CANDIDATE NUMBER:

[TO FILL]

Version: IB/1

EGT3
ENGINEERING TRIPOS PART IIB

Wednesday 1 March 2023 11.25 to 12.15

Module 4M26

ALGORITHMS AND DATA STRUCTURES

Answer one question.

*The **approximate** percentage of marks allocated to each part of a question is indicated in the right margin.*

*Write your candidate number **not** your name on the cover sheet and at the top of each answer sheet.*

The runtime of a successful test case should be no longer than 10 seconds when run on a standard DPO machine.

No access to internet or other resources is permitted.

Solutions should not use and import any python libraries. Only default Python data structures are to be used.

Make sure to verify that you are still getting correct answers after kernel restart (Kernel->Restart & Run All), before uploading your final version of the notebook to Moodle.

STATIONERY REQUIREMENTS
Single-sided script paper

SPECIAL REQUIREMENTS TO BE SUPPLIED FOR THIS EXAM
DPO computers
CUED approved calculator allowed
Engineering Data Book

10 minutes reading time is allowed for this paper at the start of the test.

You may not start to read the questions printed on the subsequent pages of this question paper until instructed to do so.

You may not remove any stationery from the Examination Room.

1.

(a) Write the function, **sort_strings**(S), which takes a list, S, of strings and sorts it using the least significant digit (LSD) radix sort (adapted to string inputs of varying lenghts) in the following order.

1. If length of string, a, is smaller than length of string, b, then string, a, should be appearing in the output before string, b.
2. If strings, a and b, are of the same length, then string, a, should come before string, b, if $ord(a[i]) < ord(b[i])$ where i is the index of the first character for which $a[i] \neq b[i]$. Note, ord, is a python function which converts a character into its ASCII integer value (e.g. `ord('a')=97`).

Also note that a function, **get_character_dict**(S), is provided. This function creates a dictionary, $char\_dict$, which for each character used in S, stores its position in a ranked list of all characters used in S.

[25%]

Examples:

Input: `["abcd", "ab", "abc", "abcde", "a","e"]`
Output: `["a", "e", "ab", "abc", "abcd", "abcde"]`

Input: `["314", "712", "632", "201", "111","11","1","20","5","50","1a"]`
Output: `["1", "5", "11", "1a", "20", "50", "111", "201", "314", "632", "712"]`

Constraints:

- $1 \leq \text{len}(S) \leq 100.$

Code:

In [ ]:
```python
def get_character_dict(S):

    char_set = set([])
    for s in S:
        char_set=char_set | set(list(s))

    count = 0
    char_dict={}
    for c in sorted(list(char_set)):
        char_dict[c]=count
        count+=1
    return char_dict

def sort_strings(S):

    char_dict = get_character_dict(S)

    #Write your code here
```

Tests:

Run example test case 1:

In [ ]:
```python
input_value = ["abcd", "ab", "abc", "abcde", "a","e"]
print (sort_strings(input_value))
```

Run example test case 2:

```
In [ ]:
input_value = ["314", "712", "632", "201", "111","11","1","20","5","50","1a"]
print (sort_strings(input_value))
```

Automatic Evaluation:

Do not forget to run on all test cases when your final implementation is finished!

```
In [ ]:
#DO NOT EDIT THIS CODE!
from evaluation_script import evaluate_solution
evaluate_solution(question_id=3,question_part_id='a',function=sort_strings,
                  test_case_list=[1,2,3,4,5],verbose=True)
```

(b) Assuming that least significant digit (LSD) radix sort is performed on n strings of length, d, with all characters taken from a set of k different characters (e.g. $k = 10$ for decimal digits), as required in Part (a), derive its runtime using Big-Theta notation in terms of n, d, k. You can assume that finding the rank (index of a character in an ordered list of characters) of a character (ie. accessing values in $char\_dict$) takes constant time.

[10%]

Write your answer here.

(c) Explain how you could improve the runtime of the solution in Part (a), if you were allowed to change the way strings and characters are represented? Explain, in what circumstances you could achieve $\Theta(n)$ and $\Theta(\frac{n}{\log n})$ run time? You can ignore the time taken to compute a different representation of the characters and strings in your computation.

[15%]

Write your answer here.

(c) Briefly answer multiple questions below.

(i) What does it mean for a sorting method to be *stable*? Explain which, if any, of Quicksort, Insertion sort, Heapsort, Mergesort methods are stable if implemented simply?

[10%]

Write your answer here.

(ii) Explain what does the *max_heapify* procedure, used in Max-Heap data structure, do. How can this procedure be used to obtain a sorted array from an unordered array?

[10%]

Write your answer here.

(d) Quicksort algorithm works by recursively selecting a pivot value, and partitioning the array so that the elements on the left side are smaller or equal than the pivot value and the elements on the right side of the array are larger than the pivot value. Write the function, **quick_select**(L), which adapts the quicksort algorithm to find the $(k+1)$-st smallest value in the array. Your solution should have $O(n)$ average case runtime for any given k for a list of n elements. Note, you can assume that all possible input orderings are equally likely.

The input to the aforementioned function is a list, L, of two elements, the first of which is a list of numbers and the second of which is value, k.

[20%]

Examples:

Input: `[[1,2,3,4,5],3]`
Output: 4

Input: `[[4,3,2,1,7,11,12,20,18,15],7]`
Output: 15

Constraints:

- $1 \leq n \leq 1000$.
- $-10^6 \leq L[0][i] \leq 10^6$.
- $0 \leq k \leq \text{len}\,(L[0]) - 1$.

Code:

```
In [ ]:
def quick_select(L):

    #Write your code here
```

Tests:

Run example test case 1:

```
In [ ]:
input_value = [[1,2,3,4,5],3]
print(quick_select(input_value))
```

Run example test case 2:

```
In [ ]:
input_value = [[4,3,2,1,7,11,12,20,18,15],7]
print(quick_select(input_value))
```

Automatic Evaluation:

Do not forget to run on all test cases when your final implementation is finished!

```
In [ ]:
#DO NOT EDIT THIS CODE!
from evaluation_script import evaluate_solution
evaluate_solution(question_id=3,question_part_id='d',function=quick_select,
                  test_case_list=[1,2,3,4,5],verbose=True)
```

(e) Explain why your proposed algorithm in Part (d) has average case runtime, $O(n)$. Express its best case runtime using Big-O notation.

[10%]

Write your answer here.

END OF PAPER