

**CSE3358 Problem Set 6**  
**Practice Problems**  
**02/22/05**  
**Due 02/25/05**

**Decision trees and lower bounds**

**Problem 1: Merging  $k$  sorted lists**

We have so far considered a number of variations to this problem. Through our study of merge sort, we saw how to merge two sorted lists containing a total of  $n$  elements in  $\Theta(n)$  time. Moreover, Problem Set 2 introduced a modification to merge sort that calls insertion sort on small inputs ( $\leq k$  in size). This modification of merge sort can be viewed essentially as merging  $n/k$  sorted lists of size  $k$  each. We have seen how to do this in  $\Theta(n \log n/k)$  time.

Now we will revisit the problem in a generalized form: We consider having  $k$  lists containing a total of  $n$  elements. We need to merge them into one sorted list. One could generalize the two-way merging to  $k$ -way merging in the following way: We start with an empty list which will eventually contain all  $n$  elements in order. We keep  $k$  pointers, one per list. Each pointer originally points to the first element of the corresponding list. By comparing all  $k$  elements, the smallest among these is chosen and placed in the big list and the corresponding pointer is incremented. This procedure is repeated until all elements are taken. The drawback of this approach is that determining the smallest among  $k$  elements takes  $O(k)$  time, leading to a  $O(nk)$  running time for the merging. A better way is to perform 2-way merging of two lists in a tree-like form as we did previously for the modified merge sort. Here's another way:

(a) Using a heap data structure, describe a  $O(n \log k)$  time algorithm for performing  $k$ -way merging of  $k$  lists containing a total of  $n$  elements.

(b) Show that any algorithm for merging  $k$  sorted lists containing a total of  $n$  elements and that uses those elements in comparisons only, has to run in  $\Omega(n \log k)$  time. To do this, obtain all possible interleavings of  $k$  sorted lists and use a decision tree argument similar to the one we used to obtain the sorting lower bound.

Note: Merge sort is nothing but merging  $n$  lists of size 1 each.

**Problem 2: Yet another variation**

Consider  $k$  lists that are not necessarily sorted containing a total of  $n$  elements. In this variation, all the elements in the first list are less than or equal to all the elements in the second list, and so on... One possible method for sorting the elements is to sort the individual lists independently and then concatenate the sorted results. This will take  $\Theta(\sum_i n_i \log n_i)$  where  $n_i$  is the number of elements in list  $i$ .

*Algorithm OBVIOUS*

```
for  $i \leftarrow 1$  to  $k$ 
  do sort list  $i$             $\triangleright$  for example using merge sort
concatenate the lists
```

Although the above algorithm is OBVIOUS, nothing better can be done. Regardless of what algorithm we use for this variation of the sorting problem, show that  $\Omega(\sum_i n_i \log n_i)$  time is needed. Note that

it is not rigorous to sum the individual lower bounds because an algorithm does not necessarily work like the OBVIOUS algorithm.

Note: If all lists have the same size, this bound will be  $\Omega(n \log n/k)$ .

## Sorting

### Problem 3: Counting inversions

Given an array  $A$ , we define  $(i, j)$  to be an inversion of  $A$  if  $i < j$  and  $A[i] > A[j]$ . Therefore, a sorted array has 0 inversions.

- (a) Find the inversions of  $A = [2, 3, 8, 6, 1]$ .
- (b) What is the maximum number of inversions that an array of length  $n$  can have? Which arrays achieve that maximum?
- (c) Show that the running time of insertion sort on an array  $A[1..n]$  is  $\Theta(n + I)$  where  $I$  is the number of inversions of  $A$ .

### Problem 4: Almost sorted

Consider an array  $A[1..n]$  satisfying the following condition

$$|A[i] - i| \leq k$$

for some positive constant  $k$ .

- (a) Give an algorithm for sorting array  $A$  that satisfies the following properties:
  - the running time of the algorithm is  $\Theta(n)$
  - the algorithm is inplace
  - the algorithm is stable

Note: If the elements are integers, counting sort would run in linear time because  $A[i] \leq i + k \leq n + k = O(n)$ . However, counting sort is not inplace. Therefore, counting sort cannot be used.

- (b) Show that any algorithm to sort  $A$  has to run in  $\Omega(n \log k)$  time.

## Heaps

### Problem 5: Height of a heap

We have been using the fact that the height of a heap of  $n$  elements is  $\Theta(\log n)$ . This is definitely true because a heap is a nearly complete binray tree (i.e. all levels are filled except possibly for the last one which is filled from left to right). Prove this fact by finding both lower and upper bounds on the number of nodes in a heap of height  $h$ .

**Problem 6: Heap sort**

We proved in class that Heapsort runs in  $O(n \log n)$  worst-case time. This is because the running time is dominated by  $n$  HEAPIFY operation and every HEAPIFY runs in  $O(\log n)$  time in the worst case. Show that the worst-case running time of Heapsort is also  $\Omega(n \log n)$ , i.e. for every large  $n$ , there is an input of size  $n$  that would force Heapsort to spend a time at least some constant multiplied by  $n \log n$ .

**Hashing****Problem 7: Worst-case of Bucketsort**

- (a) What is the worst-case running time of Bucketsort?
- (b) How can you modify Bucketsort to maintain its linear expected running time and make its worst-case running time  $O(n \log n)$ .

**Problem 8: Open addressing with uniform hashing**

Consider an open address hash table with uniform hashing. Give upper bounds on the expected number of probes in an unsuccessful search and on the expected number of probes in a successful search when:

- the loading factor is  $3/4$
- the loading factor is  $7/8$
- the loading factor is  $O(1)$ . Note that in an open address hash table the loading factor is always  $O(1)$  because  $n \leq m$ .

**Problem 9: Constructing a universal family of hash function**

In this problem we will look at an easy way for constructing a universal family of hash function. Recall the definition of a universal hash family.

**Universal Hash Family  $H$ :** For any two keys  $k$  and  $l$ , let  $S_{kl} = \{h \in H | h(k) = h(l)\}$ , then  $|S_{kl}| \leq \frac{|H|}{m}$ , where  $m$  is the size of the hash table.

The important consequence of the above definition is that by picking a hash function at random from a universal family of hash functions, the probability that two keys will collide is  $\leq \frac{1}{m}$ . This makes the expected number of keys that collide with a particular key  $k$ , at most  $\frac{n-1}{m}$  (each of the  $n-1$  other keys has a probability  $\leq \frac{1}{m}$  of being in the list). Therefore, the expected length of the list in which  $k$  hashes to is at most  $1 + \frac{n-1}{m} = O(1 + \alpha)$ .

Without loss of generality, let the keys be decimal numbers with  $n$  digits. In other words, every key  $k$  has the form  $k = x_1x_2\dots x_n$  where  $x_i \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ . Let  $m$  be a prime number greater than 10 (we will see why later) and consider the following family of hash functions.

$$h(k = x_1x_2\dots x_n) = \left(\sum_{i=1}^n a_i x_i\right) \bmod m$$

for  $a_i \in \{0, 1, 2, \dots, m-1\}$ .

(a) What is the size of  $H$ ?

(b) What needs to be done to pick a hash function from  $H$  uniformly at random?

Now we want to see if  $H$  is universal. This means we need to verify whether  $|S_{kl}| \leq \frac{|H|}{m}$  for every pair of keys  $k$  and  $l$ . Let's assume that  $k = x_1 \dots x_n$  and  $l = y_1 \dots y_n$  collide. This means

$$\sum_{i=1}^n a_i x_i \equiv \sum_{i=1}^n a_i y_i \pmod{m} \quad k \neq l$$

Since  $k \neq l$  they must differ in at least one digit. Without loss of generality, let's say they differ in the first digit, i.e.  $x_1 \neq y_1$ . Therefore, we can rewrite the above as:

$$a_1(x_1 - y_1) \equiv \sum_{i=2}^n a_i(y_i - x_i) \pmod{m} \quad x_1 - y_1 \neq 0$$

This is where the choice of  $m$  being prime becomes important. From number theory, if  $m$  is prime, then every integer  $z \in \{1, 2, \dots, m-1\}$  has a **unique inverse**  $z^{-1} \in \{1, 2, \dots, m-1\}$  such that  $zz^{-1} \equiv 1 \pmod{m}$ .

(c) Argue that  $x_1 - y_1$  has a unique inverse  $(x_1 - y_1)^{-1}$  such that  $(x_1 - y_1)(x_1 - y_1)^{-1} \equiv 1 \pmod{m}$ .

(d) By multiplying both sides of the equation above by  $(x_1 - y_1)^{-1}$ , show that  $a_1$  is uniquely determined from  $a_2, a_3, \dots, a_n$ .

(e) Using part (d), find the number of hash functions  $|S_{kl}|$  that can cause  $k$  and  $l$  to collide, and verify that  $H$  is universal.