

## Homework 2, CS 520, McConnell, Fall '09, due February 24 before class

1. Derive the worst-case time bounds you get for **Select** if you divide the elements into groups of 7 elements instead of groups of 5 elements.

Do the same for groups of 3 elements.

2. There is an easier way to get the  $O(n)$  expected time bound for **Select** than the one given on page 188.

For the moment, let us assume that all elements are distinct and that  $(z_1, z_2, \dots, z_n)$  are the elements in ascending order. The algorithm has no knowledge of this ordering of the elements, but that doesn't keep us from using this knowledge in our analysis. Suppose **Select** is looking for the element of rank  $r$ , that is, it is looking for  $z_r$ .

Let us define some random variables to help with the analysis.

- For  $\{(i, j) | i < j < r\}$ , let  $X_{ij} = 1$  if  $z_i$  is compared with  $z_j$  at some point during execution of **Select**, and  $X_{ij} = 0$  otherwise.
  - For  $\{(i, j) | r < i < j\}$ , let  $Y_{ij} = 1$  if  $z_i$  is compared with  $z_j$  at some point and  $Y_{ij} = 0$  otherwise.
  - For  $\{(i, j) | i < r < j\}$ , let  $Z_{ij} = 1$  if  $z_i$  is compared with  $z_j$  at some point and  $Z_{ij} = 0$  otherwise.
  - For  $\{i | i \neq r\}$ , let  $R_i = 1$  if  $z_r$  is compared with  $z_i$ , and  $R_i = 0$  otherwise.
- (a) Argue that the expected asymptotic running time of **Select** is bounded by the sum of expected values of all of these variables. (The book makes this argument; paraphrase the main points briefly.)
  - (b) Argue that the sum of expected values of the  $R_i$ 's is  $O(n)$ .
  - (c) Argue briefly that  $E[X_{ij}] = 2/(r - i + 1)$ . (Notice that this is independent of  $j$ .)
  - (d) Argue briefly that for fixed  $i < r$ ,  $\sum_{j=i+1}^{r-1} E[X_{ij}] = O(1)$ .
  - (e) Argue briefly that the sum of expected values of all  $X_{ij}$ 's is  $O(n)$ .  
(A symmetric argument shows that the sum of expected values of the  $Y_{ij}$ 's is  $O(n)$ .)
  - (f) Argue briefly that  $E[Z_{ij}] = 2/(j - i + 1)$ .
  - (g) Argue briefly that for fixed positive  $k$ ,  $(\sum_{\{(i,j) | j-i+1=k\}} E[Z_{ij}])$  is  $O(1)$ .
  - (h) Argue briefly that sum of expected values of all the  $Z_{ij}$ 's is  $O(n)$ .
3. I claimed in class that the van Emde Boas trees have  $O(m)$  space, where  $m$  is the size of the allowed range of integer values. Prove this. One way to do this is by writing a recurrence for the space requirement, which leads to a lot of messy algebra. There is a nicer way that is based on the following observations. For simplicity, assume that the range  $m$  has size  $2^{2^k}$ , and that the base case, which is implemented by a naive method, has size two.
    - The space required by the arrays is bounded by the number of nodes in the tree. Throwing them out won't affect the analysis.

- Perform some surgery on the tree to reorganize it into something that's easier to analyze, as follows. First, go through the subtrees and side trees, replacing each base case with a tree node that has two leaf children. Then, recursively reorganize the subtrees and the side tree. Finally, replace the leaves of the side tree with the roots of the corresponding subtrees.
- (a) Draw a picture of the resulting tree when  $m = 16$ .
  - (b) Argue that the resulting tree has  $O(m)$  nodes, which will give the bound on the size of the original VEB tree.
  - (c) The resulting tree has a certain height  $k$ . Give a big- $\Theta$  bound on  $k$ , and give a simple observation that proves it.
  - (d) The nodes of the side heap now lie above the nodes of the subtrees. In terms of  $k$ , how far down are the roots of the subtrees.
  - (e) Briefly describe a relationship between van Emde Boas tree accesses and binary search on the depth of this new tree. You should make mention of the two cases we considered: we spend  $\Theta(\log \log n)$  time in the side tree or  $\Theta(\log \log n)$  in the subtree, but not both.
  - (f) What relationship does this binary search have to the  $O(\log \log n)$  bound for the heap operations?
4. Let a *sorted list data type* on a set of non-negative integers be an abstract data type that supports the following operations:

Constructor -- constructs an empty set.

boolean Insert(int value) - insert a non-negative integer value that doesn't already occur in the set. Return true if the insertion was legal and successful.

boolean Delete(int value) - delete an integer value that occurs in the set. Return true if the deletion was legal and successful.

int FindNext(int value) - return the smallest element of the set that's greater than the given value, or -1 if no such element occurs in the set. It is not necessary that the given value occur as an element of the set.

int FindPrevious(int value) - return the largest element of the set that's smaller than the given value, or -1 if no such element occurs in the set. It is not necessary that the given value occur as an element of the set.

void Print(): print out the elements in ascending sorted order.

- (a) Briefly describe how to obtain the best bound you can without using a van Emde Boas tree. You must come up with a bound that's  $o(n)$  on each operation except Print(), where  $n$  is the number of elements in the set. (This is a type of

elementary data structure question that's often asked at Google and Microsoft interviews. It suffices to just name a specific data structure that does the job.)

- (b) Tell what bounds you can achieve for `Print` using this data structure.
  - (c) Describe how to modify the van Emde Boas tree so that it implements this data type, with the additional restriction that all elements be in the range from  $i$  to  $j$ , the constructor takes  $O(m)$  time where  $m$  is the range  $m = j - i + 1$ , `Print` takes  $O(\min(n \log \log m, m))$  time, and the other operations take  $O(\log \log m)$  time apiece.
5. Recall the algorithm we talked about in class for finding the *length* of the longest-increasing subsequence of a sequence. We keep an array  $A[]$ , where  $A[i]$  gives the smallest ending value of an increasing subsequence of length  $i$ . We work by induction on the length of a prefix of the sequence:

(8, 2, 5, 3, 1, 6, 7, 4)

A[1]	8	2	2	2	1	1	1	1
A[2]			5	3	3	3	3	3
A[3]					6	6	4	
A[4]						7	7	
A[5]								
A[6]								
...								
A[8]								

When the next element  $j$  of the sequence is considered, we need to find the first element  $A[i]$  of  $A[]$  that is greater than  $j$  and set  $A[i] = j$ . This takes  $O(\log n)$  time by binary search, for a total of  $O(n \log n)$  over all iterations.

Make sure you can explain why the algorithm is correct; this question might appear on an exam.

- (a) The final sequence of values in  $A[]$  in the above example, (1, 3, 4, 7) is not an increasing subsequence. Briefly describe how to add pointers to the elements as they are inserted to  $A[]$  so that a longest increasing sequence, not just the length of one, can be found in  $O(n)$  time. To make it easier for me to grade, illustrate the installation of the pointers on a photo- or handwritten copy of the above illustration.
- (b) Describe how to get an  $O(n \log \log n)$  time bound for finding a longest increasing subsequence in a permutation of the  $n$  integers  $(1, 2, 3, \dots, n)$ . (These are the fastest known bounds for this problem.)
- (c) Prove that in every sequence of  $n^2 + 1$  distinct integers, there is either an increasing subsequence of length  $n + 1$ , or a decreasing subsequence of length  $n + 1$ . *Hint: for fixed  $i$ , what can you say about the sequence of integers that get stored in  $A[i]$ ?*

6. **Programming:** Modify a copy of your VEB heap data structure to obtain the data structure you described in Problem 4c. I won't supply a VEBTest.java; make your own, and if you conform to the specifications, your data structure will work with the menu that I will use for testing.