CS41 Homework 2

This homework is due at 11:59PM on Tuesday, September 29. Write your solution using LaTeX. Submit this homework in a file named `hw02.tex` using **github**. The homework should be completed and submitted individually. It's ok to discuss approaches at a high level. In fact, you are encouraged to discuss general strategies. However, you should not reveal specific details of a solution, nor should you show your written solution to anyone else. Please ask your professor if you have any questions about the academic integrity policy.

*Note on problems labeled as "**extra challenge**"*: These are entirely **optional** problems, purely to provide some additional challenge to those interested. They are **not** required. There is **no penalty** for not solving them. I am happy to discuss these problems with anyone who is interested, but there are **no bonus points** for solving them.

The main **learning goals** of this homework are to get practice with thorough arguments, practice algorithmic design and analysis, and work with the formal definition of Big-Oh.

1. **Asymptotic analysis.** Arrange the following functions in ascending order of growth rate. That is, if g follows f in your list, then it should be the case that $f = O(g)$.

   - $f_1(n) = \frac{\sqrt{n}}{6}$
   - $f_2(n) = 12n\log(n)$
   - $f_3(n) = 5\log(n)^4$
   - $f_4(n) = \pi \cdot 2^n$
   - $f_5(n) = 7n^3$
   - $f_6(n) = 16n^2 + 22n$

   No proofs are necessary.

2. **Properties of Asymptotics.** Let f, g, and h be functions such that f is $O(g)$, f is $\Omega(h)$, and g is $\Theta(h)$. Prove or disprove: f is $\Theta(g)$.

3. **Asymptotic Analysis.** Let $f(n) = n^2\log(n) + 17n - 4$ and $g(n) = \frac{n^{2.5}}{2} - n$. Prove that $f(n) = O(g(n))$. You may use techniques and facts from class and the textbook. Your proof should be complete and formal.

4. **Practice with proofs.** Sometimes a proof which follows the correct structure can have a subtle problem. Consider the following proof.

   Claim 1. *Linear Search runs in constant — i.e., $O(1)$ — time in the worst case.*

   Proof. (by induction) We will rely on a key fact, which is that if two functions f and g are both $O(h)$ for some other function h, then $f + g$ is $O(h)$

   - base case: Consider performing linear search on a list of size $n = 1$. The algorithm looks at the single element to determine if it matches the element being searched for, and then we are done. This runs in constant time in total, so this is $O(1)$.

- inductive hypothesis: Assume that if we have a list of $n = k$ elements, linear search runs in $O(1)$ time in the worst case.
- inductive step: Let's say we have a list of size $n = k + 1$. Let this list be called A, so that the elements are, in order, $A[0], A[1], \ldots, A[n-2], A[n-1]$.

  Linear search will check the final element last. That is, we can think of this as first performing a linear search on the first $n-1$ elements (i.e., on $A[0], A[1], \ldots, A[n-2]$); if the desired element is not found, then linear search checks the last element, $A[n-1]$, to see if this is a match.

  Since checking the first $n-1$ elements is equivalent to performing a linear search on a list of size $n-1 = k$, by the inductive hypothesis this takes constant time in the worst case.

  Finally, if the element was not found, we finally check $A[n-1]$, which takes only constant time.

  Since both steps (checking the first $n-1$ elements, and then checking the last element) take $O(1)$ time in the worst case, the total time taken is therefore $O(1)$ (by the fact above that the sum of two functions which are both $O(1)$ is also $O(1)$). To be precise: if we let $f(n)$ be the number of steps needed to check the first $n-1$ elements and $g(n)$ be the number of steps needed to check the final element, the above argument says that $f(n)$ is $O(1)$ (by the inductive hypothesis) and $g(n)$ is $O(1)$ (since we only need to check one element), and the fact above says that since f and g are both $O(1)$, $f + g$ is also $O(1)$.

Therefore, linear search runs in time $O(1)$ in the worst case. □

It seems like something is probably wrong with this argument, since linear search is known to take $\Omega(n)$ time in the worst case. Explain what is wrong with the proof.

5. **Close to sorted.** Say that a list of numbers is "k-close-to-sorted" if each number in the list is less than k positions from its actual place in the sorted order. (So a 1-close-to-sorted list is *actually* sorted.) Give an $O(n \log k)$ algorithm for sorting a list of numbers that is k-close-to-sorted.

   In your algorithm, you may use any data structure or algorithm from CS35 by name, without describing how it works.

6. **Computing Ramsey Numbers by Brute Force.** Let $G = (V, E)$ be an undirected graph with vertex set V and edge set E. A *clique* is a set of vertices K such that for any two distinct vertices in the clique (i.e., for $v, v' \in K$), there is an edge in G connecting v and v' (i.e., $\{v, v'\} \in E$). An *independent set* is a set of vertices I such that for any two distinct vertices $v, v' \in I$ in the set, there is *no* edge between v and v' (i.e., $\{v, v'\} \notin E$ for all pairs $v, v' \in I$).

   For a positive integer k, the *kth Ramsey number* $R(k)$ is the smallest number n such that every graph on n vertices is guaranteed to have either an independent set of size k or a clique of size k. In lab, we looked at $R(3)$. Here, we consider a *brute-force algorithm* for finding $R(k)$ for any given k.

First, observe that if we can find a graph with n vertices that has *neither* an independent set of size k *nor* a clique of size k, then it must be the case that $R(k) > n$ (e.g., in lab, finding a graph with five vertices that had no clique nor independent set of size 3 implied that $R(3) > 5$). On the other hand, if we can somehow verify that *every* graph with n vertices must have at least one independent set or clique of size k, then we can conclude that $R(k) \leq n$.

Our brute-force algorithm is the following: For a given n, we can check whether n is an upper or lower bound on $R(k)$ with CheckRamsey below:

CheckRamsey(n, k)

```
1  for each graph G with n vertices:
2      if G has no independent set of size k and no clique of size k:
3          return R(k) > n
4  return R(k) ≤ n // every graph had an independent set or a clique of size k
```

Answer the following questions:

(a) Let $C(n)$ denote the number of different possible graphs with n vertices. Find an *exact* formula for $C(n)$, and briefly justify it. Your formula should *not* be in terms of asymptotic notation (Big-Oh, Big-Omega, etc.).

(**Hint:** How many possible edges can there be in an undirected graph with n vertices? Then note that for each possible edge, we may choose between two options: include the edge in the graph, or not.)

(b) Use your answer to the previous part to give an asymptotic lower bound on the *worst-case* running time of CheckRamsey. What is the worst case? **Note:** You should assume that generating the next graph to check, in line 1, requires at least one step, and that checking any single graph for cliques and independent sets of size k (that is, evaluating line 2 in the algorithm) requires *at least* one step.

(You may think that the check in line 2 should take much more than constant time, and you'd be right. In this case, we are okay with the fact that our lower bound is not tight.)

(c) Let us be optimists: Suppose I have a magic box that can tell me in a single step (i.e., constant time) whether any given graph has an independent set or clique of size k, so that line 2 runs in time $O(1)$, and give an asymptotic upper bound, in Big-Oh notation, for the *best-case* running time of CheckRamsey. You should assume that generating a graph to check (in line 1) takes time $O(n^2)$.

(**Hint:** What happens if $R(k) > n$, and we get lucky so that the *first* graph we check has no independent set nor clique of size k?)

(d) To compute $R(k)$ for any given value of k, we can simply run CheckRamsey(n, k) for all values of n until we find the *smallest* one for which the procedure returns "$R(k) \leq n$".

We might wonder whether the optimistic case in the previous part might happen frequently so that computing Ramsey numbers in this way is not so bad.

Argue that, no matter how lucky we get, trying to compute $R(k)$ in this brute-force way will require time at least $\Omega(C(R(k)))$.

(**Hint:** Consider what happens when CheckRamsey(n, k) is run for $n = R(k)$.)

(e) It is currently known that $R(5) > 42$, and there is some reason to believe that in fact $R(5) = 43$. How big is $C(43)$? What does this say about how hard it is to compute $R(5)$?

(f) **(extra challenge)** A famous result of Paul Erdős showed that $R(k)$ is $\Omega(k2^{k/2})$. What does this tell us about the overall running time of our brute force approach for larger values of k? Is there any hope for $R(6)$?

(g) **(extra challenge)** We may think that we can be a bit more clever about what graphs we check, to reduce the total number of graphs we need to consider in CheckRamsey. For example, if the vertices of a graph are labeled $1, 2, \ldots, n$, we can *relabel* the vertices of a graph to get a new graph that is essentially a duplicate of the original (formally, we say they are *isomorphic*). The for loop in CheckRamsey would consider both of these graphs, which for our purposes results in doing redundant work. Convince yourself that for any graph on n vertices, there are $n!$ ways to relabel its vertices. Thus, we may end up checking isomorphic duplicates of some graphs up to $n!$ times. Suppose we have a way to reduce the set of graphs we check so that we do not consider any such relabeled (i.e., isomorphic) versions of a graph we have already checked (and note that, as we have just argued, we will skip at most $n!$ duplicates for each graph). Show that even with this, the total number of graphs we need to check is $2^{\Omega(n^2)}$.

7. **Paths in graphs.** A path P in a graph $G = (V, E)$ is a sequence of vertices $P = [v_1, \ldots, v_k]$ such that for all $1 \leq i < k$ there is an edge $(v_i, v_{i+1}) \in E$. Path P is *simple* if all v_i's are distinct.

   In this problem, you will examine different graphs and consider how many different paths can exist in the graph. For each of the below, give a general description that works for any $n \geq 3$.

   (a) Describe a graph G_1 on n vertices where between any two distinct vertices there are zero simple paths.

   (b) Describe a graph G_2 on n vertices where between any two distinct vertices there is exactly one simple path.

   (c) Describe a graph G_3 on n vertices where between any two distinct vertices there are exactly two simple paths.

   (d) **(extra challenge)** Describe a graph $G_4 = (V, E)$ on n vertices and two distinct vertices $s, t \in V$ such that there are $2^{\Omega(n)}$ simple $s \rightsquigarrow t$ paths.

8. **(extra challenge)** For these problems, your example functions should have domain and range the positive integers $\mathbb{N}$.

   - Find (with proof) a function f_1 such that $f_1(2n)$ is $O(f_1(n))$.
   - Find (with proof) a function f_2 such that $f_2(2n)$ is not $O(f_2(n))$.