CompSci 220 (2020 S2) Assignment 2 Due: 2020-09-04, 9pm

There are **five** problems listed below. To get full credit for this assignment you need to complete all of them!

If you are stuck or confused by any of the problems, feel free to post on Piazza!

Show all working; if you do not show your work the mark will be reduced. You should submit via Canvas a **SINGLE PDF** file containing the answers to the written questions. A scanned handwritten submission is acceptable if and only if it is very neatly written. If typing the assignment, do the best you can with mathematical symbols. For exponents, write something like

```
2^n
```

if using plain text. Use LaTeX if you really want it to look good.

Answers to programming questions must be submitted via the automated marker system:

`automarker.cs.auckland.ac.nz`

Question 5 will be unlocked on Monday August 24th

Best of luck, and enjoy the problems!

1. (4 marks) Solve the following recurrence relation:

$$\begin{cases} T(n) = \frac{1}{n}(T(0) + T(1) + \ldots + T(n-1)) + 5n, \\ T(0) = 0. \end{cases}$$

Show all working. You can leave the n-th harmonic function as H_n in your answer.

2. (3 marks) Solve the following recurrence relation if n is a power of 7:

$$\begin{cases} T(n) = T(\left\lfloor \frac{n}{7} \right\rfloor) + \log_3(n), \\ T(1) = 0. \end{cases}$$

Show all working.

3. (3 marks) Find the worst running time of the following piece of code:

function strange (list $a[0..n-1]$ of integers such that $abs(a[i]) \leq n$ for every $0 \leq i \leq n-1$, list $b[0..2n]$ of zeroes)
for $i \leftarrow 0$ to $n-1$ **do**
    $a[i] \leftarrow a[i] + n$
for $i \leftarrow 0$ to $n-1$ **do**
    for $j \leftarrow 0$ to $abs(a[i]-1)$ **do**
        $b[j] \leftarrow b[j] + 1$
return b

Note, that $abs(x)$ returns the absolute value of x. Count only + as an elementary operation and ignore all others. Explain your answer.

4. **(3 marks)** Find the recurrence relation and the initial conditions of the average running time of the following piece of code:

```
function strange-sum(list a[0..n − 1] of integer numbers)
sum ← a[n − 1]
if n > 2 then
        middle← ⌊(n − 1)/2⌋
        b[0..middle] is a list of zeroes.
        for  i ← 0 to middle do
                if a[i]%2 = 0 then
                        b[i] ← a[i] + a[n − 1 − i]
        sum← strange-sum(b)
return sum
```

Count only $+, \%$ and comparisons as elementary operations and ignore all others including $-$. Explain your answer.

5. Top songs **(7 marks)**

Devise an algorithm and implement it in a program to solve the following problem, similar to one often faced by an MP3 player. For our purposes, a song consists of the following data fields: title (a nonempty ASCII string), composer (a (possibly empty) ASCII string), running time (a positive integer). Input consists of n songs and an integer k with $1 \leq k \leq n$. Your program must find the k songs with longest running times, and output these songs in descending order of length of song. If songs have the same running time, then we use lexicographic order on the title, and then on the composer, to get a total ordering. The lexicographic order comes from the usual order on ASCII characters.

- You should first make sure that your algorithm and implementation are correct! It is up to you to come up with test cases and the corresponding correct output. Usually, if a program contains an error, this can be detected on a small test case. You are allowed to share test cases with the class via the class forum.

  Roughly 60% of marks will be awarded for correctness. About half of this will be for performance on the automarker instances, and half will be awarded by markers inspecting your code and reading the comments. *To obtain full marks, you must give clear comprehensive comments explaining the algorithm you use and implementation details.* As a rough guide, at least 20 lines of comments will be needed to do this properly, and perhaps more.
- Next you should consider efficiency. Remember that the test cases may use any value of k and n, so test it thoroughly and think about hard cases for your algorithm.

  Roughly 40% of marks will be awarded for speed on large inputs.
- You may use any standard library functions in your programming language. Please see below for information on input and output formats. You must submit via the automarker `automarker.cs.auckland.ac.nz`.

Questions involving programming

- Python, Java, C++ and some other languages are supported.
- Your answer to each question should be a single file (containing all nonstandard classes you use). You may assume that the markers have access to all standard libraries.
- A sample input and output file for each question will be available from Canvas. The markers may check the output with a text comparison program, so it must be in EXACTLY the right format.
- The automated feedback and submission system ("automarker") is available. You must submit your answer via this system. You may take account of the feedback given by the automarker, and resubmit before the deadline without penalty.

- Your program(s) may be tested on randomly generated input files, some of which may be very large. Some of these test files may not be those used as feedback test cases on the automarker. Marks will be allocated for correctness and speed of the programs (usually about 60% for correctness, 40% for speed). Simply "passing" the test cases on the automarker may not guarantee maximum marks, but it will guarantee more than half marks for correctness. If full marks for correctness are not obtained, then the marks for speed are automatically set to zero.
- You must at least include comments with the name of the author, UPI, and the purpose of the code. Human markers may inspect the code and in some cases will deduct marks for very messy code.

Input and output format

For this assignment, the following format will be used. Sample input and output is available on Canvas. Pay attention to line breaks and beware of nonstandard software such as anything made by Microsoft. For best results, use a Linux/Unix environment (the automarker does).

You can assume that input will come from standard input in a stream that represents one string per line. Output should be sent to standard output. In the automarker, the input will come from a file that is piped to standard in and the output redirected to a file, but your program shouldn't know that.

The first line gives the integer k, and the second line a separator character that is not contained in any of the song or composer strings. The subsequent lines give the song data, with fields separated by the separator character.