- **Mark your attendance** at the "Quiz 1 Attendance" module on Moodle.
- This exam has 6 questions for a total of 150 marks, of which you can score at most 100 marks.
- The deadline for uploading your answers is 14:00 hours on October 10, 2020.
- You may answer any subset of questions or parts of questions. All answers will be evaluated.
- You are free to refer to the lectures/your notes/any other material that you wish, but you are **not** permitted to confer with one another in any manner.
- Warning: CMI's academic policy regarding cheating applies to this exam.

Unstated assumptions and lack of clarity in solutions can and will be used against you during evaluation. You may freely refer to statements from the lectures in your arguments. You don't need to reprove these unless the question explicitly asks you to, but you must be precise. That is, "As we saw in class ... " will not get you any credit, but "As we saw in Exercise x of Lecture y" will. Please feel free to ask us via the designated channels if you have questions about the questions.

Whenever you are asked to design an algorithm, you must make it as efficient as possible. The credit will depend on the running time of your algorithm as well, apart from its correctness.

1. Consider the procedure DEFRAGICATE described in pseudocode below. The procedure takes two non-negative integers as arguments. For a real number x the notation $\lfloor x \rfloor$ denotes the largest integer which is not larger than x. What function of the two input numbers does the procedure return? Justify your answer with a proof. **You will get the credit for this question only if your arguments correctly justify your estimate of the function.** [25]

```
DEFRAGICATE(p, q)
1  if $p == 0$
2      then return 0
3  $r \leftarrow q + q$
4  $s \leftarrow \lfloor p/2 \rfloor$
5  $t \leftarrow$ DEFRAGICATE(s, r)
6  if $p$ is even
7      then return $t$
8      else return $t + q$
```

Hint: Give a few different pairs of non-negative integers (p, q) as input to DEFRAGICATE and compute the outputs. Identify the pattern, formally state it, and prove it using induction.

2. Recall the relations $f(n) = \mathcal{O}(g(n))$ and $f(n) = \Omega(g(n))$ defined in Lecture 3. Indicate whether *each of these two relations* holds, or does not hold, for *each pair of functions* $f(n), g(n)$ in the list below. In each case, justify your answer by exhibiting appropriate constants, or proving that they don't exist.

   You will get the credit for an answer **only if** you provide a justification in terms of constants in the following way: To justify a claim that a certain relation (say $f(n) = \Omega(g(n))$) holds for a pair f, g you must *exhibit* constants (and not just prove their *existence*) which make the relevant inequalities hold. To justify a claim that a certain relation does *not* hold for a pair f, g you must prove that *no* constants exist which can make the relevant inequalities hold.

   (a) $f(n) = n^n, g(n) = n!$ [10]

   (b) $f(n) = (\log_2 n)^{\log_2 n}, g(n) = n^{100}$ [15]

3. A certain political party conducts its leadership elections in two rounds. The first round is a *run-off* election with *write-in voting*. That is: there is no pre-defined list of candidates; every member of the party has one vote, and they can vote for *any* registered member of the party (including themselves). The "vote" is done by writing on the ballot paper the party membership ID of the person for whom they want to vote. This is the "write-in" part. The *run-off* part means that if some member gets *strictly* more than half of the votes—that is, if their ID is written on strictly more than half the ballots cast—then they win the election, and there is no second round. A member who wins this way is called a *run-off winner*. If there is no run-off winner then the top two candidates from the first round compete in a second round of "normal" voting. [25]

   Your task is to design a procedure RO-WINNER for deciding if there is a run-off winner in the first round. The input to RO-WINNER is an array $V[1 \ldots n]$ of votes, where each vote $V[i]$ is the membership ID of one member or another. RO-WINNER(V) returns:

   1. **None** if there is no run-off winner for the set of votes V, and
   2. The ID of the run-off winner for V, if there is one.

   This is all that RO-WINNER(V) returns; in particular, it does *not* return the number of votes for the run-off winner (if there is one). Assume for the sake of simplicity that n is a power of 2.

(a) Let $V_1 = V[1 \dots \frac{n}{2}], V_2 = V[\frac{n}{2}+1 \dots n]$. Prove that there is a way to compute RO-WINNER(V) from the values of RO-WINNER(V_1) and RO-WINNER(V_2).

(b) Write the pseudocode for an algorithm that uses the above idea to compute RO-WINNER(V) in time $o(n^2)$. Note that this is *small-Oh* of n^2.

(c) Argue that your pseudocode correctly computes RO-WINNER(V).

(d) Analyze the running time of your pseudocode and show that it indeed runs in $o(n^2)$ time.

Note: You will get credit for this question *only if both* your pseudocode (part (b)) *and* your justification (parts (a) and (c)) are correct. Merely writing correct pseudocode will NOT get you *any* credit.

4. You have a machine on which only one job can be scheduled at any point of time. You have n jobs (a_i, d_i), $i = 1, \dots, n$ where a_i is the time at which the ith job arrives, and d_i is the number of time slots required for the ith job. Assume that all the time slots are integers. For finishing job i, you need to assign d_i (not necessarily consecutive) slots to job i. You obviously cannot schedule job i in the first $a_i - 1$ slots, since it is not available before the time slot a_i. The goal is to minimize the sum of finish times of all the jobs. Design an algorithm, analyze its time complexity, and prove that it is correct. [25]

5. You are given a rooted binary tree on n vertices, and an integer $k \leq n$. You need to color k vertices in the tree so that each vertex has a *colored ancestor* nearby. A *solution* to this problem is the set of k vertices that you colored. The *cost* of a solution S is given by $\text{cost}(S) = \max_v \text{cost}_v(S)$ where the maximum is taken over all vertices v in the tree and $\text{cost}_v(S)$ is the distance of v to its nearest colored ancestor in S. Note that the cost of a solution which does not include the root is ∞. [25]

   Design an algorithm that computes a solution of the least cost. Prove that your algorithm is correct and analyze its time complexity.

6. In this problem you sort the alphabets in your own name using heapsort.

   (a) Construct an array N of length 10 from your name as you wrote it at the beginning of this answer sheet, converted to all uppercase and without punctuation. If your name has more than 10 letters then delete a suffix to make N. If your name has fewer than 10 letters then add a prefix of "ZYXWVUTSRQ" to its end. Here are some examples derived from the names of CS faculty members at CMI: [0]

   - [P, R, A, J, A, K, T, A, N, I]
   - [G, P, H, I, L, I, P, Z, Y, X]
   - [P, R, A, V, E, E, N, M, Z, Y]

(b) Show the steps involved in sorting the array N *constructed from* **your** *name*, using *heapsort*. [25]