

MIDTERM 1 REVIEW

Problem 1. (10 points) What is printed to the screen when the following method is called? Will an exception be thrown?

```
public static void unknown() {
    int[] a={0,1,2,3,4,5,6,7}, b={9,8,7,6,5,4,3,2,1,0};
    a = b;
    b[0] = 5;
    b[1] = 4;
    a[2] = 9;
    for (int i=0; i<10; i++)
        System.out.println( a[i] + " " + b[i] + " " + a[b[i]] );
}
```

Answer 1. Output (no exceptions will be thrown):

```
5 5 4
4 4 5
9 9 0
6 6 3
5 5 4
4 4 5
3 3 6
2 2 9
1 1 4
0 0 5
```

Problem 2. (10 points) Assume that you have a class MyVectorOfCDs which is an implementation of a vector of MyCD using size and capacity. Write a method for this class which inserts a MyCD at the **front** of the vector.

Answer 2.

```
public void insertAtFront(MyCD theCD) {
    if ( size==capacity ) {
        MyCD[] temp = new MyCD[capacity+10];
        for (int i=0; i<size; i++)
            temp[i] = theDisks[i];
        theDisks = temp;
        capacity += 10;
    }
    for (int i=size; i>0; i--)
        theDisks[i] = theDisks[i-1];
    theDisks[0] = theCD;
    size++;
}
```

Problem 3. (10 points) How many basic steps are executed (as a function of n) when the following method is called? What is the worst case time complexity (big- $O()$)? Under what circumstances will it return both true and false?

```

public static boolean howLong(int n) {
    int first = 0, second = 0;           1
    for (int i=0; i<n; i++)              n
        for (int j=1; j<n; j*=2)        n lg n
            first++;                    n lg n
    for (int i=n; i>1; i=i/2)           lg n
        for (int j=n; j>1; j=j-2)      n lg n / 2
            second++;                  n lg n / 2
    if (first >= second)                1
        return true;                    1
    return false;                       0
}

```

Answer 3. Adding up the number of times each line is executed yields $3n \lg n + \lg n + 3$ basic steps and time complexity $O(n \lg n)$. It never returns true and false (in fact it will always return true).

Problem 4. (10 points) Erica and Richard are both implementing a vector. However, they disagree on how the array should be resized (when inserting into a full array). Erica plans to increase the capacity of the vector by 20 locations (when necessary). Richard plans to increase the capacity of the vector by tripling the number of locations (when necessary). Under what circumstances should you prefer Erica's vector? Under what circumstances should you prefer Richard's vector? Explain.

Answer 4. If the vector will become very large then Erica will resize the vector far more often than Richard. Since resizing vectors is slow, Richard's should be preferred if the vector grows very large. However, in many cases Erica's method uses far less space. If memory is scarce, then Erica's should be preferred.

To illustrate the points above, imagine that the initial capacity of the vector is 10 and there will be 1001 items inserted. Erica will have to resize 50 times and have 19 empty spaces while Richard will only have to resize 5 times, but have 1429 empty locations.

Problem 5. (10 points) The following method is included in a class called MyLL (which is an implementation of a linked list). What would be accomplished by calling **figureMeOut()**? Don't just translate line by line, but instead describe it at a high level. Show a small example. Explain.

```
public int figureMeOut(){
    int a=0, b=0;
    for (MyNode nd=head; nd!=null; nd=nd.getNext())
        a += nd.getValue();
    for (MyNode nd=head; nd!=null; nd=nd.getNext())
        if (nd.getValue() < 0)
            b += nd.getValue();
    return a-b;
}
```

Answer 5. The first loop finds the total of all numbers. The second loop finds the total of all the negative numbers. By returning $a-b$ the result is the sum of the positive entries. If you had a linked list containing 5,6,-2,4,-3 then $a = 10$, $b = -5$, and the return value will be 15.

Problem 6. (10 points) Assume that you have a class called MyNode each instance of which has 2 parts: a char letter and a MyNode next. The MyNode class has the normal methods (getNext(), setNext(), getLetter(), and setLetter()). Further assume that you have a class called MyLL which is an implementation of a linked list (using the class MyNode). Write a method called **howManyOfThisLetter(char letter)** which returns the number of times letter appears in the linked list.

Answer 6.

```
public int howManyOfThisLetter(char letter) {
    int count = 0;
    for (MyNode nd=head; nd!=null; nd=nd.getNext())
        if (nd.getLetter()==letter)
            count++;
    return count;
}
```

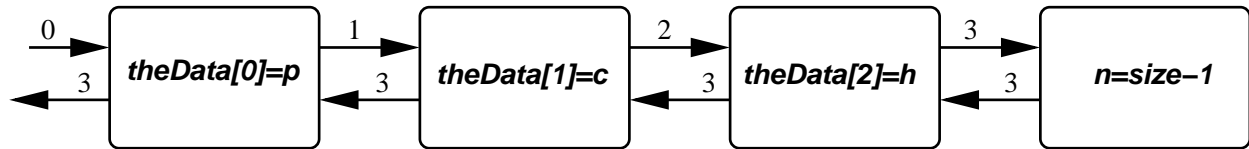
Problem 7. (10 points) Otis and Pamela are both implementing a linked list. Otis is planning to write a doubly linked list. Pamela is planning to write a singly linked list. What are the advantages of Otis's doubly linked list? What circumstances would lead you to prefer Otis's doubly linked list? What are the advantages of Pamela's singly inked list? What circumstances would lead you to prefer Pamela's singly linked list?

Answer 7. Since the nodes will contain a prev reference. Otis's doubly linked list allows moving backward. So inserting and deleting from the middle of the linked list is easier (don't have to keep a previous reference). Pamela's will be simpler to understand, easier to program, have fewer special cases, and have (slightly) less overhead.

Problem 8. (10 points) The following method is included in a class called MyVector. What would happen if you called this method with **unk(0)**? Don't just translate line by line, but instead describe it at a high level. How does the array change? What is returned? Show a small example including the stack frames. Explain.

```
public int unk(int n) {
    if (n >= size-1) {
        size--;
        return size;
    }
    theData[n] = theData[n+1];
    return unk(n+1);
}
```

Answer 8.



Array when *unk(0)* called =

<i>x</i>	<i>p</i>	<i>c</i>	<i>h</i>	
----------	----------	----------	----------	--

Array when *unk(1)* called =

<i>p</i>	<i>p</i>	<i>c</i>	<i>h</i>	
----------	----------	----------	----------	--

Array when *unk(2)* called =

<i>p</i>	<i>c</i>	<i>c</i>	<i>h</i>	
----------	----------	----------	----------	--

Array when *unk(3)* called =

<i>p</i>	<i>c</i>	<i>h</i>	<i>h</i>	
----------	----------	----------	----------	--

This method deletes the first item from a vector by sliding all of the items one position forward. Even though there will be an “extra” copy of the last element in the array, it will never be seen because `size` is decreased. It returns the (new) size of the vector.