

Homework 2

February 11, 2003

Due Date: February 18

1. (10 pts) In this problem we look at hashing. Assume the hash table is called `table` and is of size 7 (i.e. slots 0,1,2,3,4,5,6). Here are the hash values that you will need:
`baseHash(73)=5`, `stepHash(73)=4`, `baseHash(22)=1`, `stepHash(22)=2`,
`baseHash(51)=1`, `stepHash(51)=4`, `baseHash(30)=5`, `stepHash(30)=3`,
`baseHash(45)=5`, `stepHash(45)=1`, `baseHash(47)=5`, `stepHash(47)=6`.
 - (a) Show the result of inserting items with keys of 73,22,51,30,45, and 47 when using chaining to resolve collisions.
 - (b) Show the result of inserting items with keys of 73,22,51,30 (in that order) using open addressing with double hashing to resolve collisions.
 - (c) Suppose that `table[0]`, `table[1]`, `table[3]`, and `table[5]` reference `dictionaryRecords` that are either currently used or deleted. Give the probe sequence (in the order they occur) when searching for the item with a key of 22.
2. Here we consider randomized quicksort.
 - (a) (5 points) Illustrate the operation of the `Partition` algorithm on the array: 15,7,9,12,4,10,8 when partitioning around the 12. Indicate where the swaps occur during the algorithm's execution and show the state of the array after each swap.
 - (b) (5 points) In class we proved that randomized quicksort has expected $\Theta(n \log n)$ complexity where the expectation depends only on the coin flips used within randomized quicksort. How would you need to change this claim if quicksort was modified to always partition around the element in the middle of the subarray being sorted (i.e. element $\lfloor(\text{left} + \text{right})/2\rfloor$). Be very explicit in your answer.
3. (25 pts) This problem is designed to be sure you understand the definition of expected value and can use it to determine the expected number of times a portion of code is executed. There is a practice problem on the webpage like this so you should look at that if you feel like you need more guidance before starting.
 - (a) For this part you are to compute the expected number of array elements examined by a binary search when searching in array $A[0..14]$ for an element x where for $0 \leq i \leq 14$, x is $A[i]$ with a probability of $1/15$. I'll help get you started. The first element examined will be $A[7]$. If $x = A[7]$ (which occurs with probability $1/15$) then the search will end with only 1 array element examined. Else if $x < A[7]$ then binary search will recursively search for x in $A[0..6]$. Else (so $x > A[7]$) binary search will recursively search for x in $A[8..14]$.
Beginning with the definition of expected value, compute the exact value for the expected number of array elements examined in this problem.

- (b) In this part you consider a situation in which calls will be repeatedly made to search for an item in a unsorted list L . Further, you know that items searched for recently are more likely to be searched for again. Thus, to reduce the search time, whenever an item is found it is moved to the front of L . Here's pseudo-code for the search procedure where `ptr.value` gives the value of the item pointed to by `ptr` and `ptr.next` is a reference to the next list item.

```

boolean search(L,x){           \\searches for x in the unsorted list L
    initialize ptr to be a reference to the first item in L
    while (ptr != null && ptr.value != x)           \\searches for x
        ptr = ptr.next;
    if (ptr != null) {           \\if found
        move item referenced by ptr to the front of L
        return true;
    }
    else return false;
}

```

Based on an empirical analysis on this algorithm executing across a large number of previous runs (where n is the number of items in the list) you have been provided with the following table:

list position	1(front)	2	3	...	n	not in list
probability	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{3}{8(n-2)}$...	$\frac{3}{8(n-2)}$	$\frac{1}{8}$

What is the expected number of times that the comparison (`ptr.value != x`) is executed (using the probabilities given in the above table)? You are to required to give an EXACT answer that should not be in the form of a summation. *Be sure to show your work starting from the definition of expected value.*

4. (15 points) In order to have an *extensible* array (such as Java's `ArrayList` that has no upper limit on the size, the most efficient implementation is to double the array size (copying over the elements in use) when the array is full. We study an extensible array here in the context of open addressing but the same basic analysis would hold for Java's `ArrayList` implementation. Suppose that you have a hashtable with open addressing that is initially set to a size of 2. Whenever the number of items stored in the hashtable reaches $1/2$ of the current array size you are to double the array size and must then rebuild the array since the hash values have changed. Assume that there are 2^k elements inserted (and none deleted). Compute the number of re-insertions needed during the doubling and then report the average cost per element of the doubling by dividing the number of re-insertions by the number of elements (2^k). This is an example of amortized analysis in which you periodically do a lot of work (the doubling operation) but we can show that the work per insertion operation is still small. In other words, the cost of each doubling operation is amortized over a large number of insertions.
5. (40 points) For each of the following two situations you are to pick a data structure that is best suited for it and that meets the given specifications. The following components should

be clearly given in each of your solutions. See the course web page for a sample solution for a problem of this type.

- You should very clearly describe your data structure choice, including all decisions about how the data structure is to be applied (e.g. what is used as the key, what is the associated data, what is the table size and collision resolution choice for hashtables...).
- You should clearly describe how each of the provided operations will be implemented AND analyze the efficiency for each of the operations. You only need to describe any new methods or variations of methods covered in class that you need. For the methods covered in class (e.g. insertion into a hashtable) you don't need to describe how it or derive the expected time complexity. You can just specify what standard method you are using (e.g. insertion) along with its parameters (e.g. what is given as the key, and what is given as the associated data).
- Briefly discuss your choice of data structures included the hashtable size and the method for collision resolution.

HINT: You will often find a need when you have two data structures that store related information, to have elements in one data structure hold a reference to the corresponding element in the other data structure. For example, you may find it useful to add a component within the elements (i.e. the DictionaryRecord class from Lab 2) of one hashtable that is a reference to the elements in another hashtable.

- (a) For this problem you want to search a collection of historical events for those that contain specify keywords. Your are to design a data structure to implement the following operations in the specified time. In `insertEvent` the string `description` is assumed to be separated by spaces and you are guaranteed that the search will be for a word within the description. For the purpose of this assignment you need not worry about any searches that may give multiple words or two word combinations.

Operation	Expected Time Requirement
<code>insertEvent(int date, String description)</code>	$O(w)$ where w is the number of words within <code>description</code>
<code>findOccurences(String word)</code>	$O(k)$ where k is the number of occurrences of <code>word</code>

- (b) You have been hired as a consultant by University X. They have an enrollment of s students where s is approximately 25,000. Each student has an id number which is a 9 digit number that is unique for that student. In addition for each student there is a `StudentRecord` that includes a last name (20 characters), a first name (20 characters), a middle initial (1 character), an email address, a mailing address, current courses (initially empty) and a transcript. Each course also has unique id (e.g. E61241SP03 which could then be converted to a unique integer). For each course there is a `CourseRecord` that includes a professor, classroom and class list (initially empty). Each semester, University X teaches approximately c courses where c is approximately 4500. (Note that “ c ” is a variable here, not a constant.) The size of course i is denoted by n_i where $5 \leq n_i \leq 300$.

Finally, you can assume that each student takes no more than 10 courses in a semester. (10 should be viewed as a constant here.)

You are to design a data structure to implement the following operations in the specified *expected* time. From the names of the methods it should be clear what is expected of them. For `printSortedRoster` the roster must be sorted alphabetically by the students' names (and also include their id and email address). If you are uncertain about the semantics of any other `methods` please ask. You should assume that both the `insertStudent`, `removeStudent`, `registerStudent`, and `withdrawStudent` methods are called frequently, whereas the `insertCourse` method is not called very frequently. Also, notice that courses are never removed.

Operation	Expected Time Requirement
<code>insertStudent(int studentID, StudentRecord data)</code>	$O(1)$
<code>removeStudent(int studentID)</code>	$O(1)$
<code>insertCourse(int courseID, CourseRecord data)</code>	$O(1)$
<code>registerStudent(int studentID, int courseID)</code>	$O(1)$
<code>withdrawStudent(int studentID, int courseID)</code>	$O(1)$
<code>printSortedRoster(int courseID)</code>	$O(n_i \log n_i)$ for course i

Challenge Problems:

Only work on these after you have completed the required problems. Note that you can obtain 100% without doing either of these.

1. (3 points)

The following program determines the maximum value in an unordered array $A[0..n-1]$.

```

1  max = -∞
2  (int i = 0; i < n; i++)
3      if A[i] > max
4          then max = A[i]
```

What is the expected number of times the assignment in line 4 is executed? (Assume that all numbers in A are drawn randomly from the interval $[0,1]$.) You may find it useful to let s_1, s_2, \dots, s_n be n random variables, where s_i represents the number of times (0 or 1) that line 4 is executed during the i th iteration of the **for** loop. Since $s = s_1 + s_2 + \dots + s_n$, by linearity of expectations $E(s) = E(s_1) + \dots + E(s_n)$.

2. (2 points) Generalize your work from Problem 3a to compute the expected number of elements examined when the array has $n = 2^r - 1$ elements (for r an integer).