METU Department of Computer Engineering
CENG 213 Data Structures
Final Exam
120 min.

ID: ______________________________

Name: ______________________________

Section: ______________________________

Q1 (20 pts)	
Q2 (20 pts)	
Q3 (20 pts)	
Q4 (20 pts)	
Q5 (20 pts)	
TOTAL	

> **Note:** Asking questions is not allowed. If you think that there is a problem with a question, please indicate it clearly and we'll evaluate it when grading your papers.

Q1. (20 pts.) What is the worst case running time (in big-O notation) for each of the following operations?

Operation	**Answer**
Finding the maximum element in a complete binary search tree of *N* elements.	
Finding the in-degree of a given node in a graph with *V* vertices and *E* edges under adjacency list representation.	
Deleting all edges from a given node in a directed graph with V nodes and E edges under adjacency list representation.	
Sorting inversely sorted sequence of N numbers with Merge-Sort.	
Sorting already sorted sequence of N numbers with Selection Sort.	
Finding an element in a hash table constructed with open addresing with linear probing, when load factor is less than 0.5.	
Inserting N elements successively into an empty AVL tree.	
Traversing a binary tree with N nodes with level-order traversal.	
Deleting the minimum element from a priority queue that is implemented with a binary heap.	
Inserting a new element into a priority queue that is implemented with a sorted linked list	

b) Fill in the following table of worst-case complexities for each of the given data structures and operations. Assume that both singly-linked list and doubly-linked list are circular and the elements are sorted from smallest to largest. **For deleting a given element, do not include the cost of finding it.** Assume separate chaining implementation for the hash table and a good hash function is used with a low load factor.

	Sorted singly linked list	**Sorted doubly-linked list**	**Binary Tree**	**Stack**
Finding the smallest element				
Finding the largest element				
Searching for a given element				
Deleting a given element				
Finding the median				

Q2. (20 pts) A queue can be implemented by using a circular linked list and a single pointer, called the `last`, which always points to the lastly inserted element. Its next pointer, `last->next`, always points to the first element in the queue. Use this circular linked list approach to implement the `enqueue` and `dequeue` functions. Use the following definitions. Note that for an empty queue `last` points to `NULL`.

`template <class T> struct Node {`
`    T element;`
`    Node* next;`
`}`
	`template <class T> class Queue {`
	`    Node<T>* last;`
	`public:`
	`    void` **`enqueue`**`(const T& element);`
	`    void` **`dequeue`**`();`
	`};`

```
template <class T> void Queue<T>::enqueue(const T& element) {
    Node<T>* newNode = new Node<T>;
    newNode->element = element;
    if (last) {
```

```
    }
    else {
```

```
    }
}
template <class T> void Queue<T>::dequeue() {
    if (last) {
```

```
    }}
```

Q3. (20 pts.) Assume that a new method is being added to the binary heap class, namely `remove`, which removes the given object from the binary heap. You may assume that there is no duplication of objects in the heap. You can use available methods of BinaryHeap class in your implementation. Note that the heap structure and the heap order properties should be preserved after removal of the given object. **Important Note:** Do not shift the elements in the heap!

```
template <class Object>
class BinaryHeap {
  public:
    BinaryHeap(int capacity = 100); // Assume that index starts at 1.
    bool isEmpty() const;
    const Object & findMin() const;
    void insert(const Object & x);
    void deleteMin();
    void deleteMin(Object & minItem);
    void makeEmpty();
    void remove(const Object & x); // YOU WILL IMPLEMENT
  private:
    int theSize;  // Number of elements in heap
    vector<Object> array;   // The heap array
    void percolateDown(int hole);
};

template <class Object>
void BinaryList<Object>::remove(const Object & x) {

}
```

Q4. (20 pts) a) Consider a hash table of size **7** and the hash function ***h*(*x*) = *x* mod 7**. Suppose the following keys are inserted in the order shown into an empty hash table:

`15, 17, 8, 23, 3, 5`

Insert these keys using each of the following approaches. (12 pts.)

i. h(x) = x % 7; linear probing

0	
1	
2	
3	
4	
5	
6	

ii. h(x) = x % 7; quadratic probing

0	
1	
2	
3	
4	
5	
6	

iii. h(x) = x % 7; double hashing with $h_2(x) = x / 7 + 1$ (using integer division)

0	
1	
2	
3	
4	
5	
6	

b) Considering your answer to part (**i**) of **(a)** only, i.e. hash table created using linear probing:

i. What is the load factor of the table? (2 pts.)

ii. What is the average number of probes in a successful search? (3 pts.)

iii. What is the average number of probes in an unsuccessful search? (3 pts.)

Q5. (20 pts.) a) What is the smallest key that could be at the leaf level of a heap built from a random order of the keys **1** through **16**?

(3 pts.)

b) Insert the following items into an empty heap one by one in the given order (use index 0 for the first element).

77, 22, 9, 68, 16, 34, 13, 8,15,20

Show the resulting heap as an array: (4 pts.)

Show the resulting heap as a complete binary tree: (4 pts.)

c) Given the following array, apply the `buildHeap` algorithm. (Recall that the buildHeap algorithm builds a heap from bottom up through a sequence of percolateDown operations.)

Z	**X**	**C**	**V**	**B**	**N**	**M**	**A**	**S**	**D**

Show the resulting heap as an array: (5 pts.)

d) Consider the following heap:

12	**15**	**24**	**37**	**50**	**62**	**75**	**82**

You are going to apply *two* `deleteMin()` operations on this heap. Show the array contents after each `deleteMin`. (4 pts.)
