Problem Number(s)	Course Evaluation	Possible Points	Earned Points
COMPULSORY			
1	Queues	10	
2	Sorting	15	
3	Binary Tree	15	
4	BST	15	
5	AVL	10	
6	Binary Heap	15	
7	Priority Queue	10	
8	Binomial Queue	20	
9	Hash Tables	10	
		TOTAL POINTS 120	

Instructions:

i. **All Questions Are Compulsory**

ii. **You have 3 hours to complete the Exam.**

iii. If information appears to be missing from a problem, make a reasonable assumption, **state it and proceed.**

iv. If the space to answer a question is not sufficient, use the back of each question's page.

v. **The exam is closed book. No calculators allowed.**

1. **QUEUE**
   A queue abstract data type (ADT) is specified using a vector representation as shown below:

```
struct queue
{
      int size; //trusted
      int capacity; //trusted
      int* data; //trusted
      int front; //not trusted unless the queue is non-empty
      int rear; //not trusted unless the queue is non-full
};

typedef struct queue Queue;
```

Assuming that an opaque object QUEUE has been defined in a queue.h file, use the above representation to answer the questions below.

a) Complete the queuing function below making sure that an item is inserted at the back (rear) of the queue. This function returns a Status (defined in a status.h file). [10 points]

```
Status queue_enqueue(QUEUE hQueue, int item) {
    Queue* pQueue = (Queue*) hQueue;
    int* temp;
    int i;

    if (pQueue->size >= pQueue->capacity) //it is full (there is no room) so fix it
    {
        temp = (int*) malloc(sizeof(int) * pQueue->capacity * 2);
        if (temp == NULL)
        {
            return FAILURE;
        }
        //copy the old info into temp
        for (i = 0; i < pQueue->size; i++)
        {
            temp[i] = pQueue->data[(pQueue->front + i) % pQueue->capacity];
        }
        free(pQueue->data);
        pQueue->data = temp;
        pQueue->front = 0;
        pQueue->rear = pQueue->size;
        pQueue->capacity *= 2;
    }
    pQueue->data[pQueue->rear] = item;
    pQueue->size++;
    pQueue->rear = (pQueue->rear + 1) % pQueue->capacity;

    return SUCCESS;


}
```

2. SORTING

Consider an array, **SORT,** shown below

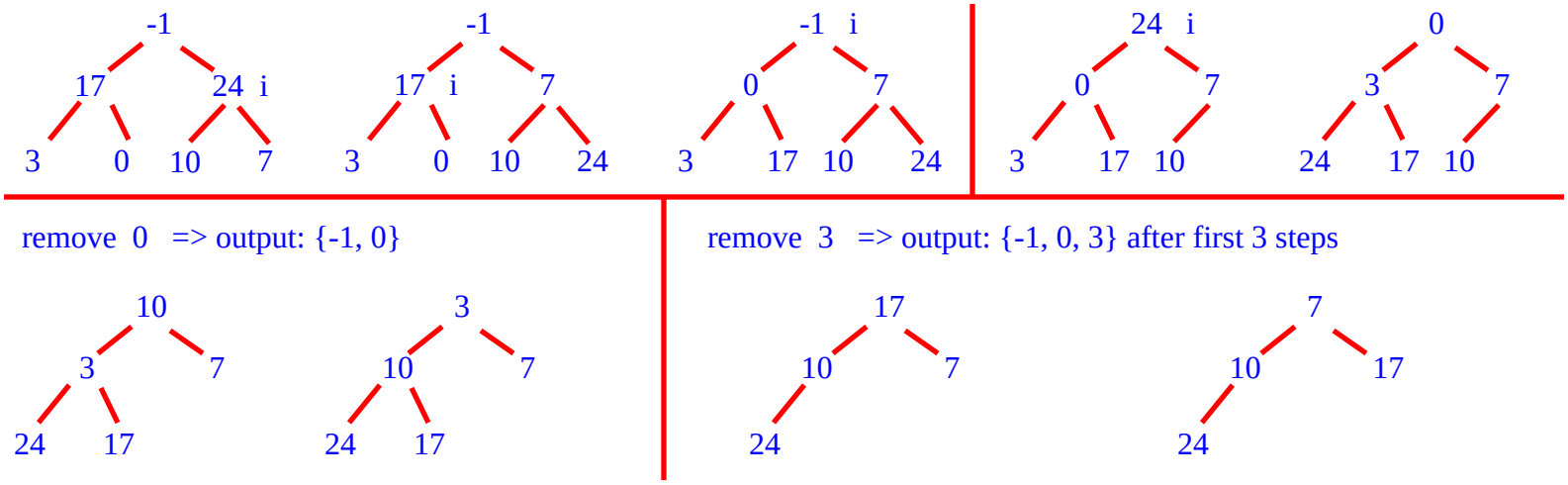
SORT =

-1	17	24	3	0	10	7

Assuming that the elements are to be arranged in an ascending order starting from the leftmost element, use this array to provide the solution for the following questions.

a) Show the first partition achieved using the Quick sort algorithm. Assume the left-most element (i.e. index 0) is selected as the pivot [2 points].

b) Show the step by step process of sorting the above array using the Heapsort algorithm: Please show the resulting array for the first 3 steps after the min heap is created. [5 points].



c) In class we looked at sorting using shell sort algorithm. Wikipedia describes Shell sort as "a generalization of insertion sort that allows the exchange of items that are far apart. The idea is to arrange the list of elements so that, starting anywhere, considering every h^{th} element gives a sorted list. Such a list is said to be *h*-sorted". Starting with SORT as show, show the sorting achieved with decreasing h (gap length as the sorting continues). Please label the gap length used in each sorting iteration [3 points]:

SORT	-1	17	24	3	0	10	7
h – 5							
h - 3							
h - 2							

d) Heap Sort involves first converting the array into a heap and repeatedly extracts the max element and heapify the tree until all elements are sorted from the rear. Show the process of Sorting the array in c above using Heap sort [5 points]

3. Trees

A tree definition is given as follows:

```
typedef struct binaryNode*   BinaryNode_Ptr;
struct binaryNode
{
   BinaryNode_Ptr left;
   BinaryNode_Ptr right;
   int item;
};
```

A pointer to the root of the tree is declared as follows:

```
BinaryNode_Ptr root;
```

a) Write a recursive function, **void mirrorTree(`BinaryNode_Ptr tree`),** that changes a given input tree so that it becomes the mirror image of the original tree. [8 points]

For example:

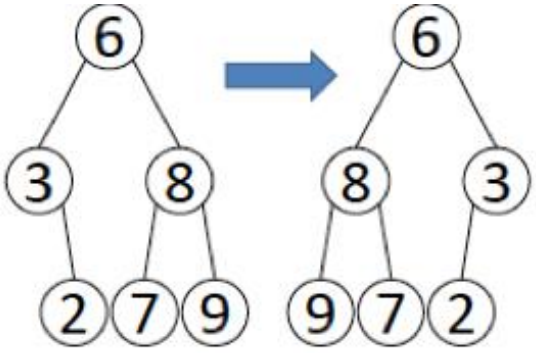


Figure 1: Mirror trees.

For this question, assume you have a node object that has the basic methods implemented: getLeft(), getRight(), setLeft(), setRight(), getValue(). All the values in a node are integers.

```
void mirrorTree(struct Node* node)
{
    if (node==NULL){
        return;
    }
    else {
          node* temp;

        /* do the subtrees */
        mirrorTree(node->left);
        mirrorTree(node->right);

        /* swap the pointers in this node */
        temp = node->left;
        node->left  = node->right;
        node->right = temp;
        }
}
```

b) Write a function that checks whether your mirror tree is the same as the original, i.e. checks whether your tree is identical to the original tree [7 points].

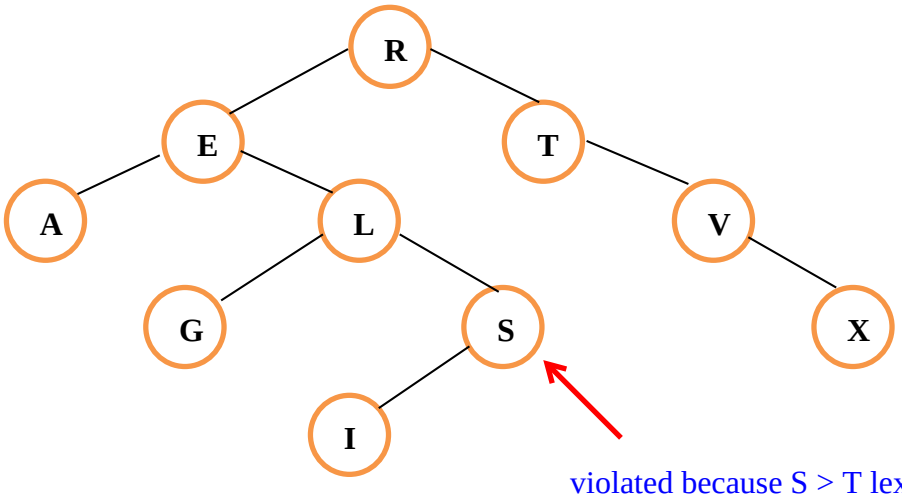
```
int identicalTrees(node* a, node* b)
{
    /*1. both empty */
    if (a == NULL && b == NULL)
        return 1;
    /* 2. both non-empty -> compare them */
    if (a != NULL && b != NULL)
    {
        return
        (
            a->data == b->data &&
            identicalTrees(a->left, b->left) &&
            identicalTrees(a->right, b->right)
        );
    }

    /* 3. one empty, one not -> false */
    return 0;
}
```

4. Binary Search Tree

a) Briefly explain what a binary search tree (BST) is, listing its properties [3 points]

b) Is the following binary tree a BST or not, and why? [2 points]

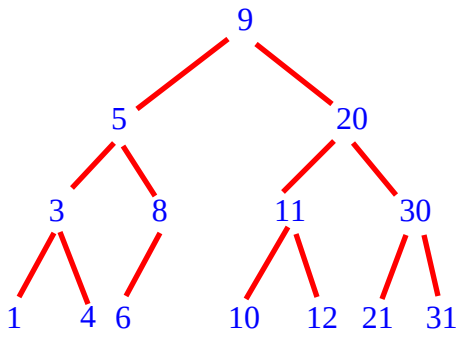


violated because S > T lexicographically
S must be Left child of T
I < L, I must be Right child of G

c) If you answered (No) on (b) above, re-draw the tree in correct ordering [2 points]

d) Given the following preorder and in-order traversals for an unknown binary tree, determine the exact tree that would generate these traversals and then draw that tree. Once you have generated the tree be sure to check your work [8 pts].

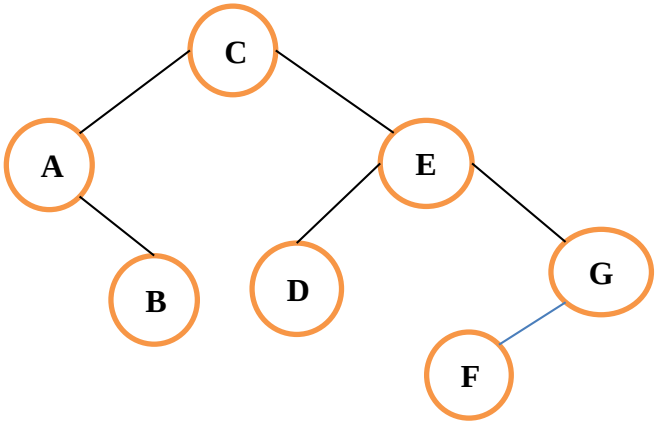
Preorder: 9,5,3,1,4,8,6,20,12,10,11,30,21,31
Inorder: 1,3,4,5,6,8,9,10,11,12,20,21,30,31



5. AVL Tree

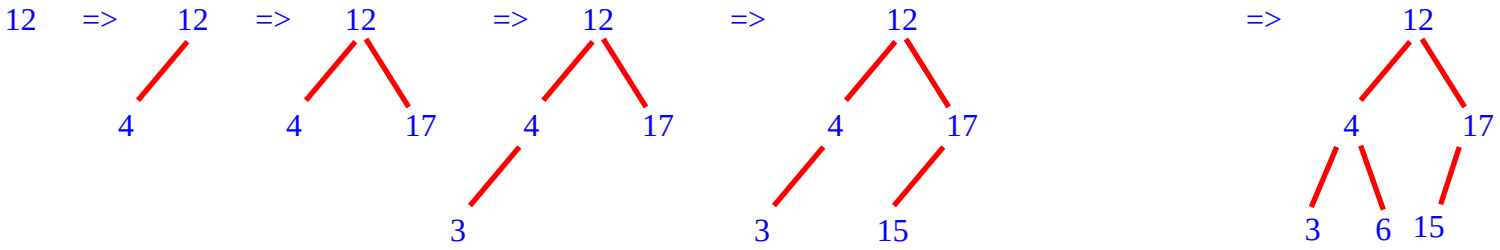
a) For each node shown in the binary tree below, show its depth, height and the AVL balance factor. Write your answers in the given table [5 points]

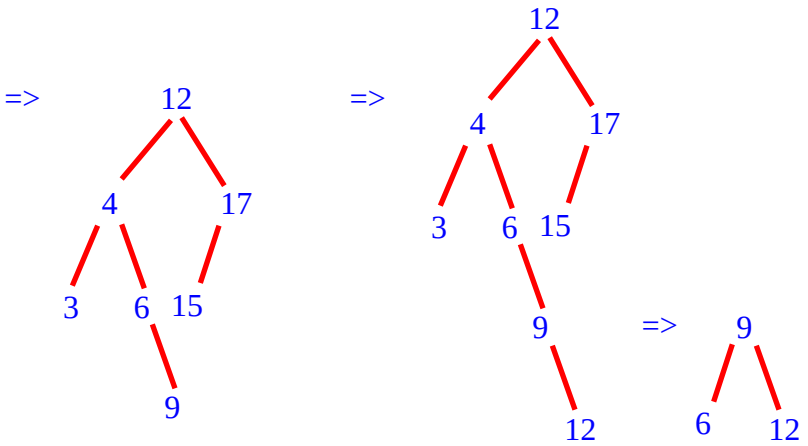
NB: An incorrect answer will attract -1 point



Node	Depth	Height	Balance Factor
A			
B			
C			
D			
E			
F			
G			

a) Draw the sequence of AVL trees obtained when the following keys are inserted one-by-one, in the order given into an initially empty AVL search tree: {12, 4, 17, 3, 15, 6, 9,12}. Identified rotations, if there is any [5 points].





6. Binary Heap

a) Explain what the heap data structure is, state its defining properties and explain how to convert between the tree and vector (array) representations of a heap. [2 marks]

b) Describe an optimally efficient algorithm for transforming any random vector into a binary heap vector and explain why it works. [2 marks]

c) Using the tree instead of the vector representation for clarity, apply this algorithm to the binary tree to the integer vector, 14, 3, 8, 49, 4, 2, 1,78, producing a frame-by-frame trace of the execution. For this answer, please assume a Max-Heap and show a new tree whenever any nodes change. [6 marks]

d) Explain how to rearrange the heap after having extracted its top so that what remains is still a heap. Follow this procedure to extract the top three values, one by one, from the heap you built, producing a frame-by-frame trace as above. [5 marks]

7. Priority Queue

a) What is a priority queue? Explain the data structure known as a binary heap and document how a heap is stored in a simple linear block of memory. [1 points]

Priority Queue is an extension of queue with following properties.

Every item has a priority associated with it.
An element with high priority is dequeued before an element with low priority.
If two elements have the same priority, they are served according to their order in the queue.

b) Describe, and estimate the costs of, procedures to:

i. Insert a new item into an existing heap [1 points];

ii. Delete the topmost item from a non-empty heap [1 points];

iii. Starting from an array holding N items in arbitrary order, rearrange those items so that they form a heap, taking time less than that which would be needed if the items were just inserted into the heap one after the other [1 points];

iv. Inserting and Removing an item from a priority queue implemented as [6 points]:

	Remove Max	Insert element
Sorted Array	O(1)	O(N)
Sorted Linked list	O(1)	O(N)
Unsorted Array	O(N)	O(1)
Unsorted Linked List	O(N)	O(1)
Binary Heap	O(logN)	O(logN)
Binomial Queue	O(logN)	O(logN)

8. **Binomial Queue**

A binomial queue is implemented as a Max-Heap.

a) Give the binomial queue that results when the following elements are inserted into an initially empty binomial queue (Show a frame by frame sequence) [2 points].
17, 24, 3, 12, 78

b) Give the binomial queue that results when the following keys are inserted into an empty binomial queue [4 points].
12, 2,167, 19, 64, 3,45, 19, 23, 16, 12, 4

c) Give the result of delete the maximum for each of the above binomial queues [4 points]

d) Give the results when the "merge/join" operation is performed after part c above is completed. [6 points]

e) Draw the binomial queue represented by this binary number: 10101 and determine the number of nodes which are there in the forest? [4 points]

9. **Hash Tables**

Consider a hash table with table size, m = 7 and the hash function:
Use the following **Collision resolution mechanisms**, draw the hash table (with the above table size and hash function) that would be produced by inserting the following values, in the given order (from left to right), into an initially empty table:

26, 32, 19, 11, 12.

For hashing use: hash(value) = value % m

a) Chaining [2 points]

b) Open addressing with Linear probing [2 points]
Recall that this means: index = (hash(value) + i) % m i=0,1,2….

c) Open addressing with Quadratic Probing [2 points]
Recall that this means: index = (hash(value) + i + i^2) % m i=0,1,2….

d) Open addressing with Double Hashing. [4 points]
Use hash2 (value % 9)
Recall that this means: index = (hash(value) + i *hash2 (value%9)) % m
i=0,1,2….