CPSC 413 June 2, 2020

University of Calgary

Homework Assignment # 2

Due: 9. June 2020, **11:59pm**

LATE ASSIGNMENTS WILL NOT BE ACCEPTED

Please fill out (or reproduce the information in) the collaboration page at the end of this assignment and include it in your pdf submission.

Collaboration Rules:

In order to solve the questions, you are allowed to collaborate with students from the same course, but not with anybody else. If you do collaborate with other students, then you have to list them in the appropriate fields on the collaboration page. You have to write up the solutions *on your own* in *your own words*. You can meet with your collaborators in order to generate ideas, but you are not allowed to write up the solutions during these meetings or to take any notes, pictures, etc. away from those meetings. Public/private forums, chats, messaging portals, D2L, Zoom, Discord, Piazza, etc. are all possible collaboration spaces included in this limitation. Any solutions posted in one of these spaces would be a violation of the collaboration guidelines by the poster and any student that plagiarizes the solution.

You are allowed to use literature, as long as you cite that literature in a scientific way (including page numbers, URLs, etc.) to form parts of your solution. You cannot however use literature to just copy a complete solution. In any case, your solutions must be self-contained. For example, if you use a theorem or lemma that was not covered in the lecture and that is not "well-known", then you have to provide a proof of that lemma or theorem.

If you are in doubt whether a certain form of aid is allowed, ask your instructor!

Academic misconduct (cheating, plagiarism, or any other form) is a very serious offense that will be dealt with rigorously in all cases. A single offense may lead to disciplinary probation or suspension or expulsion. The Faculty of Science follows a zero tolerance policy regarding dishonesty. Please read the sections of the University Calendar under the heading "Student Misconduct".

Your submission **must**:

- Consist of a single, clearly legible file uploadable to GradeScope with clearly indicated solutions to the problems. (PDFs produced via LaTeX, Word, Google Docs, or other editing software work well. Scanned documents will likely work well. **High-quality** photographs as pdf are OK if we agree they're legible.)
- Include prominent numbering that corresponds to the numbering used in this assignment handout. Put these **in order** starting each problem on a new page, ideally. If not, **very clearly** and prominently indicate which problem is answered where!
- Include at the start of the document the statement: **"I have read and followed the guidelines for academic conduct in CPSC 413 Spring 2020. As part of those rules, when collaborating with anyone, (1) I took no record but names away, and (2) after a suitable break, I created the assignment I am submitting without help from anyone other than the course staff."**

1 Safe by Committee [6 marks]

In the Safety Committee Problem we have a bunch of shifts $SHIFTS$. Each shift $S \in SHIFTS$ is worked by a unique employee and has a start time $s(S)$ [also known as a left endpoint] and an end time $f(S)$ [also known as a right endpoint] ($s(S) < f(S)$) which often overlaps with other employees working at the same time. We want to designate a subset $SHIFTS_{comm} \subseteq SHIFTS$ of those shifts to be part of a safety committee. The goal is to minimize the honorarium we pay to every employee in the committee. So we want to select a committee that is as small as possible yet still covers our butt while ensuring that at all times one of the committee shifts overlaps every shift of an employee not on the committee.

Below is pseudo-code for a correct greedy algorithm that solves the Safety Committee Problem. The main idea of the pseudo-code is to work its way through the day from earlier to later finding one committee member at a time. This committee member will be S_{max}. Once a committee member is identified all shifts covered by it are marked covered. The goal to determine a committee member is to find a shift which ends as late as possible while still adding safety coverage to the earliest shifts not yet covered. Once we reach the limit in our exploration we choose this shift to be on the committee and mark everything it covers as covered and continue on the next one (Note, if while looping on the next one we run into shifts that will be covered by the last committee member completely we automatically note them as covered and skip consideration of them as a committee member).

```
define produce_safety_committee(array_of_shifts):
    S_max = none            # where s(none) = f(none) = -∞
    previous_S_max = none

    list_current = empty list
    all_endpoints = all shift (left/right) endpoints, sorted in increasing order

    for endpoint p in all_endpoints:
        let S be the shift to which p belongs
        if p is a left endpoint:
            if f(S) > f(S_max)
                set S_max to S
            #
            # Shifts starting before f(previous_S_max) are already covered.
            # so we don't need to worry about covering them.
            #
            if p < f(previous_S_max):
                mark S as covered
            else:
                add S to list_current

        else if p is a right end point and not marked as covered:
            previous_S_max = S_max
            add S_max to the committee
            mark all shifts from list_current as covered
            list_current = empty list
```

In this question you will complete a proof of correctness of this algorithm.

1.1 Part 1 [2 marks of 6]

Let $S_1, \ldots, S_k$ be the set of shifts returned by our algorithm. Assume without loss of generality that $S_1, \ldots, S_k$ is sorted by finishing times. We will denote the starting and finishing times of shift S_j by $s(S_j)$ and $f(S_j)$ respectively. We first prove that $S_1, \ldots, S_k$ is a valid safety committee, by proving that:

Fact 1 *If S is a shift such that $s(S) \leq f(S_j)$, then S is overlapped by one of $S_1, \ldots, S_j$.*

Proof: The proof is by induction on j.

Base Case: For $j = 1$, observe that ... **FILL IN THIS PART**.

Inductive Hypothesis: Suppose now that every shift S whose starting point is $s(S) \leq f(S_j)$ is overlapped by $S_1, \ldots, S_j$.

Consider a shift S such that $s(S) \leq f(S_{j+1})$.

- If $s(S) \leq f(S_j)$ then ...
  FILL IN THIS PART.
- Otherwise, S is a shift $f(S_j) < s(S) \leq f(S_{j+1})$ such that $S_1, \ldots, S_j$ does not overlap it. We need to show that S_{j+1} overlaps S to complete our proof. ...
  FILL IN THIS PART.

We have shown though induction that if S is a shift such that $s(S) \leq f(S_j)$, then S overlaps one of $S_1, \ldots, S_j$.

End Proof

1.2 Part 2 [2 marks of 6]

Let $T_1, \ldots, T_m$ be the shifts in a smallest safety committee, sorted by increasing finishing time. Our algorithm returns a complete safety committee $k \geq m$, as we could have the smallest solution or it is bigger.

Next we will establish that

Fact 2 *For $j = 1, \ldots, m$, $f(T_j) \leq f(S_j)$.*

Proof: The proof is by induction on j.

Base Case: This is clear for $j = 1$, because ...

FILL IN THIS PART

Inductive Hypothesis: Suppose now that the statement is true for S_j and T_j.

Consider S_{j+1} and T_{j+1}. Let S be the shift with the earliest finishing time amongst those that start after $f(S_j)$.

... **FILL IN THIS PART**

Hence $f(T_{j+1}) \leq f(S_{j+1})$.

End Proof

1.3 Part 3 [2 marks of 6]

In order to prove that $k \leq m$ we only need to prove that every shift overlaps an element of $S_1, \ldots, S_m$. Proof by contradiction: Indeed, if there were a shift S that does not overlap any of them, then ...

FILL IN THIS PART

This is clearly impossible, since in that case none of $T_1, \ldots, T_m$ would overlap with S. By contradiction then there is no shift $s(S) > f(S_m)$ that isn't overlapped which means $k \leq m$. Combined with earlier we can now say $k == m$, AKA our algorithm finds a minimal size committee.

2 Recurrently [6 marks]

2.1 Sort Of [3 marks of 6]

Consider the following sorting algorithm:

```
define meh_sort(A, first, last):
  if A[first] < A[last]:
      exchange A[first] and A[last]

  if (first + 1 < last):
    mid = (last - first + 1) // 3     # integer division
    meh_sort(A, first + mid, last)  # sort last two-thirds
    meh_sort(A, first, last - mid)  # sort first two-thirds
    meh_sort(A, first + mid, last)  # sort last two-thirds again
```

a. Write a recurrence relation that describes the worst-case running time of function `meh_sort` in terms of n, where $n = \texttt{last} - \texttt{first} + 1$. You can ignore floors and ceilings.

2.2 Not At All [3 marks of 6]

Consider now this even less useful algorithm:

```
define not_useful(A, first, last):
  if last < first + 4:
      x = A[last] - A[first]
  else:
    x = not_useful(A, first+1, last-1) - not_useful(A, first+2, last-2)
  return x
```

Write a recurrence relation that describes the worst-case running time of function `not_useful` in terms of n, where $n = \texttt{last} - \texttt{first} + 1$.

3 Test Test Testing [6 marks]

Consider a sequence $A[1..n]$ of integers. An *equivalence test* is an operation that takes as input two indices $i, j \in \{1, \ldots, n\}$ and returns "EQUAL" if $A[i] = A[j]$, and "NOT EQUAL", otherwise.

A *majority element* in A is an index i such that the value of $A[i]$ appears more than $n/2$ times in A. Using only equivalence tests, we want to find out, whether there is a majority element $A[i]$, and if yes, determine its index i. Using only equivalence tests means you cannot sort the input as you can't do a comparison or order, just equivalence. You can access the size of input of course.

Design a Divide and Conquer algorithm that solves this problem. Specifically, for any array $A[]$ of n integers, if there is an integer a and a set $I \subseteq \{1, \ldots, n\}$, $|I| > n/2$, s.t. $A[i] = a$ for all $i \in I$, then your algorithm should output an arbitrary index $i \in I$. If there is no such set I, the algorithm should output ∞.

Your algorithm can access the input only through equivalence tests, and it must use $o(n^2)$ of them. For full marks, your algorithm should need at most $O(n \log n)$ equivalence tests.

Describe your Divide and Conquer algorithm, argue why it is correct and analyze its running time. For your analysis, you are allowed to use without proof the result of the "general recurrence" from class (Section 5.3). Describe in detail which data structures you are using to achieve the claimed running time. Provide pseudo-code for your algorithm in addition to a high level plain text explanation.

Hint: Consider the following, simpler *majority verification problem*. The input is an array $A[]$ and an index i, and the output is "yes", if $A[i]$ is a majority element, and "no" if it is not. Design a simple algorithm that solves the majority verification problem using $O(n)$ equivalence tests. Use this algorithm as a sub-routine in your Divide and Conquer algorithm for the majority element problem, to determine which of the solutions for the sub-problems is also a solution for the original problem.

4 Got Stones [6 marks]

Recently, we've researched a new currency, based on grouping various numbers of stones together. Each number k of stones is assigned a value $val(k)$, for some non-decreasing function val. The researchers realized, that by splitting a fixed number of stones into smaller groups, one can obtain different sums of values for all groups combined. For example, consider the value-function in the table below. If we split 8 stones into three groups of 1, 3, and 4 stones, respectively, then the total value is $val(1) + val(3) + val(4) = 1 + 8 + 9 = 18$. On the other hand, splitting the 8 stones into two groups of 2 and 6 stones, respectively, yields a total value of $val(2) + val(6) = 5 + 17 = 22$.

number of stones	1	2	3	4	5	6	7	8
value	1	5	8	9	10	17	17	20

The researchers are interested in finding an efficient method of grouping a given number of stones, so that the total value of all groups combined is as large as possible. Can you help them?

The input for the Stone Grouping Problem is a positive integer n and an array $val[]$ of length n that assigns each number i of stones a value $val[i]$ such that $val[i+1] \geq val[i]$, for $i \in \{1, \ldots, n-1\}$. A solution for this problem is a sequence $s_1, \ldots, s_k$ of positive integers, such that $s_1 + \cdots + s_k = n$. The value of this solution is $val[s_1] + \cdots + val[s_k]$. An optimal solution is one of maximal value.

4.1 Riddles me Stones [1 mark of 6]

For the input below, give an optimal solution and its value.

$n = 7;$

i	1	2	3	4	5	6	7
$val[i]$	3	6	8	9	10	17	18

4.2 The Bell Stone [1 mark of 6]

Give a Bellman Equation that describes the value of the optimal solution for any input to the Stone Grouping Problem. Briefly explain why the Bellman Equation is correct.

4.3 42 Stones [2 marks of 6]

Give a Dynamic Programming algorithm in pseudo-code that computes the *value* of an optimal solution to the Stone Grouping Problem. Your algorithm must be based on the Bellman Equation you designed in Part (b), and have polynomial running time. For full marks its running time should be $O(n^2)$. State the running time of your algorithm and briefly explain why your statement is correct.

4.4 Stone Tablets [2 marks of 6]

Give an algorithm in pseudo-code that computes the optimal *solution* to the Stone Grouping Problem. Your algorithm can use as a sub-routine your algorithm from Part (c). It must have polynomial running time, and for full marks its running time should be $O(n^2)$. State the running time of your algorithm and briefly explain why your statement is correct.

Grading

The assignment will graded by calculating a total marks earned. This total marks will be divided by the total marks available. The highest letter grade the resulting percentage exceed or matches will be assigned as the letter grade for the assignment. The conversion of percentage value ranges to letter grades is as follows.

	A+	A	A-	B+	B	B-	C+	C	C-	D+	D
Minimum % Required	95%	90%	85%	80%	75%	70%	65%	60%	55%	50%	45 %

Cover Page for CPSC 413
Homework Assignment # 2

Name: _____

Course Section: _____

Collaborators:

Question 1: _____

Question 2: _____

Question 3: _____

Question 4: _____

Other Sources:

Question 1: _____

Question 2: _____

Question 3: _____

Question 4: _____

Declaration:

I have written this assignment myself. I have not copied or used the notes of any other student.

Date/Signature: _____