

## Task 1:

(1) To rearrange the items in a linked list, the data does not need to "move", but rather the pointer variables can be reassigned. Since pointers have a constant size (irrespective of the size of the data stored in the list) this leads to a very efficient method in practice for rearranging lists. The rearrangement of pointers still needs some scratch space, to hold the values during a reassignment, but this can be kept to a constant (eg 4 or so temporary variables), irrespective of the size of the list.

The merge algorithm itself is still theoretically  $O(n)$  since each item in the list must still be inspected.

(2) As the variables are processed in the merge function they are "removed" from their respective portion of the list and added to the merged list. The two portions of the list may be delimited by variables (eg pf and s for the first half, and ps and s for the second half).

The invariant is that the elements in the merged list are always sorted. On termination, all elements are in the merged list, which is therefore a sort of the original elements.

```
merged list:      () ---> () ---> ... ()      sorted
list elements remaining :   () ---> .. --> () --> .. ().. --> .. ()
                                pf           f           ps           s
```

(Please note, that marks were awarded if there was a description of how the merge function works.)

(3) Many people wrote in general about quicksort and mergesort, and those answers received some credit. The better answers included some specific details about implementing quicksort on linked lists. Some people pointed out the difficulty of finding the mid-point in a linked list for implementing mergeSort, and that this was not a problem for quicksort. Others pointed out that if the partition function were implemented in quicksort (in the way given in lectures for array-based partition converted to linked lists) then there might be some difficulty in making the recursive calls unless a doubly-linked list were used.

(4 & 5) These questions were well answered by those who attempted them.

## Task 2:

(1) Shuffle: A random number is assigned to each shuffle tag, and the list is sorted according to the shuffle tag. This uses the  $O(1)$  scratch and  $O(n \log(n))$  sort function developed in Task 1;

SmartShuffle: Shuffles the list and then sorts the result according to played. It uses 2 sorts.

(2) There were quite a few good suggestions to this question. The best ones suggested assigning a number which was both random and based on the played tag, so that it would be more likely for items with larger played count to have a high number assigned to the shuffle tag. The best answer of all suggested assigning to the shuffle tag

$$\text{random}(0..n-1) + \text{playedCount} * n$$

As this would guarantee that the items with playedCount =0 would have a number set in the range  $(0..n-1)$ ; those with playedCount =1 would have a number set in the range  $(n..2*n-1)$ ; playedCount =2 would have a number set in the range  $(2*n..3*n-1)$ , and so on, thus the random number would naturally be partitioned according to played count, and randomised

within that. Overall this used exactly 1 sort.

All such suggestions were awarded 1 mark.

(3) This was asking about a specific C++ suggestions. The most popular answer suggested combining the three variations of the sorting function into 1 eg using something equivalent to a template. These and other suggestions received the mark.

(4) This is not a perfect way to produce random sequences, as many of you pointed out and were able to demonstrate by carefully tabulating sets of results. Weak randomisation is observed when two items are given the same random number (and this has quite a high probability, compare for example the birthday paradox). Since mergesort is a stable sort this means that in the case where two items are given the same random number, then they will certainly not be rearranged.

However this kind of shuffling is probably ok for iTunes.

Please note: The marks released on Blackboard for Task 1 and Task 2 refer to the written parts only of the assignments. Note also I am still working through some of the work which was submitted on-line --- if you were one of those people then you should have your mark by the end of the week.

AM

4/5/09