CS483 - Practice

Jana Košecká

1. **Chapter 5**, Problem 2

   We will design a recursive divide and conquer algorithm for counting significant inversions. The main difference will be that in the merge stage we merge twice; first merge $b_1, \dots b_k$ with $b_{k+1}, \dots b_n$ just for sorting and then we merge $b_1, \dots b_k$ with $2b_{k+1}, \dots 2b_n$ for counting significant inversions. In the merge step the ALG returns N_1 and $b_1, \dots b_k$ and N_2 and $b_{k+1}, \dots b_n$ which are sorted and N_1 and N_2 are the numbers of significant inversions. Then we need to compute the number of significant inversion N_3 and returns $N_1 + N_2 + N_3$.

2. **Chapter 5** , Problem 3.

   This problem can be solved using divide and conquer. Suppose that you divide the cards into two equal $n/2$ parts and call this algorithm recursively on both sides. In order to have a majority (more then n/2) cards which are equivalent, if we split the away into two subarrays A and B, each having $n/2$ cards. In order for the final set to have more then $n/2$ cards, one of the subsets has to have more then half of the cards which are equivalent (if both would have less then half ($n/4$) then if could never add up to $n/2$). Take that set which has the majority (e.g. A) return one card and check that card against the set B - that will take $O(n)$. If A does not have a majority, try if the B has majority, if yes then pick a card from set B and check it against the set A. Now that portion of the algorithm for checking whether set has a majority of the cards which are equivalent will be run recursively on both sets, i.e. in the worst case you try it on A and if it does not have majority, you will need to run it on the set B, so the recurrence is $T(n) = 2T(n/2) + 2n$ for two subproblems and then for checking the rest of the array.

3. **Chapter 6**, Problem 17

   a) Consider sequence 1,4,2,3. The greedy algorithm produces rising trend 1,4 while the optimal solution is 1,2,3.

   b) Let $OPT(j)$ be the length of the longest increasing subsequence on the set $P[j] \dots P(n)$ including element $P[j]$. $OPT(n) = 1$ and $OPT(1)$ is the length of the longest rising period.

   Consider $OPT(j)$; its first element is P(j) and its next element is $P[k]$ for some $k > j$ and for $P[k] > P[j]$. From k onwards it is simply the longest sequence that starts with $P[k]$. Hence we have a following recurrence

   $$OPT(j) = 1 + \max_{k>j:P[k]>P[j]} OPT(k)$$

   OPT can be build in the decreasing order and the total running time is $O(n^2)$.

4. Suppose that you have two strings on length m and n.

   a) What is the worst case running time of a brute force algorithm for finding longest common subsequence of the two strings ?

   $O(m2^n)$. There are 2^n possible subsequences on X and for each of them it takes $O(m)$ steps to compare the two strings.

   b) Come up with a dynamic programming solution and demonstrate your algorithm on a following example $X = XMKJYWZ$ and $Y = MZJAWXU$.

   Consider $LCS(X_i, Y_j)$ be the length of the LCS of strings $x_1, \dots x_j$ and $y_1 \dots y_j$. The recursive solution is defined by the following 3 cases :

$LCS(X_i, Y_j) = 0$ if $i = 0$ and $j = 0$.
$LCS(X_i, Y_j) = LCS(X_{i-1}, Y_{j-1}) + 1$ if $x_i = y_j$.
$LCS(X_i, Y_j) = \max(LCS(X_i, Y_{j-1}), LCS(X_{i-1}, Y_j))$ if $x_i \neq y_j$.

Set up $m \times n$ table similar to the sequence alignment problem and fill it top down, left to right. Running time will be $O(mn)$.