Problem Set 1

Name: Akshay Raman

Problem 1-1.

(a) $\{f_5, f_3, f_4, f_1, f_2\}$. Using logarithm and exponential rules, we can simplify the functions. $f_1 = \Theta(n \log n)$, $f_2 = \Theta((\log n)^n)$, $f_3 = \Theta(\log n)$, $f_4 = o(n)$, $f_5 = \Theta(\log \log n)$

(b) $\{f_1, f_2, f_5, f_4, f_3\}$. Convert all exponents to the same base (2), $f_1 = \Theta(2^n)$, $f_2 = \Theta(2^{(\log 6006)^n})$, $f_3 = \Theta(2^{6006^n})$, $f_4 = \Theta(2^{(\log 6006)2^n})$, $f_5 = \Theta(2^{(\log 6006)n^2})$.

(c) $\{\{f_2, f_5\}, f_4, f_1, f_3\}$. Using sterling's approximation and formula for n choose k. $f_1 = \Theta(n^n)$, $f_2 = \Theta(n^6)$, $f_3 = \Theta(\sqrt{n}\left(\frac{6}{e}\right)^{6n} n^{6n})$, $f_4 = \Theta((6/5^{5/6})^n/\sqrt{n})$, $f_5 = \Theta(n^6)$

(d) $\{f_5, f_2, f_1, f_3, f_4\}$ Exponent term dominates the base. Can take logarithm of all function to understand better. $f_1 = \Theta(n^{n+4})$, $f_2 = \Theta(n^{7\sqrt{n}})$, $f_3 = \Theta(n^{6n})$, $f_4 = \Theta(7^{n^2})$, $f_5 = \Theta(n^{12} n^{1/n})$

Problem 1-2.

(a) Thinking recursively, to reverse a sub-sequence with k elements, we can swap the ends i.e. elements at indices i and $i + (k - 1)$, a recursively solve the sub-problems. For the base case $k < 2$, no work needs to be done. The correctness of the algorithm can be proved using induction.

The swap can be done by removing the elements in the reverse order (right end then left end). This will preserve the index values while deleting. Then we can insert in the correct order (left end then right end). So make use of index values from before.

Each swap performs four $O(\log n)$-time operations, so it happens in $O(\log n)$ time. At most, $k/2$ recursive calls are made which is $O(k)$. Therefore, the running time of the algorithm is $O(k \log n)$

```
REVERSE(D, i, k):
        if k < 2:
                return None
        xr = D.delete_at(i+k-1)
        xl = D.delete_at(i)
        D.insert_at(i, xr)
        D.insert_at(i+k-1, xl)
        REVERSE(D, i+1, k-2)
```

(b) Thinking recursively, to move a sub-sequence with k elements, we can move the first item at index i in front of index j and then recursively move the sub-sequence of size $(k - 1)$ in front of that. For the base case $k = 0$, we don't have to move anything. If we maintain that: i is the starting element of the subsequence, j is index of the item in front of which we have to place subsequence, and k is the size of the subsequence, we have prove that algorithm works correctly by induction.

After removing an item at index i. If $j > i$, the value of j decreases by 1 i.e. $j = j - 1$. Also, when we insert an item after index j and $j < i$, the entire subsequence shift to the right by one i.e. $i = i + 1$.

The recursive procedure make no more than $O(k)$ recursive call. In each call, it does $O(\log n)$ work. Therefore, the running time of the algorithm is $O(k \log n)$.

```
MOVE(D, i, k, j):
        if k < 1:
                return None
        x = D.delete_at(i)
        if (j>i):
                j -= 1
        D.insert_at(j+1)
        j += 1
        if (j<i):
                i+=1
        MOVE(D, i, k-1, j)
```

Problem 1-3.

Problem 1-4.

(a)

(b)

(c)

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