CS/ECE 374 FALL 2018	Zhe Zhang (zzhan157@illinois.edu)
Homework 4 Problem 3	Ray Ying (xinruiy2@illinois.edu)
	Anqi Yao (anqiyao2@illinois.edu)

3. It is common these days to hear statistics about wealth inequality in the United States. A typical statement is that the the top 1% of earners together make more than ten times the total income of the bottom 70% of earners. You want to verify these statements on some data sets. Suppose you are given the income of people as an n element *unsorted* array A, where $A[i]$ gives the income of person i.

   (a) Describe an $O(n)$-time algorithm that given A checks whether the top 1% of earners together make more than ten times the bottom 70% together. Assume for simplicity that n is a multiple of 100 and that all numbers in A are distinct. Note that sorting A will easily solve the problem but will take $\Omega(n \log n)$ time.

   (b) More generally we may want to compute the total earnings of the top α% of earners for various values of α. Suppose we are given A and k numbers $\alpha_1 < \alpha_2 < \ldots < \alpha_k$ each of which is a number between 0 and 100 and we wish to compute the total earnings of the top α_i% of earners for each $1 \leq i \leq k$. Assume for simplicity that $\alpha_i n$ is an integer for each i. Describe an algorithm for this problem that runs in $O(n \log k)$ time. Note that sorting will allow you to solve the problem in $O(n \log n)$ time but when $k \ll n$, $O(n \log k)$ is faster. Note that an $O(nk)$ time algorithm is relative easy. *Hint:* Use the previous part with $\alpha_{k/2}$ first and then use divide and conquer.

   You should prove the correctness of the second part of the problem. It helps to write a recursive algorithm so that you can use induction to prove correctness.-

---

Solution:

(a) The solution is to find the top 1% and bottom 70% of the elements in A by QUICKSELECT, which requires constant time and iterate over A to sum up the incomes. The algorithm is as following:

- Find the index i of the $99\% * n + 1$ ranked element by QUICKSELECT.
- Find the index j of the $70\% * n$ ranked element by QUICKSELECT.
- Iterate over the array A and store the elements greater than $A[i]$ in another array A_top, and store the elements smaller than $A[j]$ in array A_bottom.
- Sum up the elements in A_top to get Sum_top and sum up the elements in A_bottom to get Sum_bottom.
- Return $A\_top > 10 * A\_bottom$.

The running time for QUICKSELECT is constant, i.e., $O(1)$,the time for iterate over A is $O(n)$, summing up the new array takes $O(n)$, and comparison simply costs $O(1)$. Thus overall the algorithm takes $O(n)$.

(b) The idea is to rank $\alpha_i s$ from 1 to k, and use the divide and conquer method on this ranked array to find out the top α_i% of A. The algorithm is as following:

- Rank $\alpha_i s$ from 1 to k, and let S denote the array of $\alpha_i s$.
- Use QUICKSELECT to find the median m of S, and use QUICKSELECT again to find the m_{th} ranked element in A. Then partition the A by pivot $A[m]$ to get A_l and A_r where A_l stores element smaller than $A[m]$ and A_r stores element greater than $A[m]$.
- Recursively divide the array A_l and A_r into subarrays like in previous step.Also divide S into S_l and S_r by $S[m]$ in the similar way. Let sum_i denote the sum of the elements in each subarray. sum_total is the sum of all subarrays and it is the value to return.

The function can be represented as follows:

```
S <- ranked array of α_i s

SUM_UP(A, S):

    if(len(R) == 0) return
    m <- QUICKSELECT(r, ⌈len(R)/2⌉)
    a_m <- QUICKSELECT(A, m)
    (A_l, A_r) <- PARTITION(A, a_m)
    Return (total_sum − SUM(A_l))
    S_l <- S[1, ..., m − 1]
    Sr <- S[m, ..., len(R)]
    for i in S_r: i < −i − m
    SUM_UP(A_l, R_l)
    SUM_UP(A_r, R_r)
```

This algorithm takes $O(nlogk)$, because both QUICKSELECT and PARTITION takes $O(n)$ time and the recursion for divide and conquer is:

$$T(n,k) = T(r,k/2) + T(n-r,k/2) + O(n)$$

where n is $len(A)$ and k is $len(R)$. Since it takes $logk$ times to divide R into single $\alpha_i s$, which means the recursion tree has a height of $logk$, and each level takes $O(n)$ time. In total, the recursion takes $O(nlogk)$ time to complete.

Now prove the correctness of the algorithm.

Proof: By induction on k, where k is the number of $\alpha_i s$, i.e., $i \in 1,2,...,k$ for α_i.

- Base case: When $k = 0$, $len(S) = 0$, and $sum = 0$ as the algorithm returns. Thus the base case is correct.
- Induction: Suppose the algorithm is true for $i = 1,2,...,k-1$. Need to show it is correct for $i = k$. Let m = $\lceil len(S)/2 \rceil$. Then there are following cases for i in α_i:
  - $1 \le i < m$: Then α_i is in the left part of S, i.e., S_l, so the n_{th} element in A_l is the n_{th} element in A. Since $len(R_l) < k$, by the inductive hypothesis, the algorithm is correct.
  - $i = m$: Since A_l contains all elements smaller than a_m by how we do the partition, $(total\_sum - SUM(A_l))$ is the correct income for top a_m% people. Thus the algorithm is correct.

- $m < i \leq k$: In this case, since we used the median m as pivot in the QUICKSELECT for R in recursion, we can have the correct rank for $R[i]-R[m]$. Since the n_{th} element in A is the $n-m_{th}$ element in A_r and $len(A_r) < k$, by the inductive hypothesis, the algorithm is correct.

Hence, the algorithm is correct.

■