CSC263 Winter 2021 Problem Set 1

Samar Sabie & Michelle Craig

Due: February 11, 2021 11 am ET on MarkUs

Instructions

- Your problem sets are graded on both correctness and clarity of communication. Solutions that are technically correct but poorly written will not receive full marks. Please read over your solutions carefully before submitting them.
- Each problem set may be completed as an individual or in a group of two students. Partners do not need to be registered in the same section of the course.
- Solutions must be typeset electronically, and submitted to MarkUs as a PDF with the correct filename. This filename is **ps1.pdf** for Problem Set 1. **Handwritten submissions will receive a grade of zero.**
- Your submitted file should not be larger than 9MB. Your file might exceed this limit if you use a word processor like Microsoft Word to create a PDF; if it does, you should look into PDF compression tools to make your PDF smaller, although please make sure that your PDF is still legible before submitting!
- The work you submit must be that of your group; you may not use or copy from the work of other groups, or from external sources like websites or textbooks. We recognize that although it should not be necessary to do so, students may wish to read a different textbook or reference material about course topics. This is not prohibited, but you must declare it. (See the next bullet). All the information you need has been covered in lectures and/or in our textbook.
- On the front page of your submission, you must list **every** source of information you used to complete this homework (other than your lecture notes, our textbook, or conversations from office hours or lectures). For example, indicate clearly the **name** of every student from another group with whom you had any discussions, the **title and sections** of every textbook you consulted (excluding the course textbook), the **source** of every web document you used or website you visited (other than our own CSC263 site.)

Question 1 [5 marks]

In lecture, we analyzed the average-case runtime for the algorithm `hasTwentyOne` under the assumption that each element in the list was equally likely to have the value 21. Redo the analysis changing the assumption about the inputs. Assume now that there is at most one 21 in the list and that it is equally likely to be in any of the n positions or not in the list at all. Show your work justifying each step and the values that you are using in your analysis.

Question 2 [10 marks]

In this course for this term, problem set partners do **not** have to come from the same course section and students are allowed to work as individuals. In some other terms, groups were only legal if the student was working solo or if the two partners were registered in the same section. The MarkUs system produces a file that has one line per submission and contains the login of each partner. If the submission is from an individual, that student's login is listed in both places (so partner1 is partner1 and partner2.) [1] The function below determines if all groups on a MarkUs submission list are legal according to the stricter definition from previous terms.

```
def all_groups_legal(submissions):
    for submission in submissions:
        course_section_p1 = student_to_section[submission.partner1]
        course_section_p2 = student_to_section[submission.partner2]
        if not course_section_p1 == course_section_p2:
            return False
    return True
```

Perform a best-case, worst-case and average-case analysis of this code. State explicitly the operation or operations that you are counting in your analysis and give exact counts (not simply an asymptotic expression.)

For the average-case analysis, you will discover that you need to determine the probability at each point in the list (p), that a group contains an across-section partnership (i.e. the partners come from two different sections.) For the sake of this analysis, you may make the simplification that *this probability is independent of the position in the list and independent of the other values in the list.* [2]

Since you don't know the best value to use for p, do as much of your analysis as you can with the variable. Then, make a good argument for two different values of p to investigate and carry out both investigations using actual numbers. One of your scenarios should assume that students are no more likely to pick someone from their own section than to pick a random student from the course.

There are three sections of the course and the enrolment is respectively 240, 240 and 220 students. We know from past course data that the probability that a student works without a partner is 0.20 and that the distribution of students working alone matches the distribution of students across the various course sections.

Question 3 [15 marks]

Two arrays A and B contain a and b elements respectively. The values in these arrays are in decreasing order. Together, the arrays contain n unique elements (i.e. no duplicates) where $n = a + b$.

(a) Design an algorithm for merging the two arrays into one sorted array X with a worst-case runtime of $\mathcal{O}(n)$ element comparisons. Describe the general approach of your algorithm in a few sentences. The sentences can be fairly broad, just giving an overall description of your approach, since you will provide pseudocode in the next subquestion.

(b) Provide complete pseudocode for your algorithm.

(c) Prove that your algorithm meets the constraint on the worst-case running time.

[1] It is possible with various improvements to Markus that this file description may no longer be strictly true, but let's assume that MarkUs still works this way.

[2] This simplification is clearly untrue. To see this, imagine a class of four students in two sections of two students each with no individual groups allowed. If the first group is valid, then the second group must be valid.

(d) Assuming that a and b are large (but unknown) values, express the best-case running time of your algorithm in terms of a and b. Describe a family of input cases that can be merged in this time and justify why no other input could require fewer comparisons.

(e) Next, we want to calculate an average-case complexity of your algorithm. In order to do this, we need to make the following assumption about the distribution of possible inputs. Let X be the resulting sorted array. Assume that every combination of A and B that could have led to creating X is equally likely. For this subquestion, we are no longer assuming that neither array is empty. So one valid input (of many) is that A is empty and all n elements are originally in B.

(f) Now suppose that we have k sorted arrays all in decreasing order with a total number of n unique elements. We would like to merge these arrays into one sorted array. Devise an algorithm with worst-case runtime $\mathcal{O}(n \log k)$, using a data structure that we learned in class. Your data structure can hold **at most** k elements. Give a detailed description of your algorithm (write pseudocode if necessary), and argue that your algorithm works correctly and has the desired worst-case runtime.

Question 4 [10 marks]

In class, we implemented heaps as arrays, where each position in the array corresponded to a specific location in the binary tree. This gave us the following convenient property: a node corresponding to index i has its left child stored at index $2i$ and its right child at index $2i+1$. And for any node at index i other than the root, its parent is stored at index $\lfloor i/2 \rfloor$. We stored the size of the heap separately from the array and so finding the array element into which we would insert a new heap element was a constant-time operation.

(a) Suppose now that we implement a heap using a nearly-complete binary tree T as discussed in lecture, but we **do not have a corresponding array**. Given a reference to the current last node v, describe an efficient algorithm for finding the insertion point for node w the new last node, using only the following methods of binary tree interface:

- `T.left` Return a pointer to the left child of node T (or null if no left child).
- `T.right` Return a pointer to the right child of node T (or null if no left child).
- `T.parent` Return a pointer to the parent of node T (or null if T is the root).

Be sure to specify your algorithm clearly. Using pseudocode in the style of CLRS is one way. One other alternative is to write Python-like pseudocode or your own pseudocode. The important thing is that the algorithm does not use methods other than those provided by the interface above and that you handle all possible corner cases.

(b) Assume that your heap contains k items. Analyze the best-case running time of your algorithm. Hint: Remember that you need to provide a running time, provide an input on which the algorithm takes this much time, and provide an argument that no other input can possibly do better.

(c) Perform a worst-case analysis of the running time of your algorithm. Reminder: look at the hint from part (b).

Question 5 [10 marks]

We define a *Delete-Insert-Stable* (DIS) heap as a heap with no duplicate elements where the result of extracting the root element and immediately re-inserting that element is the original heap.

1. Provide an example of a DIS max heap with seven elements. Display both the original heap (which is of course also the final heap) and the intermediate heap (after the call to ExtractMax) as trees.
2. Provide an example of a max heap with seven distinct elements that is not a DIS heap. Display the original heap, the intermediate heap and the final heap all as trees.
3. Formally describe the relationship between the elements that must hold for a heap to be a DIS max heap.
4. Prove that your description in part (c) describes all DIS max heaps and does not hold for max heaps that are not DIS.