Exam Number:

Points off	1	2	3	4	5	Total off	Net Score

CS 307 – Midterm 2 – Spring 2009

Name____

UTEID login name ________________________________

TA's Name: Guhan Todd Xiuming (Circle One)

Instructions:

1. Please turn off your cell phones and other electronic devices.
2. There are 5 questions on this test.
3. You have 2 hours to complete the test.
4. You may not use a calculator on the test.
5. When code is required, write Java code.
6. When writing methods, assume the preconditions of the method are met. Do not write code to check the preconditions.
7. In coding question you may add helper methods if you wish.
8. After completing the test please turn it in to one of the test proctors and show them your UTID.

1. (2 points each, 30 points total) Short answer. Place you answers on the attached answer sheet.

- If the code contains a syntax error or other compile error, answer “compile error”.
- If the code would result in a runtime error / exception answer “Runtime error”.
- If the code results in an infinite loop answer “Infinite loop”.

Recall that when asked for Big O your answer should be the most restrictive correct Big O function. For example Selection Sort has an average case Big O of $O(N^2)$, but per the formal definition of Big O it is correct to say Selection Sort also has a Big O of $O(N^3)$ or $O(N^4)$. I want the most restrictive correct Big O function. (Closest without going under.)

A. What is the Big O of the following method? N = `mat.length`

```
// pre: mat != null, mat is a square matrix
public double methodA(double[][] mat){
    int total = 0;
    for(int c = mat.length - 1; c >= 0; c--)
        for(int r = c; r >= 0; r--)
            total += mat[r][c];
    return total;
}
```

B. What is the worst case Big O of the following method? N = `data.length`

```
public int methodB(int[] data){
    int count = 0;
    for(int i = 0; i < data.length; i++){
        if( data[i] % 2 == 0 )
            count++;
    }
    for(int i = 1; i < data.length; i++){
        if( data[i] % 10 == 0 )
            count++;
    }
    return count;
}
```

C. What is the worst case Big O of the following method? N = `data.length`. The `length` method from the `String` class is O(1).

```
public int methodC(String[] data){
    int sum = 0;
    for(int i = 0; i * i < data.length; i++){
        int temp = i * i;
        if( data[temp] != null )
            sum += data[temp].length();
    }
    return sum;
}
```

D. What is the worst case Big O of the following method? N = `data.length`.

```
public boolean methodD(int[] data, int tgt){
    int temp;
    boolean found = false;
    int index = 0;
    int limit = data.length - 4;
    while( index < limit && !found ){
        temp = 0;
        for(int off = 0; off < 5; off++)
            temp += data[index + off];
        found = temp == tgt;
        index++;
    }
    return found;
}
```

E. A method is O(2^N). It takes 5 seconds for the method to complete when given a data set with 100 elements. What is the expected time for the method to complete when given a data set with 104 elements?

F. A method is O(N^2). It takes 1second for the method to complete when given a data set with 10,000 elements. What is the expected time for the method to complete when given a data set with 30,000 elements?

G. What is returned by the method call `methodG(7)`?

```
public int methodG(int n){
    if(n >= 10)
        return 3;
    else
        return (10 - n) + methodG(n + 1);
}
```

H. What is output by the method call `methodH("UTCS")`? The `substring(int)` method returns a substring that begins with the character at the specified index and extends to the end of the string.

```
public void methodH(String s){
    if( s.length() == 1 )
        System.out.print(s);
    else{
       System.out.print( s.charAt(0) );
       methodH( s.substring(1) );
       System.out.print( s.charAt( s.length() - 1 ) );
    }
}
```

I. What is returned by the method call `methodI(4)`?

```
public int methodI(int n){
    if(n <= 0)
        return 2;
    else
        return n + methodI(n - 1) + methodI(n - 3);
}
```

J. What is the Big O of removing the element at position 0 from an array based list that already contains N elements?

K. What is the Big O of removing the element at position 0 from a linked list that already contains N elements?

L. The best case Big O of the insertion sort algorithm is O(N). Under what conditions does this best case occur?

M. Given a sorted array of 4000 elements, about how many elements will the binary search algorithm examine when searching for a value that is not present in the array? Given your answer as an integer. Recall $\log_2 1{,}000 \sim= 10$.

N. Briefly explain what operation on a linked list can be made more efficient by having an iterator for the linked list.

O. Consider the following methods from the Java `ArrayList` class.

Method Summary	
void	**add**(E o) Appends the specified element to the end of this list.
void	**add**(int index, E element) Inserts the specified element at the specified position in this list.
E	**get**(int index) Returns the element at the specified position in this list.
E	**remove**(int index) Removes the element at the specified position in this list. Return the element that is removed from the list.
E	**set**(int index, E element) Replaces the element at the specified position in this list with the specified element. Returns the element that was previously at index.
int	**size**() Returns the number of elements in this list.

What is output when method `methodO` is called?

```
public void methodO(){
    ArrayList<String> ls = new ArrayList<String>();

    ls.add("M");
    ls.add(0, "K");
    ls.add("O");
    ls.add(1, "I");
    ls.add(ls.get(2));
    ls.set(2, "GG");

    for(String s : ls)
        System.out.print( s );
}
```

2. (Implementing Data Structures 20 points) In this question you will work with a type of list called a `FrequencyList`. The `FrequencyList` is very similar to the `List` class but it also tracks how often items are accessed via the `get` method. Every time an element of the list is accessed its frequency is incremented. You will write a method to consolidate the `FrequencyList`.

Here is an abstract view of the `FrequencyList`.

position in list	0	1	2	3	4	5
data	"cat"	"dog"	"cat"	"dog"	"duck"	"cat"
access frequency	2	12	18	10	18	1

Notice some values appear multiple times in the `FrequencyList`. The `consolidate` method removes all duplicates from the `FrequencyList` and adds the frequency of these duplicate elements to the first occurrence of that element. Elements after the duplicate are shifted forward in the list. After a call to `consolidate` the `FrequencyList` shown above would become the following:

position in list	0	1	2
data	"cat"	"dog"	"duck"
access frequency	21	22	18

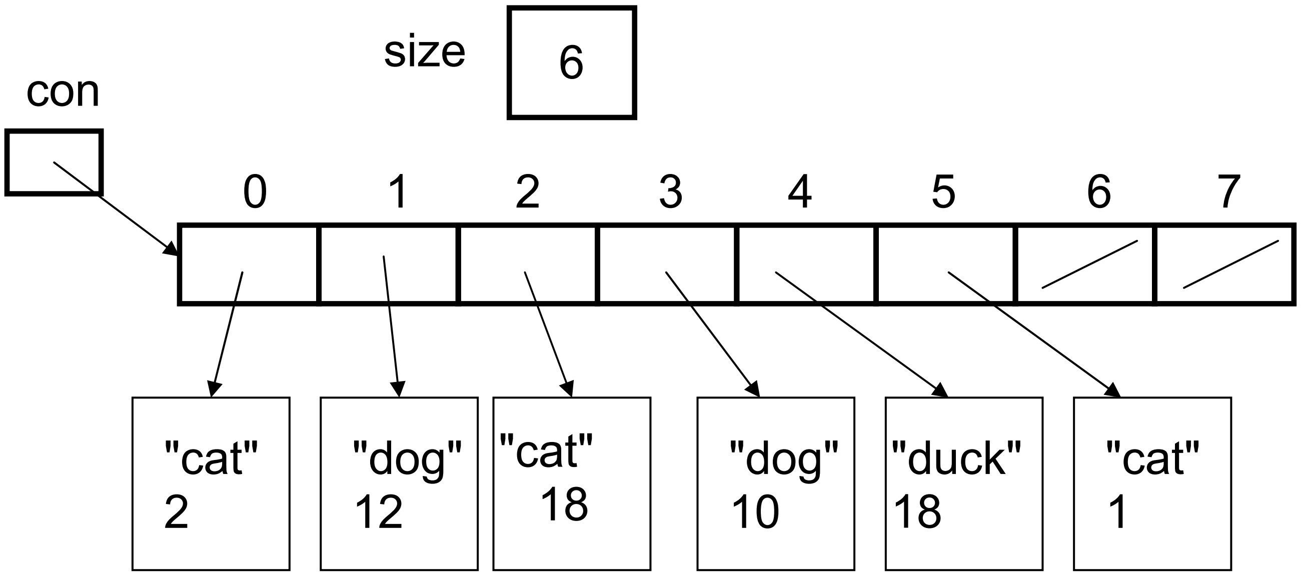
The `FrequencyList` uses an array of `Pair` objects much like the array based list example we did in class. Each `Pair` object holds one `Object`, which is the actual element in the list and an `int`, which is the number of times that element has been accessed.

Just like out array based list examples in class the array of `Pair` objects in the `FrequencyList` class may have extra capacity.

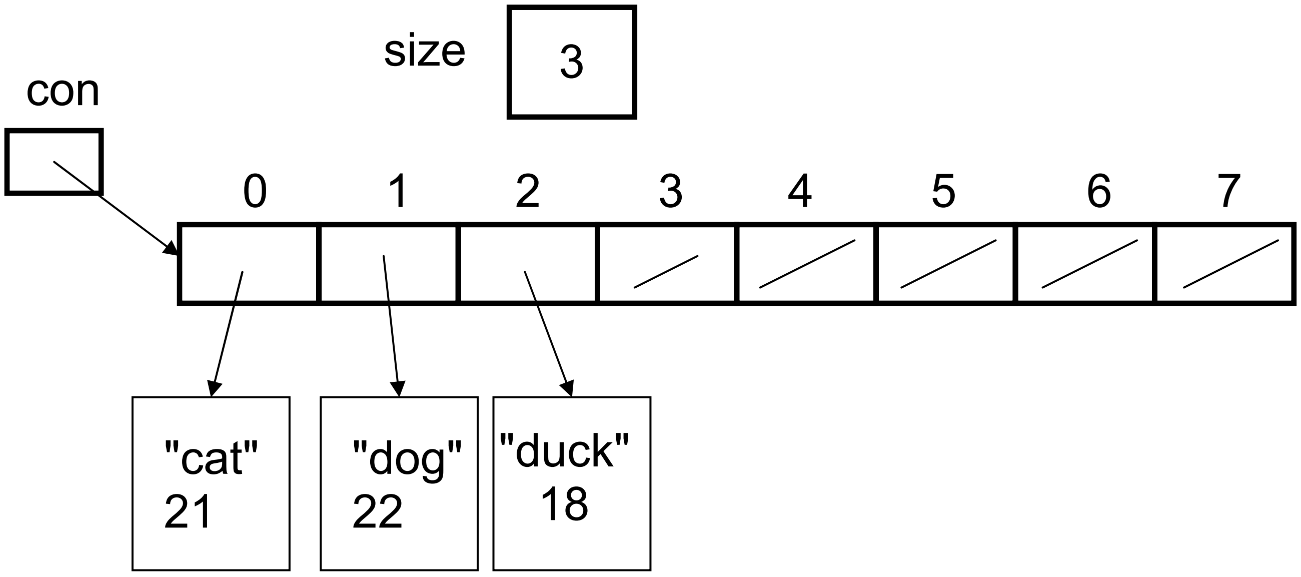
Here are the `FrequencyList` and `Pair` classes.

```
public class FrequencyList{

      private Pair[] con;
      private int size;

      public void consolidate() // to be completed by students
      // other methods not shown.
}

public class Pair{
      public Pair(Object obj) // create new Pair with frequency 0

      public Object getData()
      public int getFreq()

     public void increaseFreq(int n) // increase freq by n, n >= 0
}
```

Here is a more concrete view of the original `FrequencyList` shown above.



And this is what it would be after a call to `consolidate`:



Complete the method `consolidate` which is a method in the `FrequencyList` class.

- You may not use any other methods in the `FrequencyList` class unless you write them yourself.
- You can use the `equals` methods from the elements of the `FrequencyList`.
- **You cannot use any of the Java Collection classes such as `ArrayList` or `LinkedList`.** (Native arrays are not a Java Collection class.)

```
// recall this method is part of the FrequencyList class
// pre: none
// post: this FrequencyList has been updated as described in the
// question.
public void consolidate(){
```

3. (Implementing data structures, 20 points). Write an instance method for a `LinkedList` class named `swap`. This method swaps two elements in the linked list with each other.

- This `LinkedList` class uses singly linked nodes.
- Each node stores a single `Object` and a link to the next node.
- This `LinkedList` class only contains a reference to the first node in the list.
- The last node in the list's `next` reference is set to `null`.
- When the list is empty `head` is set to `null`.
- The list uses 0 based indexing.

Examples of the affect of the `swap` method on lists:
`[A, C, D, A, B].swap(1, 3)` changes the list to `[A, A, D, C, B]`
`[A, C, D, A, B].swap(0, 4)` changes the list to `[B, C, D, A, A]`
`[A, C, D, A, B].swap(1, 1)` is a valid call, but does not alter the list.

- **You may not use any other methods in the `LinkedList` class, but you may create your own helper methods if you wish.**
- **Your solution must be O(1) space. In other words you cannot use an auxiliary array or ArrayList. Your method should be O(N) time.**
- **Hint: Do not actually change the position of any nodes in the list. Instead swap the data in the affected nodes.**

```
public class Node {
     public Node(Object value, Node next)

     public Object getData()
     public Node getNext()

     public void setValue(Object value)
     public void setNext(Node next)
}

public class LinkedList {
     // points to the first node in the list
     // if the list is empty, head == null
     private Node head;
     private int size;

     // pre: 0 <= pos1 < size(), 0 <= pos2 < size, pos1 <= pos2
     // post: The data at positions pos1 and pos2 have been
     // swapped.
     public void swap(int pos1, int pos2){
```

```
// pre: 0 <= pos1 < size(), 0 <= pos2 < size, pos1 <= pos2
// post: The data at positions pos1 and pos2 have been swapped.
public void swap(int pos1, int pos2){
     assert pos1 >= 0 && pos1 < size && pos2 >= 0 && pos2 < size
               && pos1 <= pos2 : "Failed precondition.";
```

4. (Using data structures, 15 points) Write a method that returns the kth element from a `SortedSet`. The kth value in a `SortedSet` is the kth smallest value in the `SortedSet`. (See examples below.) The `iterator` for the `SortedSet` iterates through the elements of the `SortedSet` in ascending order. **This method is <u>not</u> in the `SortedSet` class so you do not have access to the `SortedSet`'s storage container or other instance variables.**

- Assume the `SortedSet` class is generic based on Java's generic syntax.
- The `SortedSet` is not altered as a result of this method.
- Your method must be O(1) <u>space</u>. In other words you cannot use an auxiliary array or ArrayList.

Implement the following method:

Examples of method calls. Assume values are `Integer` objects, not primitive `ints`.

```
(1, 5, 10, 12, 50, 100).getKth(1) -> returns 1
(1, 5, 10, 12, 50, 100).getKth(2) -> returns 5
(1, 5, 10, 12, 50, 100).getKth(4) -> returns 12
(1, 5, 10, 12, 50, 100).getKth(6) -> returns 100
(1, 5, 10, 12, 50, 100).getKth(-2) -> returns null
(1, 5, 10, 12, 50, 100).getKth(10) -> returns null
```

Here are the methods from the `SortedSet` class you may use:

`Iterator<E>` **`iterator`**`()`
Returns an iterator over the elements in this set in ascending order.
`boolean` **`contains`**`(Object o)`
Returns `true` if this set contains the specified element.
`int` **`size`**`()`
Returns the number of elements in this set (its cardinality).

Recall the methods from the `Iterator` interface:

`boolean` **`hasNext`**`()`
Returns `true` if the iteration has more elements.
`E` **`next`**`()`
Returns the next element in the iteration.
`void` **`remove`**`()`
Removes from the underlying collection the last element returned by the iterator

Complete the following method on the next page:

```
// pre: none
// post: return the kth element of theSet. theSet is not
// changed as a result of this method call. If k is not
// a valid value return null.
public int getKth(int k){
```

```
// pre: none
// post: return the kth element of theSet. theSet is not
// changed as a result of this method call. If k is not
// a valid value return null.
public int getKth(int k){
```

5. (Recursion, 15 points) Write a method to determine how many ways a competitor may earn honorable mention in a chess tournament.

- The tournament consists of a number of rounds.
- A player plays every round.
- The possible results for a round are win, draw, and lose.
- A win is worth 5 points, a draw is worth 3 points, and a loss is worth 1 point.
- The number of points required for honorable mention will vary per tournament.

So for example, if a tournament consists of 3 rounds and 12 points are required to get honorable mention, there are 4 ways (out of a possible 27 in this example) to achieve the required 12 points.

Round 1 Result	Round 2 Result	Round 3 Result	Total Points
Win	Win	Win	15 (5 + 5 + 5)
Win	Win	Draw	13 (5 + 5 + 3)
Win	Draw	Win	13 (5 + 5 + 3)
Draw	Win	Win	13 (5 + 5 + 3)

Complete the following method. You may add helper methods if you wish. You may assume the initial values for `numRounds` and `pointsRequired` will be greater than 0.

```
// Return the number of ways required to achieve the number of
// points to get honorable mention in a chess tournament with
// the given number of rounds and the minimum required number
// of points.
public int waysToEarnHonrableMention(int numRounds,
                                     int pointsRequired){
```

// More room on next page

Name:_______________________________

TAs name: Guhan Todd Xiuming (Circle One)

Answer sheet for question 1, short answer questions

A. _______________________________ I. _______________________________

B. _______________________________ J. _______________________________

C. _______________________________ K. _______________________________

D. _______________________________ L. _______________________________

E. _______________________________ M. _______________________________

F. _______________________________ N. _______________________________

G. _______________________________ O. _______________________________

H. _______________________________