CS 3353 Spring 2023

Program 3 – Preliminary job screening

Due: 4/21 (Fri) 11;59pm

You are the head of the HR department of your company. Your company wants to hire a new CTO (Chief Talking Officer), and you are tasked with screening off initial applicants. You have decided to using two criteria to screen the applicants. For each applicant, you measure two quantities:

- Words per minute (WPM): How many words can the applicant speak per minute and remain clear. You would like that value to be as high as possible.
- Improper words per minute (IPM): There are certain words/phrases that are deemed improper by your company. You want to detect how likely the applicant would mention those terms. You want that value to be as low as possible.

We consider an applicant x to be an eligible candidate if there is no other applicants that is ***strictly*** better than applicant x in both WPM and IPM. Your task is that, given a pool of applicants, select all the eligible candidates.

Algorithms

The simplest way of solving this problem is a naïve double for loop.

```
S = ϕ
For each applicant x
    Eligible = true
    Fir each applicant y that is not x
        If (y is better than x)
            Eligible = false
            Break
    If Eligible)
        Add x to the set of solution S
Return S
```

This algorithm will take $O(n^2)$ time for a set of n applicants. However, there is a recursive solution to this problem that you can implement and can run in O(n log n) time. Your task for this program is to implement that algorithm.

Details

You will be given a program (hw3test.cpp) that will contain a driver program that will read in a set of candidate (in terms of WPM and IPM). Your task is to implement the following function (please do NOT implement it as a class method):

```
vector<int> BestApplicant(const vector< pair<float, float> >& applicants);
```

where applicants is a vector of pairs of float. The first value is the WPM and the second value is the IPM.

Your function should return a vector of integers containing the list of eligible candidates (each integer denotes the location of the applicant in the applicants' vector that is passed in. Notice that there cannot be duplicate value in the output vector.

Your function should have the following structure:

```
vector<int> BestApplicant(const vector< pair<float, float> >& applicants)
{
    // any pre-processing code that is necessary
    S1 = recursiveBestApplicant(.....)
     // any post-processing that is necessary
   Return ….
}
```

You should write a second function, recursiveBestApplicant() that is the actual recursive algorithm.

The recursive algorithm

The basic idea of the recursive algorithm is like this

- Break your set of applicants into two groups
- Recursively find the best applicants of each group (as if that group is the only applicants)
- Combine the results of the two sets to form the final solution.

For example, if you have six applicants (app[0] ... app[5]), what you may do is the following

- Divide the applicants into two groups (app[0], app[1], app[2]), (app[3], app[4], app[5]))
- In the group of (app[0], app[1], app[2]) find all applicants that are eligible (as if those 3 are the only applicants)
- Similar for the group (app[3], app[4], app[5])
- Look at the eligible candidate returns, and from them select the applicants that are eligible candidates (as if all 6 of them are applicants).

(Hint: you can do better than just applying the naïve algorithm for the combine step)

What to hand in

You need to submit a single file, "hw3.cpp" that contain the code of your functions. You can define other functions and class to help you out. I do require your function to adhere to the C++-17 standard, and not the C++-20 standard.

Once I received the file, I will compile it with the driver program for testing purpose. ***You MUST ensure that I can have a project that has "hw3.cpp" and "hw3test.cpp" as my only 2 files, and can compile and run the program WITHOUT ME HAVING TO CHANGE ANYTHING IN ANY FILE. Otherwise you will automatically be disqualified from the bonus round.***

(For example, you should NOT have a main() function in your hw3.cpp file.)

Scoring and bonus

The base score for this program is 95 (if you implement it correctly, using the recursive algorithm above). If you do NOT do recursion at all, the best you can get is 15.

All programs that passes the test will be timed and the most efficient programs will get bonus as in program 2. Notice that ONLY program that is handed in without using the late pass is eligible for bonus.