CS 61B – Summer 2005 – (Porter)

Final Exam Part 1 – Aug 10, 2005

Do not open until told to begin

This exam is CLOSED BOOK, but you may use 2 letter-sized page of notes that you have created.

Problem 0: (1 point) Please fill out this information, and when told to begin, please put your name **on each page** of the exam.

Your name:	SOLUTION
Your cs61b login:	
Lab time:	
Lab TA's name:	
Name of person to your left:	
Name of person to your right:	

Do not write below here:

Problem	Score	Total possible
0	1	1
1 (10 minutes)	9	9
2 (10 minutes)	15	15
3 (40 minutes)	50	50
4 (20 minutes)	25	25
Total:	100	**100**

Do not open until told to begin

Good Luck!

Problem 1 (9 pts) – Java / Programming Methodology – (10 minutes)

What is the purpose of interfaces in Java? Why are they beneficial, and what problem do they solve?

Interfaces allow programmers to separate the specification of a data structure from its implementation. This is beneficial because programmers can modify a data structure's implementation to improve reliability, fix bugs, improve performance, etc., without disrupting the rest of the program.

What is polymorphism? Give a small example.

A method/function is polymorphic when it can be applied to a variety of different data types. For example, printList() can be applied to LinkedLists, SortedLinkedLists, ArrayLists, SortedArrayLists, etc.,

What is the error in the following code segment that prevents it from compiling?

```
public interface MyInterface {
     public int A();
     public int B();
}

public class MyClass implements MyInterface {
     public int A() { return 3; }
     public int B() { return 5; }

     public static void printVal(int val) {
          System.out.println(val);
     }

     public static void main(String[] args) {
          MyInterface obj = new MyClass();
          obj.printVal( obj.A() + obj.B() );
     }
}
```

printVal cannot be applied to obj since MyInterface doesn't contain its signature.

Problem 2 (15 pts) – Algorithm Analysis – (10 minutes)

Given $T(N) = 3N^2 + 2N$, prove or disprove the statement $T(N)$ is $O(N^2)$. Make use of the definition of Big-Oh.

T(N) is O(N^2) if and only if there exist constants c and N0 such that

3N^2 + 2N <= cN^2 for N > N0

subtract cN^2 from each side

(3-c)N^2 + 2N <= 0

factor out N

N ((3-c)N + 2) <= 0

N is positive for large values of N, so we have to show that ((3-c)N + 2) <= 0 for large N. Choose c = 4. We get

-N + 2 <= 0 for large n. Setting N0 to 3 satisfies this.

c = 4, N0 = 3.

Problem 3 (50 pts) – Employment Database – (40 minutes)

A company wishes to keep a record of all of its employees. For now, an employee is represented by an object of the following class:

```
public class Employee {
        public String name;
        public int yearsAtCompany;
        public float salary;

        public Employee (String name, int years, float salary) {
                this.name = name; this.yearsAtCompany = years; this.salary = salary;
        }
}
```

A) Devise a data structure that efficiently supports the printInOrderOfSeniority() method. This method prints out employees in order of how long they have worked at the company (those who have been at the company the longest are printed first, those who have been at the company the least are printed last). If two employees have worked at the company the same number of years then they are printed alphabetically (according to String's compareTo method). Describe your data structure in detail, and give the instance variables (and their types) it will contain.

Best solution (that I know of) 1: O(N)

Assume max age of a person is 120 years. Create a hashtable with 120 bins. Hash each Employee to the bin corresponding to their seniority level. Keep each linked list sorted.

Good solution 2: O(N log N)

Keep an unsorted linked list of Employee objects. To access the employee objects, copy them into an array, sort with merge sort or quicksort (using yearsAtCompany as the sort key).

Good solution 3: O(N log N)

Keep a priority queue (backed by a binary heap) of Employee objects. Use yearsAtCompany as the priority if using a max heap. If using a min heap, use (120-yearsAtCompany) as the priority.

B) Write an insert method for your data structure. If the employee is already in the data structure you should return false, else return true. Your method should have the following signature:

public boolean addEmployee(Employee e) { ... }

Soln 1: hash the seniority, add the object to the appropriate linked list

Soln 2: tack the employee record onto the linked list

Soln 3: use the insert method of the priority queue

C) What is addEmployee()'s running time? Briefly describe your answer.

Soln 1: O(1)

Soln 2: O(1)

Soln 3: O(log N)

D) Write a method called giveRaise() that increases each employee's salary according to the following system: The most senior 30% should get a 10% raise, and everyone else should get a 5% raise. If multiple employees with the same seniority fall across the 30% line, then give those employees the raise too. The signature should be:

public void giveRaise() { ... }

Soln 1: Create an array; foreach chain in the hash table, copy into the array; sort the array with mergesort or quicksort. Create an iterator for the list with three fields: a "currentRaise" field that is a float, and a "thirtyPercentLine" field that is an int, and a currentSeniority field that is an int. Set thirtyPercentLine to be the number of employees times 0.3. Set currentRaise to be 1.10. Traverse the list multiplying each employees salary by currentRaise. When you have given thirtyPercentLine employees the 10% raise, set currentSeniority to the value of seniority of the record your iterator points to. If there are more employees with that seniority level, give them the 10% raise. When you find an employee with a lower seniority, then set currentRaise to 1.05 and finish off the rest of the list.

Soln 2: Copy the linked list into an array and proceed as described in soln 1.

Soln 3: Sort the binary heap (which is already stored as an array). Proceed as soln 1.

E) What is the running time for giveRaise()? Briefly describe your answer

Soln 1: O(N) to build the array. O(N log N) to sort it. O(N) to traverse. Thus O(N log N)

Soln 2: same

Soln 3: O(N log N) to sort, O(N) to traverse, thus O(N log N)

Problem 4 (25 pts) – Hash Tables – (20 minutes)

In this problem, you compare two structures for storing a list of 25,000 words. All the questions for this problem ask for behavior "on the average"; for the purpose of this problem, you should assume that all the words will be accessed equally often.

Consider a hash table with n cells, each of which contains a pointer to an unsorted linked list of words that hash to that cell. Suppose 25,000 words are to be added to the hash structure, and the hash function is good enough to distribute them evenly throughout the table (that is, all the n lists contain the same number of words).

How many comparisons, on the average, are needed to access a word in the hash structure? Give your answer in terms of n, and show how you got it. The formula

$$1 + 2 + 3 + \ldots + k-1 + k = \frac{k(k+1)}{2}$$

will be useful. For the purposes of the computation, assume that n evenly divides 25,000.

Each chain in the hash table will be 25000 / N items long. Set k to be 25000 / N.

The average time to find an item in a k-length list is (1 + 2 + 3 + ... + k) / k. Given the formula above, that is (k(k+1)) / 2k = (k+1) / 2. Substitute 25000 / N for k.