Aditya Shirode, Rhythm Shah, Shrenuj Gandhi

7. [6 points] *Purpose: understanding merge sort; demonstrating abstract problem solving ability needed in CSC 505; also another analysis invloving inversions*

   (a) Modify MERGE-SORT so that it returns the number of inversions in its input (array or list). Prove that your algorithm is correct. This is much easier if you use the linked-list version and recursion invariants (see lecture notes titled *recursive list algorithms* on the Moodle site).

   (b) Suppose there are no inversions in the original list, i.e., the listed is sorted to begin with. Exactly how many key comparisons does MERGE-SORT do in this case? What if the list is in reverse order? You may assume that n, the number of elements, is a power of 2 to get an exact answer.

   4 points for part (a): 3 for an algorithm that works, 1 for correctness, 2 points for part (b)

Answer:

(a) Building on the code given in class notes from *Dr. Stallman*'s slides:

```
 1 def mergesort(L):
 2   if L is empty: return L
 3   elif rest(L) is empty: return first(L)
 4   else: return Merge( MergeSort(oddHalf(L)), MergeSort(evenHalf(L)))
 5
 6 def merge(L1, L2):
 7   if L1 is empty: return L1
 8   elif L2 is empty: return L1
 9   elif first(L2) < first(L1):
10          noofinversions += 1
11          return first(L2) + merge(L1, rest(L2))
12   else: return first(L1) + merge(rest(L1), L2)
13
14 def oddHalf(L):
15   if L is empty: return L
16   elif rest(L) is empty: return [first(L)]
17   else: return first(L) + oddHalf(rest(rest(L)))
18
19 def evenHalf(L) is
20   if L is empty: return L
21   elif rest(L) is empty: return []
22   else: return first(rest(L)) + evenHalf(rest(rest(L)))
```

We have added *Line* 10 in the given code, which adds 1 to the global variable $noofinversions$ (number of inversions), because in that for loop, index of $L1 <$ index of $L2$, but value of $L1 >$ value of $L2$, which is an *inversion*.

Consider the example: $[1, 5, 4, 2, 6, 3]$
Initially, the list will be divided into the odd and even halves $[1, 4, 6]$ and $[5, 2, 3]$ respectively.
Recursively they both will be divided further into the lists $[1, 6]$, $[4]$,$[5, 3]$ and $[2]$.
On applying the merge function, we get $[1, 6]$, $[4]$, $[3, 5] and [2]$, then recursively we get $[1, 4, 6]$, $[2, 3, 5]$ and finally $[1, 2, 3, 4, 5, 6]$.

As can be seen, the comparison $first(L2) < first(L1)$ holds true 6 times
(when comparing (6,4), (5,3), (3,2), (4,2), (4,3), and (6,5)), which is also the number of inversions which are (5,2), (4,2), (5,3), (4,3), (6,3) and (5,4).
From the example, we can conclude that the algorithm is correct.

(b) Carefully observing the Mergesort algorithm, we can say that,
# comparisons for sorted list and reverse order list is the same.
and
# comparisons at any level $= \frac{\#instances}{2} * [(2 * instancesize) - 1]$ with the assumption that $n = 2^k$

Thus we have

level	# instances	instance size	total comparisons
0	n	1	$\frac{n}{2}(2 * 2^0 - 1$
1	$n/2$	2	$\frac{n}{4}(2 * 2^1 - 1)$
2	$n/4$	4	$\frac{n}{8}(2 * 2^2 - 1)$
.	.	.	.
i	$n/2^i$	2^i	$\frac{n}{2^{i+1}}(2 * 2^i - 1)$
.	.	.	.
k	$n/2^k = 1$	2^k	0

From the above table we can write

# comparisons = C(n) $= \sum_{i=0}^{k-1} \frac{n}{2^{i+1}}(2 * 2^i - 1)$

$$= n(lgn - 1) - \frac{n}{2}\sum_{i=0}^{k-1} \frac{1}{2^i} \qquad \text{as } k = lgn$$