Tutorial week 9, Nov. 3, 2023

This tutorial is grading-based. You need to implement a variant of quicksort algorithm. It has two changes compared to the quicksort in class; implementing each gives you partial credit.
Total credit: 20 pts

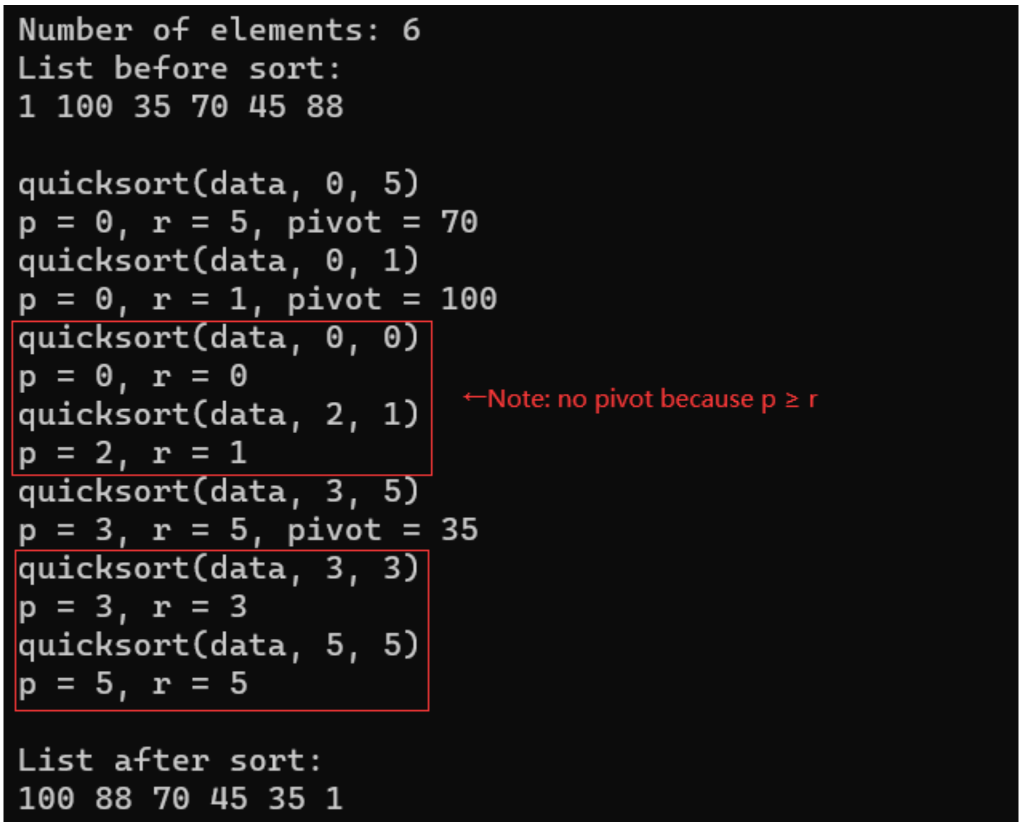
Modification/change 1 (12 pts):
In basic Quick Sort algorithm, the strategy of choosing the pivot affects the running time of the algorithm. One of the strategies is to **select the median among the first, the last, and middle** element as pivot. The middle element is the value at position $\left\lfloor \frac{n}{2} \right\rfloor$ (the floor of n/2), where n is the total number of elements in the input. For example, 3 is selected in the list (1, 7, 3 ,4, 5) and 4 is selected in the list (1, 7, 4 ,5).
In summary, the pivot is the median of the three values for an input array A of n elements: A[0], A[n-1], and A[$\left\lfloor \frac{n}{2} \right\rfloor$].
You can simply switch this pivot with the first element of your array before data partition.

Modification/change 2 (8 pts):
Instead of sorting the input in ascending/increasing order, sort them in decreasing/descending order. Note that you must do this in the data partition part. If you still use the standard quicksort and only reversely print out the array, you won't get credit for this part.

Task:
Implement the Quick Sort algorithm (in **descending order**) with the above pivot choosing strategy.

Specific requirements:
1) Read a file containing an integer list and save the content into an array. This file contains two lines. The first line tells me how many elements in the second line. The second line is the input. This code is provided by us.
2) Implement the Quick Sort algorithm **quick_sort(data, p, r)**.
3) Sort the array in descending order using the above function
4) **Output all the called quicksort functions; print out the values of p ,r and pivot element** whenever the function is called.
5) print out the sorted array.

Example output:



```
Number of elements: 6
List before sort:
1 100 35 70 45 88

quicksort(data, 0, 5)
p = 0, r = 5, pivot = 70
quicksort(data, 0, 1)
p = 0, r = 1, pivot = 100
quicksort(data, 0, 0)
p = 0, r = 0
quicksort(data, 2, 1)
p = 2, r = 1
quicksort(data, 3, 5)
p = 3, r = 5, pivot = 35
quicksort(data, 3, 3)
p = 3, r = 3
quicksort(data, 5, 5)
p = 5, r = 5

List after sort:
100 88 70 45 35 1
```

←Note: no pivot because p ≥ r

Grading: we will test your programs using other test files with the same format as the given ones. Two test cases. Each has half credit for each modification. If the program cannot run correctly, 0 credit for the corresponding part.