assume it chooses the last element in the array as the pivot.

288 444 422 399 477 666 633 555

1. Theoretically (if it could be done), what would be the best value to use as the pivot for a quicksort round of partitioning?

288 399 422 444 477 555 633 666

444 or 477 would be the best value

**Questions about stacks and queues.**

1. After the operations push(1), push(2) and push(3), what value does pop() return?

3

1. After the operations enqueue(1), enqueue(2), and enqueue(3), what value does dequeue() return?

1

**Coding questions (array):**

**//**private helper function, Van hilst says we dont need to error check, but I would //anyway

private boolean validIndex (const int[] array, int index) {

return (index >= 0 && index < array.length);

}

1. Write a swap function that takes three arguments for the array and the two indexes to have their values swapped.

//note: you dont need to make this a boolean, and most people don’t. It’s just my personal //preference

public boolean swap(int[] array, int firstIndex, int secondIndex) {

if (validIndex(array, firstIndex) && validIndex(array,secondIndex)) { //not needed

int temp = array[firstIndex]; //temp variable to hold one of the values

array[firstIndex] = array[secondIndex]; //swap clone second into first

array[secondIndex] = temp; // set second to first

return true;

} else {

return false; //invalid inputs

}

}

For the following questions, assume the existence of a swap function like the above that takes three arguments for the array, and the two indexes.

1. Write a function for performing for performing one round of bubbling, low to high. The function should be named bubble, and take three arguments for the array, the first index and the last index.

void bubble (int array[], int firstIndex, int secondIndex) {

if ( validIndex(array, firstIndex) && validIndex(array, secondIndex)) { //validity check

for(int i = firstIndex; i < secondIndex; i++) { // go through the array once

if (array[i] > array[i+1]) { //check to see which index is greater

swap(array, i, i+1); // swap if the first one is

}

}

}

}

1. Write a function that replaces the first value in the array with the minimum value in the array. The function should be named selectMin and take three arguments for the array, the first index, and the last index.

void selectMin(int[] array, int firstIndex, int lastIndex) {

if(validIndex(array, firstIndex) && validIndex(array, secondIndex)) { //validity check

int minIndex = firstIndex; //start assuming the min is in the first spot

for(int i = firstIndex + 1; i <= lastIndex; i++){ //remember to to <= for the lastIndex

if (array[i] < array[min]){ //change the min location if needed

min = i;

}

}

swap(array,firstIndex,min); //swap the first index and the min

}

}

}

**(single linked list):**

The following questions assume that the above code for a LinkList class with singly linked Node’s already exists.

**public** **class** Node { **public** **int** value; **public** Node next;

**public** Node(**int** value) { **this**.value = value; next = **null**;

}

} **public** **class** LinkList { **private** Node head;

**public** LinkList() { head = **null**;

}

}

1. Write a method for the LinkList class to insert a Node at the head of the singly linked list. The function should be called insertHeadNode and take a Node element as the argument.

public void insertAsHead(node newHead) {

newHead.next = head; //set the newHead’s next to head

head = newHead; //head becomes newHead

}

}

1. Write a method for the LinkList class to insert a Node at the head of a singly linked list. The function should be called insertHeadValue and take an integer value as the argument.

//theres a few diffrent implementations here, pick the one that makes the most sense to you //and run with it

public void insertAsHead(int value) {

Node newHead = new Node(value);

if (head == null) {

newHead = head;

} else {

newHead.next = head;

head = newHead;

}

}

public void insertHeadValue(int value) {

Node node = new Node(value);

if(this.head != null){ //not list

node.next = head; //node's next is the old head

}

head = node; //set node as the new head

}

//another alternative, works based on the fact that we dont need a special case for inserting in //an empty singlelink list

public void insertHeadValue(int value) {

Node newHead = new Node(value);

newHead.next = head;

head = newHead;

}

1. Write a method for the LinkList class to remove the Node at the head of the singly linked list. The function should be called removeHeadValue and return the integer value of the Node just removed (or 0 if the list is empty).

int removeHeadValue() {

if (head != null) {

int headValue = head.value;

head = head.next;

return headValue;

} else {

return 0;

}

}

1. Write a method for the LinkList class to return the current length of the list. The function should be called getLength and should return an int.

public int getLength() {

int length = 0;

Node current = head;

while (current != null) { //while loop because we dont know when we are going to stop

length++;

current = current.next;

}

return length;

}

**(double linked list):**

The following questions assume that the above code for a DblLinkList class with doubly linked Node’s already exists.

**public** **class** DblLinkList { **private** Node head; **private** Node tail;

**private** **class** Node { **private** **int** value; **private** Node next; **private** Node previous;

**protected** Node(**int** value) { **this**.value = value; next = **null**; previous = **null**;

}

} **public** DblLinkList() { head = **null**; tail = **null**;

}

}

1. Write a method for the DblLinkList class to insert a Node at the tail of the doubly linked list. The function should be called insertTailValue and take an integer value as the argument.

public void insertTailValue(int value) {

Node newTail = new Node(value);

if (tail == null) { //empty list case, so head and tail must both be set

head = newTail;

tail = newTail;

}

else {

tail.next = newTail;

newTail.previous = tail;

tail = newTail;

}

}

1. Write a method for the DblLinkList class to remove the Node at the tail of the doubly linked list. The function should be called removeTailValue and return the integer value of the Node just removed (or 0 if the list is empty).

public it removeTailValue {

int tailValue = 0;

if (tail != null) {

tailValue = tail.value;

tail = tail.previous;

if(tail != null) {

tail.next = null;

}

else { //deals with deleting the last item in the list

head = null;

}

}

return tailValue;

}

//alternative

public int removeTailValue(){

if(tail==null)

return 0;

else{

int tailValue=tail.value;

tail=tail.previous;

if(tail != null) {

tail.next = null;

}

else {

head = null;

}

return tailValue;

}

}

1. Write an isEmpty() method that returns a Boolean value of false if the list is not empty.

public boolean isEmpty() {

return head == null;

}

public boolean isEmpty() {

if (head == null) {

return true;

}

return false;

}

**(iterators):**

**public** **abstract** **class** GoFIterator { **public** **abstract** **boolean** isDone(); **public** **abstract** **void** first(); **public** **abstract** **void** next(); **public** **abstract** **int** currentItem();

}

**public** **abstract** **class** JavaIterator { **public** **abstract** **boolean** hasNext(); **public** **abstract** **int** next();

}

//Theres really no good way for me to explain these without making //this document about 10 pages longer. Just memorize them

1. Assume that you have a data structure instance called myList, for a data structure that has a method called getIterator that returns an instance of the above GoFIterator. Write the code to print all of the values in myList, one above the other.

GoFInterator iter = myList.getIterator();

for(iter.first(); !iter.isDone(); iter.next()) {

System.out.println(iter.currentItem());

}

1. Assume that you have a data structure instance called myList, for a data structure that has a method called getIterator that returns an instance of the above JavaIterator. Write the code to print all of the values in myList, one above the other.

JavaIterator iter = myList.getIterator();

while(iter.hasNext()) {

System.out.println(iter.next();

}

**Coding questions (tree):**

The following questions assume that the above code for a binary search Tree already exists.

**public** **class** Tree { **private** Node root;

**private** **class** Node { **private** **int** value; **private** Node leftChild; **private** Node rightChild;

**protected** Node(**int** value) { **this**.value = value; leftChild = **null**; rightChild = **null**;

}

}

**public** Tree() { root = **null**;

}

}

1. Write a getMin() method that returns the minimum value in the tree (or 0 if the tree is empty). The method should use iterative code for find and return the min.

public int getMin() {

Node current = root;

if (current == null) {

return 0;

}

while(current.leftChild != null) {

current = current.leftChild;

}

return current.value;

}

//alternative

public int getMin() {

if(root != null) {

node current = root;

while (current != null && current.leftChild != null) {

current = current.leftChild;

}

return current.value;

}

else {

return 0;

}

}

1. Write a getMax() method that returns the maximum value in the tree (or 0 if the tree is empty). The method should use a recursive helper function to find and return the max.

public int getMax(){

if(root == null){

return 0;

}

else {

return getMax(root);

}

}

private int getMax(Node root){

if(root.rightChild == null){

return root.value;

}

else {

return getMax(root.rightChild);

}

}

1. Write a hasValue() method that takes an integer value and returns true or false depending on whether the value argument can be found in the tree. Use iterative code for the search.

public boolean hasValue(int value) {

Node current = root;

while(current != null) {

if(current.value == value) {

return true;

}

else if(current.value < value) {

current = current.rightChild;

}

else if (current.value > value) {

current = current.leftChild;

}

}

return false;

}

1. Write a hasValue() method that takes an integer value and returns true or false depending on whether the value argument can be found in the tree. Use a recursive helper function for the search.

public hasValue(int value) {

if(root == null) {

return false;

}

else {

return hasValue(root, value);

private boolean hasValue(Node node, int value) {

if(node == null) {

return false;

} else if (node.value == value){

return true;

} else if (value < node.value) {

return find(node.leftChild, value);

} else {

return find(node.rightChild, value);

}

}

1. Write a getDepth() method that returns the depth of the tree. The function should use a recursive helper function to find the depth. The helper function should take two arguments for the current Node, and an integer argument for the current depth. It should return the value of the depth argument if it has no children. (Otherwise it returns the greater of the depths of its children.) The root has depth 0. It should return -1 if the tree is empty.

public int getDepth() {

if (root == null) {

return -1;

}

else {

return getDepth(root, 0);

}

}

private int getDepth(Node current, int currentDepth) {

int leftDepth = 0;

int rightDepth = 0;

if (current.rightChild == null && current.leftChild == null) {

return currentDepth;

}

else {

if(current.leftChild != null) {

leftDepth = getDepth(current.leftChild, currentDepth+1);

}

if (current.rightChild != null) {

rightDepth = getDepth(current.rightChild, currentDepth+1);

}

if (rightDepth < leftDepth) {

return leftDepth;

}

else {

return rightDepth;

}

}

}