There are some exercises (for practice, and to teach you ideas you may need to solve problems) and four problems on this assignment. **Please solve problem #1 by yourself, you are allowed to collaborate for the other problems.** Please limit yourself to groups of two (or at most three) people. We adhere to a “whiteboard” collaboration policy: you should discuss with your partner on whiteboards (or equivalents) but then should go away and write down your solutions yourself. You must not share written work. **You must write down the names of the person(s) you collaborate with.** In general, you should solve the homework problems using only to material we refer you to (or put on the course page), and not books or other online resources. If you have reason to use any resources we point you to (except the course notes), please cite them.

A word of advice: please try to solve the problem by yourself before working with your partner(s). That is the way to improve your problem-solving skills.

Homeworks will be due at 11:59pm on the due date on gradescope. Corrections and changes will appear on the course webpage and on Piazza, please check them regularly.

---

**Exercises (just for practice, not for submission)**

1. **(Coin-Flipping)**
   - (a) Suppose you have access to a bent coin, that randomly outputs a 1 with some fixed but unknown probability \(p \in (0,1)\), and a 0 with probability \(1-p\). Give an procedure that uses flips of this bent coin to implement a fair coin: i.e., a procedure that outputs “heads” with probability exactly 1/2. What’s the expected number of times you flip the bent coin?

   (b) Now suppose you have access to a fair coin! Given the binary representation for some value \(p \in [0,1]\), give an algorithm for implementing a coin with bias \(p\) that only flips the fair coin few times. What is the expected number of times you flip the fair coin?

   (c) You want to sample uniformly at random from the set of all \(n\)-bit strings that are balanced, i.e., that contain exactly \(n/2\) 0’s and \(n/2\) 1’s. You do the following: take a uniform random sample \(\omega\) from the set of all \(n\)-bit strings. Output \(\omega\) if it is balanced, else reject, and sample again. Show that the expected number of times you sample an \(\omega\) is \(O(\sqrt{n})\). (Hint: look up Stirling’s formula!) You should also try to show that the expected number of samples is \(\leq n + 1\), by arguments from first principles.

2. **(Minimum Spanning Tree)**
   For each of the following three statements, decide whether it is true or false. If it is true, give a short explanation. If it is false, give a counterexample.

   - (a) Suppose we are given an instance of the Minimum Spanning Tree Problem on a graph \(G\), with edge costs that are all positive and distinct. Let \(T\) be the minimum spanning tree for this instance computed by Kruskal’s algorithm. Now suppose we replace each edge cost \(c_e\) by its square root, \(\sqrt{c_e}\), thereby creating a new instance of the problem with the same graph but different costs. Is \(T\) still a minimum spanning tree for this new instance? Prove or give a counterexample.
(b) Suppose we wish to design a spanning network for which the most expensive edge is as cheap as possible. More specifically, we define the bottleneck edge of a spanning tree $T$ to be the edge of $T$ with the greatest cost. A spanning tree $T$ of a graph $G$ is a minimum-bottleneck spanning tree if there is no spanning tree $T'$ of $G$ with a cheaper bottleneck edge.

Is every minimum-bottleneck tree of $G$ a minimum spanning tree of $G$? Prove or give a counterexample.

(c) Is every minimum spanning tree of $G$ a minimum-bottleneck tree of $G$? Prove or give a counterexample.

3. (Super Independent Hashing)

Suppose $\mathcal{H}$ is a family of hash functions mapping from a large input space $\mathcal{X}$ to some smaller output space $\mathcal{Y}$, and let $M := |\mathcal{Y}|$ be the number of elements in the output space. We said that $\mathcal{H}$ was $k$-wise independent if, for any $k$ inputs $x_1, \ldots, x_k \in \mathcal{X}$ and $k$ outputs $y_1, \ldots, y_k \in \mathcal{Y}$, we have

$$P_{h \sim \mathcal{H}} (h(x_1) = y_1, \ldots, h(x_k) = y_k) \leq \frac{1}{M^k}.$$ 

One could imagine defining a stronger condition called super $k$-wise independence as:

$$P_{h \sim \mathcal{H}} (h(x_1) = y_1, \ldots, h(x_k) = y_k) < \frac{1}{M^k}.$$ 

Does there exist a family of hash functions $\mathcal{H}$ satisfying super $k$-wise independence? If so, provide an explicit construction. If not, provide a proof that such a family is impossible.

Problems

Remember to justify the correctness of the algorithms you design.

1. Asymptotic Notation and Recurrences (25 pts)

(a) For each list of functions, order them according to increasing asymptotic growth. Provide a brief argument justifying each successive step in the ordering.

List 1, fast growing functions: $2^{3n}, 3^{2n}, n!$.
List 2, slow growing functions: $2^{\log n}, \log(3^n), (\log n)^5, \log(n^5)$.

In this course, we use $\log = \log_2$ (i.e., logarithm with base 2) and $\ln = \log_e$. When we say $\log$ (and do not specify the base), it is usually when we care only about the asymptotics and the base does not matter: it can be set to any constant.

(b) Solve the following recurrences, giving your answer in $\Theta$ notation. For each of them, assume the base case $T(x) = 1$ for $x \leq 5$. Show your work.

a) $T(n) = 3T(n/4) + n$.
b) $T(n) = T(n-2) + n^4$.
c) $T(n) = \sqrt{n}T(\sqrt{n}) + n$.

You may use the Master theorem from the following handout. Please do not use the one from Wikipedia, or from other sources.
2. **Fast Multiplication** (25 pts)

Given two $n$-bit strings, you are asked to output their product in the form of a binary string. For instance, if you received 101 and 11 as inputs, you should output 1111. The time complexity of the algorithm in this problem is the number of basic bitwise operations, such as AND, OR, XOR. You may also assume that you have access to primitive operations of addition and subtraction of $n$-bit strings which use $O(n)$ basic bitwise operations.

(a) Show the naïve algorithm we learnt from elementary school (potentially kindergarten) has time complexity $O(n^2)$.

(b) Show a faster algorithm with time complexity $o(n^2)$ and state the time complexity in $O$ notation.

**Hint 1:** Use the same trick as we did in class to solve matrix multiplication recursively.

**Hint 2:** Multiplying two 2-bit numbers $ab$ and $cd$ can be done with 4 multiplications. Can you do it with just 3? (Additional hint: expand $(a - b)(c - d)$.)

3. **Supervising committee.** (25 pts)

You are in charge of $n$ shifts that need to be staffed. Each shift is a single contiguous interval of time, and there can be multiple shifts going on at the same time (i.e. shifts may overlap). If a worker is assigned to a shift, they must work the whole shift (from start to end, inclusive).

(a) Assume that a worker can work any number of non-overlapping shifts. Since there is a shortage of workers, give a greedy algorithm that takes the schedule of $n$ shifts, and outputs the maximum number of shifts that a single worker could work (the total length of the shifts is not important). The algorithm should have a running time of $O(n \log n)$.

(b) Suppose now that there are exactly $n$ workers, and each one has already been assigned to a single shift. Your objective is to choose a subset of these $n$ workers to form a supervising committee. Such a committee is considered complete if, for every worker $w$ not in the committee, worker $w$’s shift overlaps (at least partially) the shift of some worker who is in the supervising committee. In this way, each worker’s performance can be observed by at least one person who’s in the committee. Give a greedy algorithm that takes the schedule of $n$ shifts, and produces a complete supervising committee containing as few workers as possible. For full credit, please give an algorithm with runtime $O(n^2)$.

4. **(Dynamic Binary Search)** (25 pts)

Binary search of a sorted list of length $m$ takes $\log_2 m$ time, but the time to insert a new element in such a sorted list is linear in the size of the array. We can improve the time for insertion by keeping several sorted arrays.

Specifically, we wish to make a data structure that supports two operations: SEARCH and INSERT. Suppose at some point the data structure has $n$ elements. Let $k = \lceil \log_2(n + 1) \rceil$, and let the binary representation of $n$ be $(n_{k-1}, n_{k-2}, \ldots, n_0)$. We maintain $k$ sorted arrays $A_0, \ldots, A_{k-1}$, where for $i = 0, 1, \ldots, k-1$, the length of array $A_i$ is exactly $2^i$. Each array is either full or empty, depending on whether $n_i = 1$ or $n_i = 0$ respectively. The total number of elements held in all $k$ arrays is therefore $\sum_{i=0}^{k-1} n_i 2^i = n$. Although each individual array is sorted, elements in different arrays bear no particular relationship to each other.

(a) Describe how to perform the SEARCH operation for this data structure, such that given any sequence of $n$ INSERT and SEARCH operations, the worst-case search time is $O(\log^2 n)$ per operation.
(b) Describe how to perform the INSERT operation. Show an amortized runtime of $O(\log n)$ per INSERT operation, given any sequence of $n$ INSERT and SEARCH operations. What is the best upper bound you can give on the worst-case runtime?

For this problem, you may assume that the cost to allocate a new array is negligible. However, writing any element into an array takes unit time.