

Homework 2

February 5, 2008

Due Date: February 12

1. (15 points) In this problem we consider when a buffer is going to be maintained under a variety of situations. For each, explain if you would select an array, a circular array, a dynamic array, a dynamic circular array, a singly-linked list, or a doubly-linked list to implement the buffer, and very briefly explain why.
 - (a) (5 points) The buffer is a bounded buffer that is to hold no more than 1000 elements. The only methods that are commonly used are to insert and remove elements from both the front and back of the buffer, and these methods must be as efficient as possible. The only other method needed is the ability to iterate over the buffer. The buffer is expected to hold between 500 and 1000 elements at any given time.
 - (b) (5 points) We consider the variation of part (a) in which the buffer is unbounded (i.e., there is no a prior bound on number of elements that will be placed in the buffer).
 - (c) (5 points) We next consider a different variation from part (a) in which efficient support must be provided to iterate over the buffer removing any elements that satisfy a given property (which can be checked in constant time for each element).
2. (15 points) In order to have a *dynamic array* (such as Java's `ArrayList`) that has no upper limit on the size, the standard implementation is to double the array size copying over the elements in use when the array is full. In this problem you explore the overhead in time complexity per insertion caused by the resizing¹. For this analysis, we'll assume there is a cost of 1 to insert an element at the end of the positional collection when it is not full. We treat the cost to resize a full array of x elements to a half-full array of $2x$ elements as x since x elements must be copied into the new array. (If desired the cost of allocating and initializing the array could also be included. However, copying the elements is the dominant cost.)
 - (a) (8 points) First consider the situation in which the dynamic array is initialized to length 1. Consider the program that inserts 100 elements at the end of the dynamic array. Compute the cost of the overhead associated with resizing (i.e., the cost of allocating new arrays and copying over the elements).
 - (b) (1 points) What is the cost per element inserted into the dynamic array (by the user program) of the overhead associated with resizing the dynamic array?
 - (c) (6 points) Now consider when the dynamic array is initialized to length 32. Now what is the cost per element inserted into the dynamic array of the overhead associated with the dynamic array?
3. (15 points) Suppose the partitioning procedure presented in class for randomized quicksort was applied to partition an n element array around a randomly selected pivot element. What is the expected number of elements to the right of the pivot after the partitioning is complete?

Note: While you could use your intuition to guess the answer for this problem, you are expected to use the formal definition of expected value. Be sure to show your work starting from the definition of expected value!

¹Appendix B.5 discusses *amortized analysis*. Informally, the cost of reorganization is amortized over the public methods executed that caused the reorganization to be performed.

4. (15 points) The below code for binary search assumes that A is sorted using the Comparator `comp` so that $A[i] \leq A[i + 1]$. The call `binarySearch(x, A, p, r, comp)` should return `true` exactly when some element in $A[p], \dots, A[r]$ contains x . The initial call made, in general, would be `binarySearch(x, A, 0, A.length - 1, comp)`.

```

boolean binarySearch(E x, E[] A, int p, int r, Comparator comp)
1  int mid = (p + r)/2;
2  int value = comp.compare(x, A[mid]);
3  if (p == r)
4      return (value == 0)
5  if (value == 0)
6      return true
7  else if (value < 0)
8      return binarySearch(x, A, p, q - 1, comp)
9  else
10     return binarySearch(x, A, q + 1, r, comp)

```

You are to exactly compute the expected number of times that x is compared to an element of A (i.e., that line 2 is executed) when searching $A[0..14]$ where for $0 \leq i \leq 14$, x is $A[i]$ with a probability of $1/30$ and with probability $1/2$, x is not in A .

5. (15 points) The *nuts and bolts* problem is defined as follows. You are given a collection of n bolts of different diameters and n corresponding nuts. You are allowed a **test operation** in which you try a nut and bolt together from which you can determine whether the nut is too large, too small, or an exact match for the bolt. You can not directly compare two nuts or directly compare two bolts. Each test operation takes constant time, and the problem is to match with each bolt to its nut.

Give an algorithm to correctly pair up the n nuts and bolts in expected $O(n \log n)$ time. *You can use the bound we derived on the expected cost of randomized quicksort.*

Extra Credit Problems:

- 6* (2 points) The following program determines the maximum value in an unordered array $A[0..n - 1]$.

```

1  max =  $-\infty$ 
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 $E(s) = E(s_1) + \dots + E(s_n)$.

- 7* (3 points) Generalize your work from Problem 4 to compute the expected number of elements examined when the array has $n = 2^r - 1$ elements (for r an integer) and with probability $1/2$, x is not in the array, and with probability $1/(2n)$, $x = A[i]$ for $i = 0, 1, \dots, n - 1$.