# Algorithms Homework #2

Due: 107/11/06 (Tue.) 03:00 (Programming)
107/11/05 (Mon.) 18:00 (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 put your answer sheets into the paper box in front of BL527. TAs will collect them after the deadline. **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

The randomized version of quicksort is a divide-and-conquer algorithm which works as follows:

($a$) Randomly pick an element in the list as a pivot.

($b$) Partition the list into two parts such that values smaller than the pivot value are placed to its left, while those with larger values are placed to the right.

($c$) Apply this algorithm recursively to the left and the right parts.

1. (5%) Show that its worst-case running time on an array of size $n$ is $\Theta(n^2)$.
2. (10%) Show that its expected running time is $O(n \log n)$, given that it satisfies the following recurrence relation:

$$T(n) \leq O(n) + \frac{1}{n} \sum_{i=1}^{n-1} (T(i) + T(n-i))$$

## Problem 2

There are $n$ houses built along a straight street, each of which containing some goods that worth different values. You are a thief who wants to steal the goods in these houses. Suppose you know in advance the value of the goods in each house, but you cannot steal in two adjacent houses for it may raise suspicion. You want to pick the houses to steal so that you can obtain the maximum total value $V_{max}$.

Consider the example in Figure 1 where there are 8 houses. The value of the goods in each house is stored in an array $v[\cdot]$. In this case, picking the 1st, 4th and 7th houses will result in the maximum stolen value, namely $V_{max} = 8 + 9 + 10 = 27$.

| $v[1]$ | $v[2]$ | $v[3]$ | $v[4]$ | $v[5]$ | $v[6]$ | $v[7]$ | $v[8]$ |
|---|---|---|---|---|---|---|---|
| 8 | 5 | 3 | 9 | 6 | 4 | 10 | 1 |

Figure 1: Example for Problem 2

1. (2%) Find $V_{max}$ in the case where $v = [20, 7, 18, 22, 9, 11, 13, 20, 25, 30]$.
2. (10%) Give a **dynamic programming** algorithm for solving this problem.
3. (3%) Analyze the running time of your algorithm. Write it down in $O(\cdot)$ notation.

## Problem 3

A river divides cities into two parts, one in the north and the other in the south. The order in which the cities are aligned on each side of the river may be different. We wish to build as many bridges across the river to connect the two parts of the same city, but no two bridges should intersect.

Consider the example in Figure 2 where there are 4 cities ($A$, $B$, $C$, $D$), with their northern and southern parts denoted with subscript $n$ and $s$, respectively. In this case, at most 3 non-intersecting bridges can be built ($A_n \leftrightarrow A_s$, $C_n \leftrightarrow C_s$, $B_n \leftrightarrow B_s$).

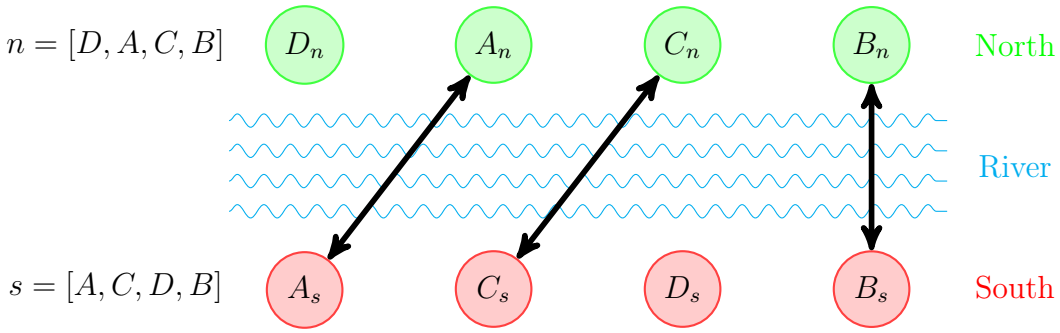


Figure 2: Example for Problem 3. Double arrows represent bridges across the river.

1. (2%) Find the maximum number of non-intersecting bridges in the case where $n = [C, B, A, F, D, E]$ and $s = [B, C, A, D, F, E]$.

2. (15%) Give a **dynamic programming** algorithm for solving this problem.
   *Hint*: Try relating this scenario to the *longest increasing subsequence* problem.

3. (3%) Analyze the running time of your algorithm. Write it down in $O(\cdot)$ notation.

# Programming Problems

## Problem 1

In this problem, you will be given a positive integer $p$. Let $N_p$ be the set of positive integers whose digits multiply to $p$ (e.g., $2 \times 6 \times 8 = 96$), with the constraint of $n > 10, \forall n \in N_p$. Note that for a given $p$, the set $N_p$ is either empty or infinite. We want to find the minimum number $n_{min}$ in the set $N_p$. Below are some examples:

($a$) $p = 96$
$N_p = \{268, 286, 348, 384, 438, 446, 464, ..., 1268, 1286, 1348, ...\}$
$n_{min} = 268$

($b$) $p = 1$
$N_p = \{11, 111, 1111, 11111, ...\}$
$n_{min} = 11$

($c$) $p = 23$
$N_p = \varnothing$

**Input**
The input of this problem is a text file of numbers, where each row contains a value of $p$. Note that we will not provide any $p$ that result in an empty $N_p$, that is, you don't need to handle case ($c$).

**Output**
The output should also be a text file of numbers where each row is the value of $n_{min}$ of its corresponding $p$.

A pair of sample input/output is provided in Table 1.

| input.txt | output.txt |
|---|---|
| 96 | 268 |
| 1 | 11 |
| 20 | 45 |

Table 1: Example output for Problem 1

For the following subproblem, 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 p1_b05901000.py input.txt output.txt
```

1. (15%) Give a **greedy algorithm** for solving this problem. Write your code in a file named "`p1_<StudentID>.py`".
2. (5%) Briefly explain your work in "`report_<StudentID>.pdf`".

## Problem 2

Frank is a student in NTU, and October has always been a particularly busy month for him. Piles of homework assignments from the many classes he takes are all waiting to eat him alive. For each homework assignment $h_i$, there is a deadline $d_i$ and a penalty $p_i$ for not being able to submit in time.

Frank is a smart student so that ideally, he can finish most of the assignments in time. Unfortunately, Frank has a girlfriend who expects him to spend at least 4 hours with her every day, and the penalty for not obeying his girlfriend is $p_{girl} = \infty$. The time left is only enough for Frank to finish *one* assignment *per day*. That being said, Frank still wants to avoid the penalties as much as possible by deciding which assignments to do first, so that the total penalty $P_t$ he receives can be minimized.

| | $h_1$ | $h_2$ | $h_3$ | $h_4$ | $h_5$ | $h_6$ | $h_7$ |
|---|---|---|---|---|---|---|---|
| Deadline $d_i$ | 1 | 2 | 3 | 4 | 4 | 4 | 6 |
| Penalty $p_i$ | 25 | 65 | 35 | 50 | 15 | 90 | 5 |

Table 2: Example homework set for Problem 2

| Day# | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| Do Assignment# | $h_3$ | $h_2$ | $h_4$ | $h_6$ | $h_5$ | $h_7$ | $h_1$ |
| Received Penalty | 0 | 0 | 0 | 0 | 15 | 0 | 25 |

Table 3: An optimal schedule for the homework set in Table 2

Table 2 contains an example set of homework assignments $h_1 \sim h_7$, where an optimal schedule is provided in Table 3. Assignments marked in green are submitted in time so that there is no received penalty. Assignments marked in red are delayed submissions so that the received penalty is the value of its corresponding $p_i$. The minimized total penalty in this case is $P_t = 15 + 25 = 40$.

**Input**

The input of this problem is a text file containing three rows of numbers, with each number separated by a space. The first row stores the indices, the second row stores the deadlines, and the third row stores the penalties, of the assignments.
Note that if the input contains $n$ homework assignments, then the following constraint exists:

$$1 \leq d_i \leq n, \forall i$$

**Output**

The output should be a text file containing two rows of numbers. The first row should describe your optimal schedule for the input homework set, where the numbers indicate the order of indices in which Frank should do his homework. The second row should contain only one number, which is the minimized value of $P_t$ calculated from your schedule. Note that for a given homework set, the optimal solution may not be unique.

| input.txt | output.txt |
|---|---|
| 1 2 3 4 5 6 7<br>1 2 3 4 4 4 6<br>25 65 35 50 15 90 5 | 3 2 4 6 5 7 1<br>40 |

Table 4: Example output for Problem 2

A pair of sample input/output is provided in Table 4.

For the following subproblem, 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 p2_b05901000.py input.txt output.txt
```

1. (20%) Give a **greedy algorithm** for solving this problem. Write your code in a file named "`p2_<StudentID>.py`".
   *Hint*: Last-minute homework is not encouraged, but it sure is *greedy*.

2. (10%) Briefly explain your work in "`report_<StudentID>.pdf`".

## 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:
  - `import sys`
- Please pack your source codes into a single `.tar` or `.zip` file named "`hw2_<StudentID>`" (e.g. `hw2_b05901000.zip`) and upload it to CEIBA before the deadline. You should include in it only the following files/folders:

```
hw2_b05901000.zip
├── p1/
│   └── p1_b05901000.py
├── p2/
│   └── p2_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 hw2_b05901000.zip
Passed! Your student ID is b05901000.
```