Solution for Problem Set 4

Mianzhi Pan, 181240045

October 15, 2020

1 Problem 1

(a) Bob's idea can only prove that there is no faster algorithm which can sort the subsequences **independently**, which varies with the origin problem.

(b) Use the decision tree, each subsequence has $k!$ permutations, the origin sequence then has $k!^{n/k}$ permutations, so the decision tree will have $k!^{n/k}$ leaf nodes. The number of comparisions is at most the height of the tree, which is

$$h = lg(k!^{n/k}) = (n/k)lg(k!)$$

Notice that

$$lg(k!) \geq lg(k \cdot (k-1) \cdots \frac{k}{2}) \geq lg(\frac{k}{2}^{k/2}) = \frac{k}{2}lg\frac{k}{2}$$

So

$$h \geq \frac{n}{2}lg\frac{k}{2} = \frac{n}{2}lgk - \frac{n}{2}$$

Hence, number of comparisions is $\Omega(nlgk)$.

2 Problem 2

(a) Similar to $BucketSort$, we allocate an array A with length $k+1$ and initialize all its elements to be 0. Then for every a among the given n integers, we do $A[a]++$. After that we obtain an array A whose element $A[m]$ is the number of m in the given integers. Then create the prefix sum array B based on A, where $B[i] = A[0] + \cdots + A[i]$. Therefore, the number of integers falling into range $[a, b]$ is $B[b] - B[a-1]$.

(b) Since the total number of digits over all the integers in n, number of digits for each integer must be no more than n. We create n 'buckets' and put integer of l digits to the l^{th} bucket, then sort the integers in each bucket and combine all the buckets at last(Notice integers of more digits are always greater than those of fewer digits).

3 Problem 3

(a) An n-element is k-sorted, we have

$$\begin{aligned}
\sum_{j=i}^{i+k-1} A[j] &\leq \sum_{j=i+1}^{i+k} A[j] \\
A[i] + \sum_{j=i+1}^{i+k-1} A[j] &\leq \sum_{j=i+1}^{i+k-1} A[j] + A[i+k] \\
A[i] &\leq A[i+k]
\end{aligned}$$

for all $1 \leq i \leq n-k$.

(b) Use the conclusion in part (a), we split the array into k parts: $A[1, 1+k, 1+2k, \cdots]$, $A[2, 2+k, 2+2k, \cdots]$,$\cdots$,$A[k, 2k, \cdots]$. Sort them independently, notice sorting each of them takes $O((n/k)lg(n/k))$ time, hence the time complexity is $O(nlg(n/k))$.

(c) Use the conclusion in part (a) again, we can extract k sorted subarrays $A[1, i+k, i+2k, \cdots]$, $A[2, 2+k, 2+2k, \cdots]$,$\cdots$,$A[k, 2k, \cdots]$, and all we need to do is $MERGE$ these subarrays. We group these k subarrays into $k/2$ groups(each group contains 2 subarrays) and do $MERGE$ in each group similar to that in $MERGESORT$. Continue this until all subarrays are merged into one array. Notice we should do lgk times $MERGE$ and there are n elements totally, time complexity is $O(nlgk)$.

(d) The lower bound of sorting each of the k parts is $\Omega((n/k)lg(n/k))$, then the toatal lower bound is $\Omega(nlg(n/k))$. Use the conclusion in part (b), we can deduce that the total lower bound is $\Theta(nlg(n/k))$. Notice k is a constant, the lower bound is $\Omega(nlgn)$.

4 Problem 4

Divide the n elements into $n/2$ pairs and do comparision in each of them. Extracting the smaller one in every pair, we repeat the same operation on these $n/2$ elements and we finally get the smallest one $min1$. The number of comparisions during this step is $1 + 2 + 2^2 + \cdots + 2^{lgn-1} = n - 1$.

Mention that we should record all elements being compared during every recurrence in the first step, we extract all lgn elements being compared with $min1$ and find the smallest one $min2$ among them by the oridinary method, which needs $lgn - 1$ comparisions. And $min2$ is the second smallest one. What's more, the total comparision number is $n - 1 + lgn - 1 = n + lgn - 2$.

5 Problem 5

(a) Groups of 7: Follow the analysis in $section 9.3$ of CLRS, suppose the median-of-median is x, then the number of elements greater than x is at least

$$4(\lceil\frac{1}{2}\lceil\frac{n}{7}\rceil\rceil - 2) \geq \frac{2n}{7} - 8$$

Similarity, at least $2n/7 - 8$ elements are less than x. Thus, in the worst case, step 5 calls $SELECT$ recursively on at most $n-(2n/7-8) = 5n/7+8$ elements. We can therefore obtain the recurrence

$$T(n) \leq \begin{cases} O(1), & n < n_0 \\ T(\lceil n/7\rceil) + T(5n/7+8) + O(n), & n \geq n_0 \end{cases}$$

Assuming that $T(n) \leq cn$, then for $n < n_0$, this holds for some suitable large constant c. For $n \geq n_0$, we have

$$\begin{aligned} T(n) &\leq c\lceil n/7\rceil + c(5n/7+8) + an \\ &\leq cn/7 + c + 5cn/7 + 8c + an \\ &= 6cn/7 + 9c + an \\ &= cn + (-cn/7 + 9c + an) \\ &\leq cn \end{aligned}$$

for all $c \geq \frac{7an_0}{n_0-63}$. Hence the algorithm works now.

Groups of 3: Analyse the problem as the above, we have

$$T(n) \leq \begin{cases} O(1), & n < n_0 \\ T(\lceil n/3\rceil) + T(2n/3+4) + O(n), & n \geq n_0 \end{cases}$$

For $n \geq n_0$

$$\begin{aligned} T(n) &\leq cn/3 + c + 2cn/3 + 4c + an \\ &= cn + 5c + an \\ &\leq cn \end{aligned}$$

this holds when $c < -\frac{an}{5}$, notice that c must be positive, so $T(n) \leq cn$ cannot holds. Hence, the algorithm doesn't work when elements are divided into groups of 3.

(b) Use the $QUICKSELECT$ to find the median m_1 among the n elements.(Since n is even, we choose the smaller one in the two medians). Then we divide the origin sequence into two subsequences, one of which contains elements no bigger than m_1, another consists of elements larger than m_1. Do the same operation on both of the subsequences and return the medians $m_2, m_3, \cdots, m_{k-1}$ until the length of subsequence is equal to n/k. Therefore, $m_1, m_2, \cdots, m_{k-1}$ are the k^{th} quantiles of a set.

Notice we will do $QUICKSELECT$ for lgk times and every time it should go through all n elements. The time complexity is $O(nlgk)$.

6 Problem 6

Use $SELECT$ to select the median of medians x, place the elements smaller than x in one set A and the ones greater than x in another set B. If the both $SUM(A.weights)$ and $sum(B.weights)$ are less than $\frac{1}{2}$, then x is the weighted median. Otherwise, the median weight must stays in the set whose sum of weights is greater than $\frac{1}{2}$. We just need to do the above operation on that set recursively.

7 Problem 7

Proof: The first node in the pre-order set must be the root of the binary tree, and that node must divide the in-order set into two subsets. The left subset contains all nodes of the left sub-tree and the right one contains all nodes of the right sub-tree. And we can do this recursively and finally build a tree. Notice the split is unique every time, we cannot build binary trees more than one.

Algorithm: Assume the set containing pre-order numbers is P and that containing in-order numbers is I, and both of them have length n.

① Obviously, $P[1]$ must be the root node. Search $P[1]$ in I, and it will split I into two subsets I_1 and I_2, the former contains all numbers in left sub-tree and the latter contains all numbers in right-subtree.

② Then we begin to build the left sub-tree with I_1. Notice $P[2]$ is the left-child of the root node, i.e. $P[2]$ is the root of the left sub-tree. We repeat step ① and we will obtain another two sets containing the left sub-tree and right sub-tree of $P[2]$. If the left(right) set is empty, $P[2]$ doesn't have left(right) child node. If there exists set whose length is 1, then it must be leaf node. Continue this operation recursively until all numbers in I_1 are placed in the proper position of the tree(Mention that we choose the 'root node' from P in order in every recurrence, i.e. $P[2], P[3], P[4], \cdots$).

③ Similar to step ②, we can build the right sub-tree with I_2.