Algorithms Homework #1

Due: 107/10/18 (Thu.) 03:00 (Programming)
107/10/17 (Wed.) 09:10 (Hand-Written)

Contact TAs: alg2018ta@gmail.com

Please read the instructions & submission rules very carefully!!
Any loss of credits due to not following the instructions is NOT negotiable.

Instructions

- There are two sections in this homework, including the hand-written part and the programming part.
- **Academic Honesty**

  Cheating is not allowed and is a violation of our school regulations. Plagiarism is a form of cheating. If cheating is discovered, all students involved will receive an F grade for the course (**NOT** negotiable).
- **Collaboration Policy**

  Self-learning by researching on the Internet or discussing with fellow classmates is highly encouraged. However, you must obtain and write the final solution by yourself. Please specify, if any, the references for each of your answers (e.g. the name and student ID of your collaborators and/or the Internet URL you consult with). If you solve the problems by yourself, you must also specify "*no collaborators*".

  You should specify the references and/or collaborators of all problems (including the programming part) on your hand-written answer sheets. **Homework without reference specification will not be graded.**
- **Delay Policy**

  For hand-written problems, you must submit your answer sheets to the instructor at the beginning of the class. TAs will collect them during the first break. **No late submission is allowed for hand-written problems**.

  For programming problems, you have a *six-hour* delay quota for the whole semester. Once you have exceeded your quota, **any late submission will receive no credits**.

Hand-Written Problems

Problem 1

Let all functions be positive. Prove or disprove the following statements using only the definition of asymptotic notations. Also, **determine appropriate bounding integer constants** (that is, n_0 and c) if the statement is correct. No credit will be given if you simply answer *true* or *false* without explanations.

1. (3%) $2n^2 + 7n + 3 \in \Theta(n^2)$.
2. (4%) $f(n) \in O((f(n))^2)$.
3. (5%) $\binom{2n}{n} \in \Omega(2^n)$.
4. (5%) If $f(n) \in \Theta(h(n))$ and $g(n) = f(\lfloor \frac{n}{2} \rfloor)$, then $g(n) \in O(h(n))$.

Problem 2

Solve the following recurrences in $\Theta(g(n))$ notation. If you apply any methods not taught in class or stated in the textbook, you must prove it before using it. Assume $T(1) = 1$.

1. (3%) $T(n) = T(\frac{n}{4}) + T(\frac{n}{5}) + cn$, where $c > 0$ is a constant.
2. (3%) $T(n) = 2T(\frac{n}{4}) + \sqrt[3]{n}$.
3. (3%) $T(n) = 3T(\frac{n}{3}) + n^2 \log_3 n$.
4. (4%) $T(n, n)$, where

$$\begin{cases} T(x, c) \in \Theta(x) & \text{for } c \leq 2, \\ T(c, y) \in \Theta(y) & \text{for } c \leq 2, \text{ and} \\ T(x, y) \in \Theta(x) + T(x, y/2). \end{cases}$$

Problem 3

Consider a bag that contains n balls of different colors. We want to check if more than half of the balls in the bag are of the same color, and if so, finding that color. Suppose comparing the colors between two balls can be done in $O(1)$.

1. (10%) Show how to solve this problem in $\Theta(n \log n)$ time.
   Hint: Split the bag into two halves and solve the subproblems recursively.
2. (10%) Show how to solve this problem in $\Theta(n)$ time.
   Hint: Pair up the balls in the bag to get $n/2$ pairs, then look at each pair. If the two balls are of the same color, keep just one of them; otherwise, discard both of them.

Programming Problems

Problem 1

In this problem, you need to sort the words in an English article by its ASCII code. The input is a text file containing the article, and the output should be a text file of sorted words.

When parsing the input, you should ignore all numbers, spaces, newlines and punctuation marks. However, compound words (e.g. *he's* or *middle-aged* or *NTUEE120th*) should be treated as a single word, and the special characters within such words should also be sorted by their ASCII codes. A pair of sample input/output is provided in Table 1.

input.txt	**output.txt**
She's a 23-year-old graduate from NTU & EE118th, graduated in 2018. "Sheep" are not "goats" (not "lambs" either!) pi = 3.1415926	23-year-old EE118th NTU She's Sheep a are either from goats graduate graduated in lambs not not pi

Table 1: Example output for Problem 1

In the attachment, we include a parser for you so that you don't have to worry about the ambiguity when parsing. You **MUST** use the provided parser which allows us to verify your results automatically. To use the parser, you should import `tools.py` in your codes. Figure 1 shows some example usages for the `input.txt` in Table 1.

For the following subproblems, your `.py` file should take two arguments of type `string`, which are the paths to the input and output text files, respectively. We will test your code in the following manner, and verify the correctness of `output.txt`:

```
$ python p1a_b05901000.py input.txt output.txt
$ python p1b_b05901000.py input.txt output.txt
```

```
>>> from tools import Parser
>>> myParser = Parser("input.txt")
>>> myParser.length()
17
>>> myParser.queryList()
["She's", 'a', '23-year-old', 'graduate', 'from', 'NTU',
'EE118th', 'graduated', 'in', 'Sheep', 'are', 'not',
'goats', 'not', 'lambs', 'either', 'pi']
>>> myParser.query(0)
"She's"
>>> myParser.query(16)
'pi'
```

Figure 1: Usage example for class `Parser`

(a) (10%) Solve the problem using **insertion sort**. Write your code in a file named "`p1a_<StudentID>.py`".

(b) (15%) Solve the problem using **merge sort**. While sorting, you need to split an array into **three** parts instead of two as usual. Assume that an array A is split into sub-arrays a_0, a_1, and a_2. The length of the sub-arrays must satisfy the following constraints:

- $|a_0| \geq |a_1| \geq |a_2|$
- $|a_i - a_j| \leq 1$ for all $0 \leq i, j \leq 2$

You must split an array until the length of its sub-arrays is 1 or 0. Some examples are listed below:

- An array of length 4 must be split into sub-arrays of length [2, 1, 1].
- An array of length 5 must be split into sub-arrays of length [2, 2, 1].
- An array of length 2 must be split into sub-arrays of length [1, 1, 0].

Write your code in a file named "`p1b_<StudentID>.py`".

(c) (6%) In the attachment, we provide 10 example input articles (`input01.txt` to `input10.txt`), each of different length. Record the running time of all 10 articles for the codes in *both* (a) and (b). In "`report_<StudentID>.pdf`", plot two graphs for the result of each algorithm, with x-axis as the size of the input (that is, the number of words in the article), and y-axis as the running time.

(d) (5%) Use the method of least-squares to verify the time complexity from the two plots in (c). Write down the fitting equations you find for each plot in "`report_<StudentID>.pdf`".

Problem 2

In this problem, we wish to design an efficient algorithm that finds any *valley* in an array and returns its **index**. For an array A with n elements, an element with index i is a *valley* if and only if

$$A[i-1] \geq A[i] \leq A[i+1],$$

where $A[-1] = A[n] = \infty$.

Consider an example array A shown in Figure 2 with 8 elements. There are two such valleys in the array, $A[0]$ and $A[4]$, with their values and indices marked in red and blue, respectively.

	$A[0]$	$A[1]$	$A[2]$	$A[3]$	$A[4]$	$A[5]$	$A[6]$	$A[7]$	
∞	2	6	10	12	3	7	9	12	∞

Figure 2: An example array

The input of this problem is a text file where each row of the file represents a single element in the array. The output should be a text file that contains only *one number*, which is a valid valley index. For instance, your code should produce an output text file containing **either** 0 **or** 4 for the array in Figure 2, since our goal is to find *any* valley. A pair of sample input/output is provided in Table 2.

input.txt	**output.txt**
2 6 10 12 3 7 9 12	0

Table 2: Example output for Problem 2

For the following subproblems, your `.py` file should take two arguments of type `string`, which are the paths to the input and output text files, respectively. We will test your code in the following manner, and verify the correctness of `output.txt`:

```
$ python p2a_b05901000.py input.txt output.txt
$ python p2b_b05901000.py input.txt output.txt
```

(a) (4%) Design an algorithm that solves the problem in $O(n)$ time. Write your code in a file named "`p2a_<StudentID>.py`".

(b) (10%) Design an algorithm that solves the problem in $O(\lg n)$ time. Write your code in a file named "`p2b_<StudentID>.py`".

Submission Rules

- Please write your code using **Python 3.x** only. You will lose scores if TAs fail to run your codes due to compatibility issues.
- You may import the following modules in this homework. You are **NOT** allowed to import any other modules unless otherwise listed below:
  - `from tools import Parser`
    (For Problem 1 only, included in the attachment)
  - `import sys`
  - `import time`
- Please pack your source codes into a single `.tar` or `.zip` file named "`hw1_<StudentID>`" (e.g. `hw1_b05901000.zip`) and upload it to CEIBA before the deadline. You should include in it only the following files/folders:

```
hw1_b05901000.zip
├── p1/
│   ├── p1a_b05901000.py
│   ├── p1b_b05901000.py
│   └── tools.py
├── p2/
│   ├── p2a_b05901000.py
│   └── p2b_b05901000.py
└── report_b05901000.pdf
```

- We include a file-checking script (`selfCheck.py`) for you in the attachment. Before uploading your `.zip`/`.tar` file, make sure that it can be verified automatically by the script. Usage is described as follows:

```
$ python selfCheck.py hw1_b05901000.zip
Passed! Your student ID is b05901000.
```