Week 4
Quicksort

Preliminary: Lecture on quicksort

Workshop

TASK:

Write a program that utilizes quicksort to sort the list of input numbers

INPUT: a sequence of n numbers. Consecutive numbers are separated by a space

OUTPUT: the list of the n numbers, sorted into *monotonically increasing order*.

Materials:

- SortingTest2.zip : some test cases and the program that generate test cases for evaluating the time complexity
- partition.py : a program that arranges its input into three partitions by values

1) Study how the provided partition function works. Write a program that utilizes it in order to verify your understanding.

2) Test the "*partition*" program with small-sized input. Make the input such that the partitioning works in the following ways:

   - The partition function splits the input into empty list and "all except the minimum".
   - The partition function splits the input into "all except the maximum" and empty list.
   - The partition function splits the input into about two halves.

   What property of input causes the function to produce result in the way that you dictate?

3) Write a "*quicksort*" program that utilizes the provided partition function. Insert the partition function into your program. Do not use import so that the next step can be done properly.

4) Due to the recursion limit of Python language, the code running time will be measured as *the number steps that a line of code repeatedly executes the most*.

   - Add a global variable "counter" and reset its value to 0.
   - Add a line counter += 1 above the line "if A[j] <= x:" in the partition function.
   - Also add the following two lines at the beginning of your program.

```
import sys
sys.setrecursionlimit(10000)
```

Test your program with the provided test cases. Project the increase of the value of "counter" variable with the input size, n. This counter value represents the running time of the program.

5) What do you conclude as the upperbound of the quicksort's running time when the input is already sorted?

$$T(n) = O(\_\_\_\_)$$

6) What do you conclude as the upperbound of the quicksort's running time when the input is already sorted but in reverse order?

$$T(n) = O(\_\_\_\_)$$

7) What do you conclude as the upperbound of the quicksort's running time when the input is in random order?

$$T(n) = O(\_\_\_\_)$$

8) Examine the code of quicksort. What do see as advantage and disadvantage when compared with mergesort?

9) [Advanced] Propose a way to make quicksort most likely avoid its worst case.