CISC235
Winter 2007
Homework for week 7
in preparation for quiz 3
Solutions

1 If we are compressing a file then the total number of symbols which I will call N, is much bigger than the total number of distinct symbols, n. The time it takes to build the Huffman tree from a Huffman forest using a naive algorithm that uses a sequential search to find the minimum weight tree, has a worst case complexity of $O(n^2)$. However, the number of distinct symbols is often of size 256, that is, the number of distinct symbols that can be stored in an 8 bit byte. Using a heap to build a Huffman tree has a complexity of $O(n \log n)$. With $n = 256$ one would expect that this would be about 8 times faster than the naive algorithm. However, if we are compressing a reasonably large text file say 100,000 characters, the tree building phase makes up a small fraction of the total algorithm.

2 There are many different possible solutions for this. One possibility is to ignore the letters and just compute the remaining number mod table size. Another option is to cast the letters to integers and the use a so called *folding* approach, that is. we group the id as follows;

9 - x - 9 x - x - 99 -x - 9 - x - x - 999999

Then obtain the sum of each group, and add the sums to obtain an overall global sum S. Finally perform S mod tablesize to get a mapping to a hash table location.

3 Using the hash function h(i) = (2i + 5) mod 11, with collisions resolved using linear probing we get:

12 → 7; 44 →5; 13 → 9; 88 → 5 → 6; 23 → 7 → 8; 94 → 6 ... → 10; 11→ 5 ... 10 → 0; 39 → 6 → ... → 1;

And this yields the final table:

0	1	2	3	4	5	6	7	8	9	10
11	39	-	-	-	44	88	12	23	13	94

6.12 - 25 The key to understanding the McDiarmid and Reed algorithm is to observe that in the moveDown phase the parent node is not used in any comparisons, only children are compared and the larger of the two moves up. The parent node moves all the way down no matter what.

Here is pseudo-code for the algorithm.

```
for(int i = n/2 -1; i >=0; i--)
     // Floyd's phase using a variant of moveDown
     tmp = data[i]; j = i;
     while (data[j] is not a leaf)
           swap data[j] with it's larger child;
           j = index of larger child;
     data[j] = tmp;
     //William's phase using moveUp
     while(j > i  and data[j]  has a larger key than its parent)
              swap data[j] with its parent;
              j = index of parent.
```

Here is a cartoon sequence of the algorithm on the given input.

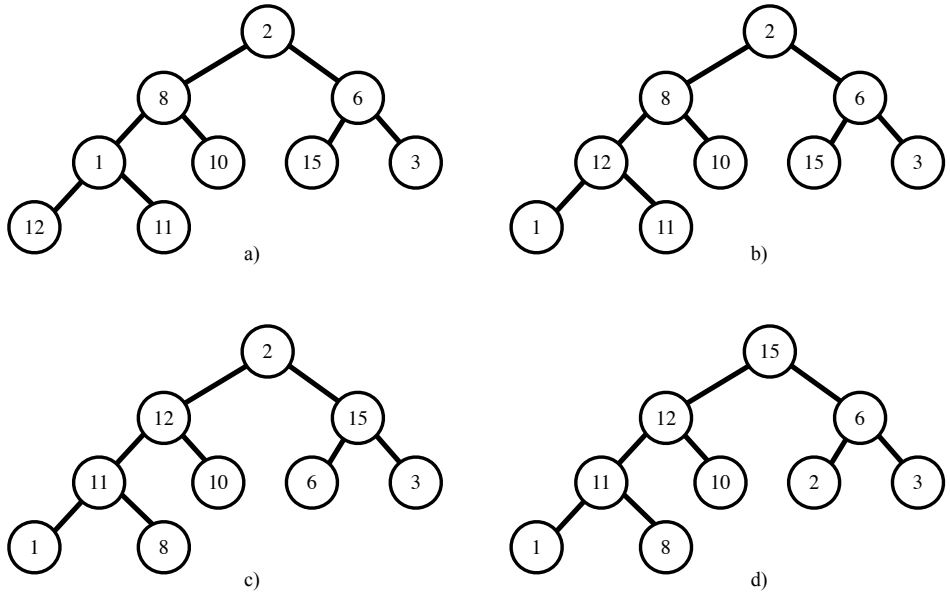

Figure 1: The original data is shown in a) with successive applications of the algorithm. The William's phase never does anything on this data. Nevertheless this is not identical to just running Floyd's algorithm because we never compared a parent to its child.

To obtain a worst case we want to maximize the number of compares and swaps. Floyd's phase swaps down to the leaf nodes no matter what. We need to force the William's phase to swap as often as is possible, that is all the way up to where the

iteration began. If the data is already in heap order then this worst case is achieved. It is not hard to see that this worst case behaviour matches the worst case of Floyd's, that is $O(n)$ for building a heap with n nodes.

9.7 - 1 a) Yes. To check whether one word is an anagram of the other we see if both words use the same symbols. By sorting the symbols first, one subsequent pass can be used to determine if they are identical.

b) No, because the minimal values can be found in one pass of the values without sorting.

c) No, order has nothing to do with averages.

d) Yes, after sorting we traverse to the middle. (Fact: the median can actually be found in linear time without sorting!)

e) Maybe. If the number of distinct symbols is proportional to the number of symbols in the input, then we can first sort the symbols then obtaining counts is easy, because all identical characters are consecutive. On the other hand if the number of symbols in the file is much larger than the number of symbols then we can just maintain a symbol frequency map in the way we did for building a Huffman tree.

10.7 - 1 The fewest number of elements that hash to their home positions, that is a position that uses no collision resolution strategy is 1. An empty hash table must hash its first entry to its home position. All subsequent entries may end up "away from home". For example insert the keys 0, 5, 10, 15, 20 into a five element hash table using the hash function h(key) = key mod tablesize.