Tuesday 7/22: Algorithms Pt.2

Recursive pseudocode analysis

Consider the following algorithm that merges two lists into one:

```
01  merge(L1,L2: sorted lists of real numbers)
02          if (L1 is empty)
03                  return L2
04          else if (L2 is empty)
05                  return L1
06          else if (head(L1) <= head(L2))
07                  return cons(head(L1),merge(rest(L1),L2))
08          else
09                  return cons(head(L2),merge(L1,rest(L2)))
```

Note that this is a recursive function, with base cases in lines 02-05, and recursive cases in lines 06-08. To analyze the overall running time, we should first **understand its recursive definition with respect to time complexity**.

Because base cases take fixed amount of work, we have $T(1) = c$.

Now for the recursive step. Lines 06-07 and 08-09 essentially do the same thing with symmetric inputs, and they have the same recursive structure—making one recursive call with input size minus 1. Therefore, we have $T(n) = T(n-1) + d$ for recursive levels, where n represents the sum of length for L_1 and L_2, and d represents some constant work to construct the return value.

With the recursive definition $T(1) = c$ and $T(n) = T(n-1) + d$, we can then **use unrolling to find the closed form**, which is $nd + c$. This suggests that this algorithm takes $O(n)$ time.

Let's take a look at another example:

```
01  mergesort(L = a1, a2, ..., an: list of real numbers)
02          if (n = 1) then return L
03          else
04                  m = ⌊n/2⌋
05                  L1 = (a1, a2, ..., am)
06                  L2 = (am+1, am+2, ..., an)
07                  return merge(mergesort(L1),mergesort(L2))
```

Again, let's try constructing a recursive definition of this function regarding time needed.

Line 02 is the base case and it takes constant time, so we have $T(1) = c$.

Lines 03-07 are for the recursive case. There are a few things happening here. Line 04 calculates m, which should be $O(1)$ time. Lines 05-06 are dividing L in half, which should take $O(n)$ time, whether we are using linked lists or arrays (think about why for your own practice). Then line 7 makes two recursive calls with input size $m = n/2$ (we can ignore the rounding here as it will not affect the analysis), and it calls *merge* with two sorted lists whose total size is n. Therefore, the time spent for each recursive level should be:

$$T(n) = 2T(n/2) + dn + e$$

.

Because dn is $O(n)$ and e is $O(1)$, we can safely ignore the asymptotically smaller term and simplify $T(n)$ as $2T(n/2) + dn$

Now with $T(1) = c$ and $T(n) = 2T(n/2)+dn$, we can unroll and find the closed form of $dn * \log n$ which is $O(n \log n)$.