Problem Set 1

All parts are due Friday, September 13 at 6PM.

Name: Your Name

Collaborators: Name1, Name2

Problem 1-1.

(a) $\lim_{n\to\infty} \frac{nlog_2 n}{log_2 n^{2019}} = \lim_{n\to\infty} \frac{(nlog_2^n)'}{(log_2 n^{2019})'} = \lim_{n\to\infty} \frac{log_2 n + 1}{\frac{2019 log_2^{2018} n}{n}} = \lim_{n\to\infty} \frac{n(log_2 n + 1)}{2019 log_2^{2018} n} = \infty \Rightarrow$
$nlog_2 n = \Omega(log_2 n^{2019})$

$(f_1, f_5, f_2, f_3, f_4)$

(b) $\lim_{n\to\infty} \frac{f_3}{f_4} = \lim_{n\to\infty} \frac{n^{6006} log_2 n}{log_2(6006^{n^{6006}})} = \lim_{n\to\infty} \frac{(n^{6006} log_2 n)'}{(log_2(6006^{n^{6006}}))'} = \lim_{n\to\infty} \frac{6006 n^{6005} log_2 n + n^{6006}\frac{1}{n}}{6006 n^{6005} log_2 6006} =$
$\lim_{n\to\infty} \frac{n^{6005}(6006 log_2 n + 1)}{6006 n^{6005} log_2(6006)} = \lim_{n\to\infty} \frac{6006 log_2 n + 1}{6006 log_2(6006)} = \infty \Rightarrow n^{6006} log_2 n = \Omega(log_2(6006^{n^{6006}}))$

$(f_1, f_2, f_4, f_3, f_5)$

(c) $(f_5, f_1, f_4, f_2, f_3)$

(d) $f_5 = \binom{n}{3} = \frac{n!}{3!(n-3)!} = \frac{1}{6} n(n-1)(n-2)$
$f_1 = O(n^n)$

$\lim_{n\to\infty} \frac{f_1}{f_3} = \lim_{n\to\infty} \frac{2^n}{\binom{n}{\frac{n}{2}}} \sim \lim_{n\to\infty} \frac{2^n}{\frac{2^n}{\sqrt{\frac{\pi n}{2}}}} = \infty \Rightarrow f_1 = O(f_3)$

$\lim_{n\to\infty} \frac{f_2}{f_5} = \lim_{n\to\infty} \frac{n^3}{\binom{n}{3}} = \lim_{n\to\infty} \frac{n^3}{\frac{1}{6} n(n-1)(n-2)} = \lim_{n\to\infty} \frac{n^3}{\frac{1}{6}(n^3 - n^2 + 2n)} = \lim_{n\to\infty} \frac{6\frac{n^3}{n^3}}{1 - \frac{n^2}{n^3} + 2\frac{n}{n^3}} =$
$\lim_{n\to\infty} \frac{6}{1 - \frac{1}{n} + 2\frac{1}{n^2}} = 6 \Rightarrow f_2 = \Theta(f_5)$

$(f_2, f_5, f_3, f_1, f_4)$

Problem 1-2.

(a) Add a pointer to the tail node i.e., the last node of the linked list. When inserting a new node, we can search the current last node and link a new node to it. After that switch the tail node on the inserted node. This operation takes O(1) in the worst-case.

(b) Add a pointer to each node will refer to a parent node. So during delete_last operation, the last node stores a pointer to the previous node, we need to re-link the tail pointer to the tail's parent node. This operation performs in O(1).

(c) Allocate static array with $2 * N$ size and start filling it from N index. The head and tail variables will track the positions of the beginning and ending of the array respectively. Thus, insertion and deletion can be performed in amortized O(1) time.

Problem 1-3.

(a) This algorithm deletes the first node in the list and stores the returned value in a variable, then inserts that variable to the end of the list. Since delete_first(1) and insert_last(1) each performs in O(1) and are repeated k times, the overall running time for this algorithm is O(k).

(b) This algorithm deletes the first and the end node of the sequence and stores each of them in own variable. It then inserts then in the opposite order to the sequence in O(1) time.

Problem 1-4.

(a) A possible algorithm is one that first finds the midpoint node (the node that would represent the last kid in the first half of the original line) by iterating through the first n nodes in the original linked list, given that there are 2n kids total. We then store this middle node into a variable called middle. Now that we know where the end of the first line is, we know what the reverse order of the second line of kids is because middle points to what will become the last kid in the second line. We want to reverse the directions that the original pointers for the nodes in the second line are pointing in so that in the end, what was originally directly after middle is now the last element in the linked list, and middle is now pointing to the original last element of the list. This is done by iterating through all of the second-half nodes and setting their pointers to the opposite node (except for the first node right after middle, which will become the last node in the list and will not point to anything else) until you've reached the end of the original list and cannot iterate through any more nodes because they don't exist. Now, we reverse the pointer directions of the nodes that were not taken care of in the loop do to null-checking (to make sure that we don't end up trying to get a next node that doesn't actually exist in the loop) and set middle to point to the original last node. We will need three temporary variables to store the chunk of the list that we are currently looking at as we iterate through the list because otherwise, when we reverse the direction of the pointer, we lose track of which node to move on to next. Having a third stored node directs us to the next node to shift our frame of reference to. The running time of this algorithm is O(n): finding middle takes O(n) time because you must iterate through n nodes out of a total of 2n to get to the middle. Setting temporary variables are each O(1). Each step within the while loop also takes O(1) time, but these are all repeated n times for the n kids in the second half of the line for a total of O(k). The last two steps each take O(1) time. Therefore, overall, the running time is O(k), since we have 2*O(k) which is the same as O(k) asymptotically.

(b) Submit your implementation to `alg.mit.edu`.