**2019 Final**

**1d) write recursive function to visit every node, and add 1 to data stored in each node**

Struct node {

Struct node \*p\_left\_tree;

Struct node \*p\_right\_tree;

Int data;

};

Void add\_one (struct node \*p)

{

// base case, if it reaches the end of a tree,

If (p->p\_left\_tree == NULL && p->p\_right\_tree == NULL)

{

p->data += 1;

return;

}

// traverse the left subtree, and when that is NULL, traverse the right

Else if {p->p\_left\_tree != NULL} return add\_one(p\_left\_tree);

Else if (p->p\_right\_tree != NULL} return add\_one(p\_right\_tree);

}// add\_one()

// better version

void add\_one(struct node \*p)

{

p->data++;

if (p->p\_left\_tree) add\_one(p->p\_left\_tree);

if (p->p\_right\_tree) add\_one(p->p\_right\_tree);

**2b) recursive function that finds greatest common denominator**

Int gcd (int x, int y)

{

// base case

If (y ==0) return x;

If (x < y) return gcd(y,x);

Else return gcd(y, x mod y);

}

**4a) write code for the swap() function**

Void swap(int \*array, int g, int h)

{

// swap g with h

Int temp;

Temp = array[g];

Array[g] = array[h];

Array[h] = temp;

}

**4d) modify the code**

Struct experiment {

Int A, B, C;

Double val;

}

Int compare(experiment x, experiment y)

{

Enum vars {A, B, C, val};

Enum vars var;

For (i=0; i<4; i++)

{

}

}

**5)**

// needs a pointer to the array and it’s size

void printDups(int \*arr, int arrSize) {

int i;

for (i=0; i<arrSize; i++) {

if (arr[abs(arr[i])-1]) >= 0 {

arr[abs(arr [i]) - 1] = -arr[abs(arr[i]) - 1];

}

else {

printf(“The element %d has been repeated”, arr[i]);

}

}

}

**4b) modify the bubble sort code**

void bubblesort (T \*a, int size) {

int i, j, swaps;

for (j=0; j<size; j++) {

swaps = 0;

for (i=0; i<size – 1; i++) {

if (a[i] > a[i+1]) {

swap(a, i, i+1);

swaps++;

}// end if

}// end for

if (swaps == 0) break; // if no swaps were performed in the inner loop, then it is already ordered

}

// still worst case of O(N2) but now there’s a stopping condition

**2017 Final**

2 – part 1

void add(Linkedlist \*l, int\_exp, int\_coef)

{

// need to instantiate 3 pointers,

// one to the previous, one to the current, and

// if the linked list is empty, make the new term the head of the list

if (isEmpty)

{

newTerm = createNode(int\_exp, int\_coef, NULL); // must create a new node

l->head = newTerm;

return;

}

// if the list is not empy

else

{

// need a loop to traverse through, allowing to check where the

// element needs to be placed

while (l->head != NULL)

{

// if the new term’s exponent is larger

if (int\_exp > l->head.exp)

l->head = l->head.next; // moves the head pointer along in the list

}

}  
}

**2018 Final**

**2)** **Make function that creates doubly linked list from singly linked**

// review double pointers

void makeDouble(struct single \*ps, struct double

3) part 2, write function for queue push operation

void push(int n)

{

}

z

**General Practice**

implementing a queue

// FIFO operation

// need head and tail pointer

// assuming the other operations such as making a node have already been defined

int count = 0; // global counter variable that keeps track of elements in queue

void q\_insert(int n)

{

// make the node

Node \*pnew = (Node \*)malloc(sizeof(Node));

if (pnew)

{

pnew->data = n;

pnew->next = null;

if (end != NULL) end->nextnode = pnew;

if (front == NULL) front = pnew;

count++;

}

else return; // if the node cannot be made

}

int dequeue()

{

if (isEmpty()) return -1; // the queue does not exist

else

{

Node \*temp; // create a temporary pointer

temp = front; // save the front of the queue to temp

front = front->nextnode; // advance the front queue pointer

count--;

free(temp)

return temp->data;

}

}

**2020 Final Exam**

Question 2

1. False, bubble sort will swap 50 and 36, putting 50 into position 1