CSC373  Fall’20
Assignment 1 Solutions: Dinosaurian Dilemmas

Q1 [25 Points] Fruit Frenzy

Dinosaur Bobby is hungry and wants to eat some fruits. He knows about \( n \) fruit gardens. Each garden \( i \) is described by two integers: \( e_i \), the easiness of reaching garden \( i \) from where Bobby is, and \( q_i \), the number of fruits in garden \( i \).

Bobby wants to pick a garden \( i \) to visit. But he wants to make a smart choice: he wants to make sure that no other garden \( j \) is simultaneously easier to get to (\( e_j > e_i \)) and has more fruits (\( q_j > q_i \)). Bobby wants your help in counting the number of gardens \( i \) that meet this requirement.

(a) [15 Points] Give an \( O(n \log n) \) time divide-and-conquer algorithm for this problem.

(b) [10 Points] Provide a brief justification of its correctness and running time.

Solution to Q1

(a) The high-level algorithm works as follows.

1. Sort all the gardens in a nondecreasing order of their \( e_i \)-s (i.e., so \( e_1 \leq e_2 \leq \ldots \leq e_n \)).

2. Call the recursive function \textsc{BestGardens} with this sorted list as the first parameter and 0 as the second parameter. Let the function return \((\text{list}, \text{value})\).

3. Return the number of gardens in \text{list}.

The recursive function \textsc{BestGardens} is given below. For a description of what the algorithm does, see the justification of correctness in (b).

\textbf{Algorithm 1:} \textsc{BestGardens}(inputList, \text{einputThr})

\begin{align*}
1 & \text{ Break inputList into two (roughly) equal halves at the middle: leftList and rightList} \\
2 & \text{ \( e_{\text{thr}} \leftarrow \) e-value of the last garden in leftList} \\
3 & \text{ (leftOutput, \(-\), \(-\)) \leftarrow \textsc{BestGardens}(leftList,0)} \\
4 & \text{ (rightOutput, rightMaxFruits, rightMaxFruitsAboveThr) \leftarrow \textsc{BestGardens}(rightList, e_{\text{thr}})} \\
5 & \text{ leftOutputFiltered} \leftarrow \text{ list of gardens } i \text{ from leftOutput such that } e_i < e_{\text{thr}} \text{ and } q_i < \text{rightMaxFruits}\text{, or } e_i = e_{\text{thr}} \text{ and } q_i < \text{rightMaxFruitsAboveThr} \\
6 & \text{ maxFruits} \leftarrow \text{ maximum } q\text{-value among all gardens in inputList} \\
7 & \text{ maxFruitsAboveThr} \leftarrow \text{ maximum } q\text{-value among all gardens } i \text{ in inputList with } e_i > e_{\text{inputThr}} \\
8 & \text{ return} \ (\text{leftOutputFiltered} \cup \text{rightOutput, maxFruits, maxFruitsAboveThr})
\end{align*}

(b) Justifications:

\textbf{Running time:} Sorting the gardens takes \( O(n \log n) \) time. Let \( T(n) \) denote the worst-case running time of \textsc{BestGardens} on a list of size \( n \). Then, Lines 3 and 4 take \( T(n/2) \) time, line 2 takes \( O(1) \) time, and lines 5, 6, and 7 take \( O(n) \) time. Hence, the worst-case running time is given by the equation \( T(n) \leq 2T(n/2) + O(n) \), which gives \( T(n) = O(n \log n) \).
Correctness: We say that garden $i$ is dominated by garden $j$ when $e_j > e_i$ and $q_j > q_i$, and call garden $i$ undominated if it is not dominated by any other garden.

When we recursively call `BestGardens` on the two halves (leftList and rightList), we get the lists (leftOutput and rightOutput) of gardens from each half that are not dominated by other gardens from the same half. Since gardens that are undominated must be part of this list, our desired list is a subset of the union of the two lists.

Further, each garden $i$ in rightOutput must be undominated: it is not dominated by any other garden from rightList, and all gardens $j$ from leftList have $e_j \leq e_i$ due to the sorting. Finally, for each garden $i$ in leftOutput, we already know that it is not dominated by any other garden from leftList, so we only need to make sure that it is also not dominated by any garden $j$ from rightList.

If all the $e$-values were distinct, then we would know that all gardens from rightList have higher $e$-value than any garden from leftList (and thus from leftOutput). Hence, we would only need to know the maximum $q$-value among gardens from rightList in order to decide which gardens from leftOutput to rule out. However, there is a slight subtlety: some gardens with equal $e$-values may get split between the left and right halves.

Hence, if $e_{\text{thr}}$ denotes the greatest $e$-value from leftList (i.e. the $e$-value of the last garden in leftList), then we need to check a garden $i$ from leftOutput against:

- maximum $q$-value among those gardens $j$ in rightList with $e_j > e_{\text{thr}}$, if $e_i = e_{\text{thr}}$;
- maximum $q$-value from rightList, if $e_i < e_{\text{thr}}$ (as any garden $j$ from rightList has $e_j \geq e_{\text{thr}} > e_i$).

Function `BestGardens` is amended such that $e_{\text{thr}}$ is passed as an argument to the call on the right half, which then returns precisely these two additional values (the additional parameter passed to the call on the left half or its additional return values do not matter). Then, Line 5 filters gardens from leftOutput using these two values, and Lines 6 and 7 compute these values in the parent call.

Q2 [25 Points] Missing Member

The dinosaur society has reached a magical point where it consists of $m = 2^n$ dinosaurs, each with a *distinct* age from the set $\{0, 1, \ldots, 2^n - 1\}$. One day, an assembly is called, and only $m-1$ dinosaurs show up. Chief Seema enlists your help to find the age of the missing dinosaur.

She has lined up the dinosaurs (including herself) in an arbitrary order. You can walk up to any dinosaur and query their age. However, the dinosaurs are ex-computer-scientists and like binary encodings. So they will only provide a single bit of their age at a time. That is, in one query, you can walk up to some dinosaur $i$ and ask the $j$-th bit of their age, where $i \in \{1, \ldots, m\}$ and $j \in \{1, \ldots, n\}$.

Design an algorithm to find the age of the missing dinosaur in $O(m)$ queries and $O(m)$ running time.

*Note: Each dinosaur’s age is represented by $n$ bits, so there are a total of $mn$ bits. That means you will need to be smart and not ask every dinosaur every bit of their age.*
Solution to Q2

The idea of the algorithm is as follows:

- We run an iterative algorithm that determines the bits in the age of the missing dinosaurs one by one (say, from the most significant to the least). Let $X_0$ be the list of all present dinosaurs.

- Consider iteration $k$, when the first $k - 1$ bits have already been determined.

- Let $X_{k-1}$ be the set of dinosaurs present such that the first $k - 1$ bits of their age match those of the age of the missing dinosaur. (We will maintain this list as the algorithm progresses.)

- Note that out of $\{0, 1, \ldots, 2^n - 1\}$, there are precisely $2^{n-k+1}$ numbers whose first $k - 1$ bits take given values. Hence, $|X_{k-1}| \leq 2^{n-k+1}$.

- Out of these, precisely $2^{n-k}$ have the next ($k$-th) bit as 0, while the remaining $2^{n-k}$ have the next bit as 1.

- Thus, by querying all the dinosaurs from $X_{k-1}$ the $k$-th bit of their age, we can determine the $k$-th bit of the age of the missing dinosaur.

- Before proceeding to the next iteration, we let $X_k$ be the list of those dinosaurs from $X_{k-1}$ whose $k$-th bit also matches that of the missing dinosaur.

This is formally presented as the algorithm below. In iteration $k$, the number of queries made (in line 4) as well as the running time (of lines 4 and 6) is $O(|X_{k-1}|)$. Hence, the overall running time and number of queries is $O(2^n + 2^{n-1} + \ldots + 2) = O(2^{n+1}) = O(m)$.

Algorithm 2: FindMissing

1. Let $k \leftarrow 0$, $X_0 \leftarrow \{1, \ldots, m - 1\}$ (indices of present dinosaurs)
2. while $k < n$ do
3.     $k \leftarrow k + 1$
4.     Query the $k$-th most significant bit of the age of all dinosaurs from $X_{k-1}$, and for $y \in \{0, 1\}$, let $n_y$ be the number of dinosaurs who give the answer $y$
5.     $b_k \leftarrow y^*$ such that $n_{y^*} < 2^{n-k}$ (there will be a unique such $y^*$)
6.     $X_k \leftarrow$ those dinosaurs from $X_{k-1}$ who responded $y^*$ in line 4
7. end
8. return the age of the missing dinosaur as $b_1b_2\ldots b_n$

Q3 [25 Points] Building Hospitals

There are $n$ dinosaur societies which live along a common one-dimensional road. The members of society $i$ live in the interval $[s_i, f_i]$, and these intervals can overlap. The chiefs of the societies recognize that they need to build hospitals in order to take care of their members. But resources are scarce, so they enlist your help.

Your goal is to design an algorithm which finds locations along the road where hospitals should be built such that each interval $[s_i, f_i]$ contains at least one hospital. Subject to this requirement, you need to minimize the number of hospitals built.
(a) [10 Points] Design a greedy algorithm for this purpose and analyze its worst-case running time.

(b) [15 Points] Prove correctness of your algorithm.

Solution to Q3

(a) The algorithm is as follows. It sorts the intervals in a nondecreasing order of their right endpoints, greedily considers them in that order, and builds a hospital at the right endpoint of any interval that doesn’t already have a hospital. To check if the current interval \( i \) contains any hospital, we only need to check if it contains the last built hospital, which will be to the right of other hospitals and to the left of \( f_i \) due to the sorting.

\[
\text{Algorithm 3: BUILDHOSPITALS}(s_1, \ldots, s_n, f_1, \ldots, f_n)
\]

1. \( G \leftarrow \emptyset \) (greedy solution)
2. \( h \leftarrow -\infty \) (location of last built hospital)
3. Sort the intervals by the \( f_i \)-s, so \( f_1 \leq \ldots \leq f_n \)
4. for \( i = 1, \ldots, n \) do
   5.     if \( h < s_i \) then
   6.         \( G \leftarrow G \cup \{f_i\} \)
   7.         \( h \leftarrow f_i \)
   8.     end
9. end
10. return \( G \)

Running time: Sorting takes \( O(n \log n) \) time and the loop takes \( O(n) \) time as each interval is processed in \( O(1) \) time. Hence, the overall running time is \( O(n \log n) \).

(b) We say that a hospital covers an interval if it belongs to the interval. We say that a set of hospitals covers an interval if the interval contains at least one of the hospitals from the set.

Let \( G = \{f_{i_1}, f_{i_2}, \ldots, f_{i_r}\} \). Let \( OPT \) be an optimal solution such that the \( k \) leftmost hospitals in \( OPT \) are precisely at \( f_{i_1}, \ldots, f_{i_k} \), and \( k \) is maximum possible. If \( k = r \), then \( G \) is an optimal solution, so we are done.

Suppose \( k < r \). We derive a contradiction by constructing a new optimal solution \( OPT' \) whose \( k + 1 \) leftmost hospitals are precisely at \( f_{i_1}, \ldots, f_{i_{k+1}} \).

Consider the \( (k+1) \)-st leftmost hospital in \( G \) at \( f_{i_{k+1}} \). Let \( X \) be the set of hospitals in \( OPT \) that lie in \( (f_{i_k}, f_{i_{k+1}}] \).

Property 1: \( |X| \geq 1 \). This is because \( OPT \) must have a hospital that covers interval \( i_{k+1} \), and since \( f_{i_k} \) does not cover it, it must lie in \( (f_{i_k}, f_{i_{k+1}}] \).

Property 2: \( X \cap \{f_{i_1}, \ldots, f_{i_k}\} = \emptyset \). This is because \( X \) only contains hospitals to the right of \( f_{i_k} \).

Let \( OPT' = (OPT \setminus X) \cup \{f_{i_{k+1}}\} \). We make three claims:

1. \( OPT' \) is a valid solution. By Property 2, \( OPT' \) still contains \( \{f_{i_1}, \ldots, f_{i_k}\} \), which covers intervals 1 through \( i_{k+1} - 1 \). Interval \( i_{k+1} \) is covered by \( f_{i_{k+1}} \). Any interval \( j > i_{k+1} \) that was covered by a hospital at \( h \in X \) is also covered by \( f_{i_{k+1}} \) because \( f_j \geq f_{i_{k+1}} \geq h \).
2. $OPT'$ is also an optimal solution. By Property 1, we are removing at least one hospital, and we are adding only one at $f_{i_{k+1}}$. Hence, we are not increasing the number of hospitals compared to $OPT$.

3. The $k + 1$ leftmost hospitals in $OPT'$ match those in $G$. By Property 2, the $k$ leftmost hospitals in $OPT'$ are still at $f_i, \ldots, f_{ik}$. Since we remove all hospitals in $(f_{ik}, f_{ik+1}]$ and add a hospital at $f_{i_{k+1}}$, the $(k + 1)$-st leftmost hospital is now at $f_{i_{k+1}}$.

This is the desired contradiction. Hence, $G$ must be optimal.

Q4 [25 Points] Traveller’s Dilemma

Dinosaur Eli is organizing a road trip across Canada. She owns an electric car, which has a battery that can last for 200km on a single charge. If the car runs out of battery, she will be stranded. Luckily for her, $n - 1$ charging stations are located along the road she plans to take; the distance of the $i^{th}$ charging station from her starting point is $d[i]$. The distance of her final destination from her starting point is $d[n]$. Here, $d[1] < d[2] < \ldots < d[n]$.

Assume that she begins the trip with a full charge, and the trip is feasible because the distance between any two consecutive charging stations is at most 200km (i.e. $d[i+1] - d[i] \leq 200$ for all $i$).

(a) [10 Points] Design a greedy algorithm which takes the array $d$ as input and finds locations of fewest possible charging stations where Eli should stop to full-charge the battery in order to successfully complete her trip. Analyze its worst-case running time.

(b) [10 Points] Prove correctness of your algorithm.

(c) [5 Points] Now suppose that there is a cost $c[i]$ to charge the battery at charging station $i$, regardless of the current amount of power in the battery. Show that the greedy algorithm given in part (a) does not necessarily minimize the total cost of travel.

Solution to Q4

(a) The greedy algorithm is given below. The algorithm always seeks the next charging station $i$ that Eli will not be able to reach given the charge she has left, and decides to recharge at the charging station just before $(i - 1)$.

\begin{verbatim}
Algorithm 4: RECHARGE(d[1], \ldots, d[n])
1 $G \leftarrow \emptyset$ (greedy solution)
2 $h \leftarrow 0$ (location of last recharge)
3 for $i = 1, \ldots, n$ do
4     if $d[i] - h > 200$ then
5         $G \leftarrow G \cup \{d[i - 1]\}$
6         $h \leftarrow d[i - 1]$
7     end
8 end
9 return $G$
\end{verbatim}
Running time: The running time of the algorithm is trivially $O(n)$ as it takes $O(1)$ time to process each $i$ in the loop.

(b) We prove, by induction on $i$, that there exists an optimal solution that matches the greedy solution on the set of charging stations that it selects from $\{1, \ldots, i\}$. The base case of $i = 0$ is trivial. Suppose the statement holds for some $i$, and let $OPT_i$ be such an optimal solution. Let $h$ be the last charging station up to $i$ that both $G$ and $OPT_i$ select. For $i + 1$, we want to show that there is an optimal solution $OPT_{i+1}$ that matches $G$ in its selection from $\{1, \ldots, i + 1\}$. There are three cases.

1. Both $G$ and $OPT_i$ select station $i + 1$. In this case, $OPT_{i+1} = OPT_i$ works.

2. $G$ selects $i + 1$ but $OPT_i$ does not. This is not possible. Because $G$ selects $i + 1$, we must have $d[i + 1] - d[h] > 200$, in which case $OPT_i$ would not be a valid solution as Eli would run out of charge.

3. $OPT_i$ selects $i + 1$ but $G$ does not. In this case, consider $OPT_{i+1}$ obtained by replacing station $i + 1$ with $i + 2$. Because $G$ does not select $i + 1$, we must have that $d[i + 2] - d[h] \leq 200$, so Eli can feasibly reach charging station $i + 2$ under $OPT_{i+1}$, and charging later only helps in reaching the later charging stations.

Thus, the inductive hypothesis holds for $i = n$, at which point we have $G = OPT_n$, i.e., $G$ is an optimal solution.


We can ignore the last station: since we only need to reach $d[n]$, we will never recharge there. The greedy solution will skip the first charging station and go to the second since it can reach the second station. But that has a cost of 100. Instead, recharging early at the first station is still sufficient to reach $d[3]$, but has a much lower cost of 1.

**Bonus Question**

**Q5 [20 Points] No More Slaps, Please!**

There are $n$ dinosaurs in a line, sorted in a nondecreasing order of their age. You are given a number $x$ and told that there is at least one dinosaur in the line whose age is $x$. Your goal is to find one such dinosaur.

At first glance, this may seem like a simple application of binary search that you, as a computer scientist, already know by heart. That is until you remember an important detail about the dinosaur society: Dinosaurs, unlike many humans, want to seem older and get offended if you guess them to be younger than they actually are. So if you approach dinosaur $i$ and ask him “Are you $x$ years old?”, three things can happen:

1. If he is younger than $x$, he says “Oh my, I’m flattered. But no, I’m younger than $x$.”

2. If his age is exactly $x$, he says, “Why yes! How did you know that?”

3. If his age is greater than $x$, he says, “Why not! I’m older than you think.”
3. If he is older than $x$, he gets offended and slaps you.

Now, you are strong, but not *that* strong. You know you can withstand at most $k$ dino-slaps. Any more slaps and you’re done for.

Design an algorithm to search a dinosaur of the given age $x$ such that you are guaranteed to get slapped no more than $k$ times. What is the asymptotic number of queries that your algorithm makes in the worst case as a function of $k$?

[Note: Think of $k$ as a constant. As a warm start, design an $O(\sqrt{n})$ algorithm for $k = 1$. To receive any partial credit, you need to satisfactorily solve the problem at least for $k = 2$.]

Solution to Q5 Let $D[1\ldots n]$ denote the age-sorted array of dinosaurs. We show that for each constant $k \geq 0$, Algorithm SLAP$_k$ is correct and makes $O(n^{1/k+1})$ queries. We show this via induction on $k$.

**Algorithm 5: SLAP$_k(D[1\ldots n])$**

1. if $k = 0$ then
2.   for $i = 1,\ldots,n$ do
3.     Query the age of $D[i]$, and if it is $x$, return $D[i]$
4.   end
5. else
6.   for $t = 0,1,\ldots,n^{1/k+1} - 1$ do
7.     $i \leftarrow t \cdot n^{1/k+1} + 1$
8.     Query the age of $D[i]$
9.     if it is $x$, return $D[i]$
10.    if it is more than $x$, return SLAP$_{k-1}(D[(t-1) \cdot n^{1/k+1} + 1 \ldots \cdot t \cdot n^{1/k+1}])$.
11. end
12. end

In the base case of $k = 0$, the algorithm simply does a linear search in the increasing order of age. Hence, it must encounter the promised dinosaur of age $x$ without receiving any slap.

Suppose SLAP$_{k-1}$ finds the dinosaur of age $x$, receives at most $k-1$ slaps, and makes $O(n^{1/k})$ queries given $n$ dinosaurs.

In SLAP$_k$, it is easy to see that line 8 makes $O(n^{1/k+1})$ queries. Further, since the dinosaurs are sorted by age, it will either encounter the promised dinosaur of age $x$ (which it will return in line 9), or it will overshoot and receive a slap. If it receives a slap at a given value of $t$ but did not receive a slap at $t-1$, then we know that the promised dinosaur must be in the range $[(t-1) \cdot n^{1/k+1} + 1 \ldots \cdot t \cdot n^{1/k+1}]$. Since it calls SLAP$_{k-1}$ on this range, given the induction hypothesis, it will find the correct dinosaur with at most $k-1$ additional slaps. Thus, SLAP$_k$ is correct. Since SLAP$_{k-1}$ is called with $n^{1/k+1}$ dinosaurs in line 10, it will make $O(n^{1/k+1}) = O(n^{1/k+1})$ queries. Adding the queries made in line 8, we have that the total number of queries is also $O(n^{1/k+1})$. 
