Here are some extra practice problems taken from past semesters. You are welcomed to check your solutions with me or discuss them in the telegram group chat.

1. **Greedy Algorithms** (AY 24/25 Sem 2). You are arranging a summer camp for n students. Student i stays during the period $[a_i, b_i]$, where a_i and b_i are positive integers such that $a_i \leq b_i$. Assume that students who arrive earlier also leave no later: $a_1 \leq a_2 \leq \ldots \leq a_n$ and $b_1 \leq b_2 \leq \ldots \leq b_n$. You would like to hold a number of parties in such a way that every student can attend at least one party. (Each party occurs at an integer time t. Student i can attend any party that occurs at a time t such that $a_i \leq t \leq b_i$.) Since parties are expensive, you would like to minimize the number of parties held. Design an analyze an algorithm to compute this number, and state the running time of your algorithm in terms of n using the O-notation.

2. **Greedy Algorithms** (AY 23/24 Sem 2). You are given n events where each takes one unit of time. Event i will provide a profit of g_i dollars ($g_i > 0$) if started at or before time t_i where t_i is an arbitrary real number. (Note: If an event is not started by t_i then there is no benefit in scheduling it at all. All events can start as early as time 0.) Also only one event can be scheduled at any particular time interval. Give as efficient algorithm as you can, to find a schedule that maximizes the total profit.

3. **Greedy Algorithms** (AY 23/24 Sem 2). Given a set of keys $1, 2, \cdots, n$, where the key i has weight w_i. The weight of the key reflects how often the key is accessed, and thus heavy keys should be higher in the tree. The Optimal Binary Search Tree problem is to construct a binary-search tree for these keys, in such a way that
$$wac(T) = \sum_{i=1}^{n} w_i d_i$$
is minimized, where d_i is the depth of key i in the tree (note: here we assume the root has a depth equal to one). This sum is called the *weighted access cost (wac)*. Consider the greedy heuristic for Optimal Binary Search Tree: for keys $1, 2, \cdots, n$, choose as root the node having the maximum weight. Then repeat this for both the resulting left and right subtrees.
Prove or disprove that the greedy heuristic is optimal.

4. **Amortized Analysis** (AY 23/24 Sem 2). You are asked to maintain a list L while supporting the following two operations:
   - Insert(x, L): Insert an integer x in the list L (cost is 1).
   - ReplaceSum(L): Compute the sum of all the integers in L. Then remove all these integers, and finally just store the computed sum in L (cost is length of the list L).

   Show that the amortized cost of both Insert and ReplaceSum operation is $O(1)$. *(As an exercise, you should try to prove it twice, once using accounting method and another time using potential method.)*

5. **Amortized Analysis** (AY 23/24 Sem 2). In the Set Union problem we have n elements, that each are initially in n singleton sets, and we want to support the following operations:
   - Union(A, B): Merge the two sets A and B into one new set $C = A \cup B$ destroying the old sets.
   - SameSet(x, y): Return true, if x and y are in the same set, and false otherwise.

   We implement it in the following way. Initially, give each set a distinct color. When merging two sets, recolor the smaller (in size) one with the color of the larger one (break ties arbitrarily). Note, to recolor a set you have to recolor all the elements in that set. To answer SameSet queries, check if the two elements have the same color. (Assume that you can check the color an element in O(1) time, and to recolor an element you also need O(1) time. Further assume that you can know the size of a set in O(1) time.)
   Use Aggregate method to show that the amortized cost is $O(\log n)$ for Union. That means, show that any sequence of m union operations takes $O(m \log n)$ time. (Note, we start with n singleton sets.)

6. **Amortized Analysis** (AY 23/24 Sem 2). Suppose Alice insists Bob to maintain a dynamic table (that supports both insertion and deletion) in such a way its size must always be a Fibonacci number. She insists on the following variant of the rebuilding strategy. Let F_k denote the k-th Fibonacci number. Suppose the current table size is F_k. After an insertion, if the number of items in the table is F_{k-1}, allocate a new table of size F_{k+1}, move everything into the new table, and then free the old table. After a deletion, if the number of items in the table is F_{k-3}, we allocate a new hash table of size F_{k-1}, move everything into the new table, and then free the old table.
Use either Potential method or Accounting method to show that for any sequence of insertions and deletions, the amortized cost per operation is still $O(1)$.

7. **Reductions** (AY 23/24 Sem 2). We have seen the (undirected) Hamiltonian cycle problem in tutorial. In a similar way, we can define the Directed Hamiltonian Cycle problem as follows: Given a directed graph whether there exists a directed cycle that visits each vertex exactly once. There is a slightly intricate poly-time reduction from the 3-SAT problem to the directed Hamiltonian cycle problem (which we will not see in this course, but you may assume that the directed Hamiltonian cycle is NP-complete).
Now we ask you to give a poly-time reduction from the directed Hamiltonian Cycle problem to the (undirected) Hamiltonian Cycle problem. (Note, we have already seen in lecture that the Hamiltonian Cycle problem is is in NP. So we get that the (undirected) Hamiltonian Cycle problem is NP-complete.) [**Hint**: Represent each node in the directed version using 3 nodes in the undirected version.]

8. **NP-Completeness** (AY 23/24 Sem 2). Consider the Max-Clique problem: Given an undirected graph $G = (V, E)$ and an integer k decide whether there exists a clique of size at least k in G (i.e., as a subgraph of G). Show that Max-Clique problem is NP-complete. [**Hint**: Try a reduction from the Maximum Independent Set problem.]

9. **NP-Completeness** (AY 23/24 Sem 2). Consider the MAX-2-SAT problem: Given a 2-CNF formula ϕ with m clauses and an integer $1 \leq k \leq m$, decide whether there is an assignment to the variables that satisfies at least k clauses of ϕ. Show that MAX-2-SAT is NP-complete. [**Hint**: Try a reduction from 3-SAT.]

Hints

1. Start with a greedy choice. Which party can be held in some optimal solution?
2. Start with a greedy choice. Which events should we prioritize?
3. Can we somehow "trick" this heuristic, leveraging the binary search tree property?
4. This should be similar to Push and Multipop discussed in tutorial.
5. Consider a singleton set at the beginning. Prove that it can be recolored at most $\log n$ times. (What is "doubled" after a recoloring?) Next, prove that there are at most $O(m)$ singleton sets involved in m operations.
6. It is similar to the dynamic table (Insertion + Deletion) in the optional segment of the tutorial. When the size of the table is closer to shrinking then save money/potential for the shrinking, otherwise save money/potential for the doubling.
7. Duplicate a vertex v 3 times into v, v', v''. Draw edges between (v, v') and (v', v''). Then, draw the directed edges. In any Hamiltonian cycle in the undirected graph, you can recover either the Hamiltonian cycle in the directed graph, or its opposite (counterclockwise) version.
8. Independent set should simply be the opposite of clique.
9. Given a clause in 3-SAT $x_i \vee y_i \vee z_i$, we should create 10 clauses, w_i, x_i, y_i, z_i, $\overline{x_i} \vee \overline{y_i}$, $\overline{x_i} \vee \overline{z_i}$, $\overline{x_i} \vee \overline{z_i}$, $\overline{w_i} \vee x_i$, $\overline{w_i} \vee y_i$ and $\overline{w_i} \vee z_i$. Intuitively, the clause $x_i \vee y_i \vee z_i$ has a true value iff 7 of these clauses can be true at the same time (by choosing w_i appropriately).