CS 341, Winter 2019
Trevor Brown, Semih Salihoglu, Doug Stinson

Assignment 2 Sample Solutions

1. [*8 marks*] *Recurrence relations.*
Consider the following recurrence:

$$T(n) = \begin{cases} T(\lfloor n/2 \rfloor) + T(\lfloor n/5 \rfloor) + 50 & \text{if } n \geq 2 \\ 1 & \text{if } n = 0, 1 \end{cases}$$

Use the guess-and-check method to prove the upper bound $T(n) \in O(n^{0.7})$.
Hint: use $T(n) \leq cn^{0.7}$ as your "guess", where c is a suitable positive constant. You can determine appropriate values for the constants c and n_0 within the induction proof.

Answer: We have

$$\begin{aligned}
&T(0) = 1 \\
&T(1) = 1 \\
&T(2) = T(1) + T(0) + 50 = 52 \\
&T(3) = T(1) + T(0) + 50 = 52 \\
&T(4) = T(2) + T(0) + 50 = 52 + 1 + 50 = 103 \\
&T(5) = T(2) + T(1) + 50 = 52 + 1 + 50 = 103.
\end{aligned}$$

Since $T(0) = 1$ and $c \times 0^{0.7} = 0$ for any c, we cannot take $n_0 = 0$. Therefore we will take $n_0 = 1$. When $m \geq 5$, the evaluation of $T(m)$ only involves terms $T(n)$ with $n \geq 1$. This means that we will have to treat $n = 1, 2, 3, 4$ as base cases in the induction proof and start the induction from $n = 5$.

Suppose $T(n) \leq cn^{0.7}$ for some c to be determined later, for $1 \leq n < m$, where $m \geq 5$. Then we have

$$\begin{aligned}
T(m) &= T(\lfloor m/2 \rfloor) + T(\lfloor m/5 \rfloor) + 50 \\
&\leq c\left(\frac{m}{2}\right)^{0.7} + c\left(\frac{m}{5}\right)^{0.7} + 50 \qquad \text{(by induction, since } m \geq 5\text{)} \\
&= cm^{0.7}\left(\left(\frac{1}{2}\right)^{0.7} + \left(\frac{1}{5}\right)^{0.7}\right) + 50 \\
&= .940cm^{0.7} + 50.
\end{aligned}$$

Clearly

$$.940cm^{0.7} + 50 \leq cm^{0.7} \Leftrightarrow .06cm^{0.7} \geq 50 \Leftrightarrow cm^{0.7} \geq 833.$$

Since $m \geq 5$, we have $m^{0.7} \geq 3$, so it is sufficient if $c \geq 278$.

Now we still have to ensure that $T(n) \leq cn^{0.7}$ for $1 \leq n \leq 4$. Thus

$$\begin{aligned}
c &\geq \max\{T(n)/n^{0.7} : 1 \leq 4\} \\
&= \max\{1/1^{0.7}, 52/2^{0.7}, 52/3^{0.7}, 103/4^{0.7}\} \\
&= \max\{1, 32, 24, 39\} \\
&= 39.
\end{aligned}$$

But we already have the requirement $c \geq 278$, so $c = 278$ is a valid choice.

2. [*4 marks*] *Master theorem.*
Find an asymptotic Θ-bound for the solution to the following recurrence relation by applying the Master Theorem. Show your work.

$$T(n) = \begin{cases} 8\,T(n/5) + 12^{\log_7 n} & \text{if } n > 1 \\ 1 & \text{if } n = 1. \end{cases}$$

Answer: Observe that $12^{\log_7 n} = n^{\log_7 12} = n^{1.28}$. Here $a = 8$, $b = 5$ and $y = 1.28$. We compute

$$x = \log_5 8 = 1.29.$$

Since $x > y$, we are in case 1 and $T(n) \in \Theta(n^{\log_5 8})$ by the Master Theorem.

3. [*16 marks*] *Divide-and-conquer.*
Define the following sequence of numbers: $F_0 = 0$, $F_1 = 1$, and

$$\begin{aligned} F_{2n} &= (F_n + F_{n-1})^2 - {F_{n-1}}^2 \\ F_{2n+1} &= (F_n + F_{n-1})^2 + {F_n}^2 \end{aligned}$$

(This is in fact the Fibonacci number sequence, but you are not required to prove it.)

(a) [*4 marks*] Give a pseudocode description of an efficient divide-and-conquer algorithm to compute F_n for a given integer $n \geq 0$, based on the above definition.

Answer:

Algorithm: *Fib*(n)
if $n = 0$
  then return (0)
  else if $n = 1$
  then return (1)
  else $\left\{\begin{array}{l} \textbf{if } n \bmod 2 = 0 \\ \quad \textbf{then } \left\{\begin{array}{l} A \leftarrow \mathit{Fib}(n/2) \\ B \leftarrow \mathit{Fib}(n/2 - 1) \\ \textbf{return } ((A+B)^2 - B^2) \end{array}\right. \\ \quad \textbf{else } \left\{\begin{array}{l} A \leftarrow \mathit{Fib}((n-1)/2) \\ B \leftarrow \mathit{Fib}((n-3)/2) \\ \textbf{return } ((A+B)^2 + A^2) \end{array}\right. \end{array}\right.$

(b) [*4 marks*] Prove that $F_n \leq 2^n$ by induction, using the usual recurrence for F_n, namely, $F_n = F_{n-1} + F_{n-2}$.

Answer: We start with $n = 0$ and $n = 1$ as base cases. We have $F_0 = 0 \leq 2^0 = 1$ and

$F_1 = 1 \leq 2^1 = 2$. As an induction hypothesis, assume that $F_n \leq 2^n$ for $0 \leq n < m$, where $m \geq 2$. Then we compute

$$F_m = F_{m-2} + F_{m-1} \leq 2^{m-2} + 2^{m-1} < 2^{m-1} + 2^{m-1} = 2^m.$$

By induction, $F_n \leq 2^n$ for all $n \geq 0$.

(c) [*8 marks*] Determine a O-bound on the complexity of your algorithm from part (a) by writing down a recurrence and solving it using the Master Theorem. Here, we are interested in the *bit complexity*. Assume that the multiplication of two k-bit numbers requires $O(k^{1.59})$ time by Karatsuba and Ofman's algorithm. You can use the fact that $F_n \leq 2^n$ (which you proved in part (b)), which implies that the number of bits in F_n is at most n.

Answer: We approximate $n/2 - 1$, $(n-1)/2$ and $(n-3)/2$ by $n/2$ for the purposes of writing down the recurrence, so we can solve it using the Master Theorem. (This could be justified by a monotonicity argument, but it is not required for the purposes of the assignment.) Then, the recurrence for both even and odd n has the form

$$T(n) = 2T(n/2) + f(n).$$

The function $f(n)$ accounts for the following:

- two multiplications (squarings) of $n/2$-bit integers, which takes time $O((n/2)^{1.59}) = O(n^{1.59})$
- one addition of $n/2$-bit integers, which takes time $O((n/2)) = O(n)$, and
- one addition or subtraction of n-bit integers, which takes time $O(n)$.

Therefore $f(n) \in O(n^{1.59})$ and our recurrence is

$$T(n) = 2T(n/2) + O(n^{1.59}).$$

We have $y = 1.59$ and $x = \log_2 2 = 1$. Then $x < y$, we are in case 3, and $T(n) \in O(n^{1.59})$ by the Master Theorem.

4. [*12 marks*] *Divide-and-conquer.*
The matrices $H_0, H_1, \ldots$ are defined as follows:

$$H_0 = (1)\,,$$

and

$$H_k = \begin{pmatrix} H_{k-1} & H_{k-1} \\ H_{k-1} & -H_{k-1} \end{pmatrix}$$

for $k \geq 1$. Thus,

$$H_1 = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix},$$

$$H_2 = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix},$$

etc.

Given a integer vector $\mathbf{v}$ of length 2^k, we want to compute the vector-matrix product $\mathbf{v}H_k$. For example, if $k = 2$, then $(2, 3, 4, -1)H_2 = (8, 4, 2, -6)$.

(a) [*8 marks*] Design a divide-and-conquer algorithm to compute a matrix-vector product $\mathbf{v}H_k$. You can assume that all arithmetic operations take $\Theta(1)$ time.

Note that you should not explicitly construct the matrix H_k. Your algorithm will take two inputs, namely $\mathbf{v}$ and k, where $\mathbf{v}$ has length 2^k.

Answer:

Derivation of algorithm (and correctness): Suppose we write $\mathbf{v} = (\mathbf{w}, \mathbf{x})$, where $\mathbf{w}, \mathbf{x}$ each have length 2^{k-1}. Then

$$\mathbf{v}H_k = (\mathbf{w}, \mathbf{x}) \begin{pmatrix} H_{k-1} & H_{k-1} \\ H_{k-1} & -H_{k-1} \end{pmatrix} = (\mathbf{w}H_{k-1} + \mathbf{x}H_{k-1}, \mathbf{w}H_{k-1} - \mathbf{x}H_{k-1}).$$

Thus, we need to solve two subproblems, namely $\mathbf{w}H_{k-1}$ and $\mathbf{x}H_{k-1}$. If $\mathbf{a} = \mathbf{w}H_{k-1}$ and $\mathbf{b} = \mathbf{x}H_{k-1}$, then $\mathbf{v}H_k = (\mathbf{a} + \mathbf{b}, \mathbf{a} - \mathbf{b})$.

Therefore the D&C algorithm is as follows.

Algorithm: $\mathit{MVProd}(\mathbf{v}, k)$

$$
\begin{array}{l}
\textbf{if } k = 0 \ \ \textbf{then return } (\mathbf{v}) \\
\quad \textbf{else} \begin{cases}
\textbf{for } i \leftarrow 1 \textbf{ to } 2^{k-1} \\
\quad \textbf{do} \begin{cases} \mathbf{w}[i] \leftarrow \mathbf{v}[i] \\ \mathbf{x}[i] \leftarrow \mathbf{v}[i + 2^{k-1}] \end{cases} \\
\mathbf{a} \leftarrow \mathit{MVProd}(\mathbf{w}, k-1) \\
\mathbf{b} \leftarrow \mathit{MVProd}(\mathbf{x}, k-1) \\
\textbf{for } i \leftarrow 1 \textbf{ to } 2^{k-1} \\
\quad \textbf{do} \begin{cases} \mathbf{c}[i] \leftarrow \mathbf{a}[i] + \mathbf{b}[i] \\ \mathbf{c}[i + 2^{k-1}] \leftarrow \mathbf{a}[i] - \mathbf{b}[i] \end{cases} \\
\textbf{return } (\mathbf{c})
\end{cases}
\end{array}
$$

(b) [*4 marks*] . Write down a recurrence relation for the complexity of your algorithm and solve it using the Master Theorem. You should aim for a complexity of $\Theta(n \log n)$, where $n = 2^k$.

Answer: The recurrence relation has the form $T(n) = 2T(n/2) + f(n)$, where $f(n)$ accounts for the work done in the two **for** loops. Clearly,

$$f(n) \in O(2^{k-1}) = O(n/2) = O(n).$$

Thus the recurrence is $T(n) = 2T(n/2) + O(n)$ and its solution is $T(n) \in O(n \log n)$, as we have seen numerous times before.

5. [*18 marks*] *Greedy.*
Suppose Alice wants to see n different movies at a certain movie theatre complex. Each movie is showing on specified dates. For $1 \leq j \leq n$, suppose that movie M_j is showing from day a_j until day b_j inclusive (you can assume that a_j and b_j are positive integers such that $a_j \leq b_j$). The objective is to determine the minimum number of trips to the theatre that will be required in order to view all n movies. For simplicity, assume that it is possible to view any number of movies in a given day.

(a) [*12 marks*] Design a greedy algorithm to solve this problem, and prove your algorithm is correct.

Answer:

(1) Sort the movies $M_j = [a_j, b_j]$ in increasing order of b_j. Suppose the movies are renamed so $b_1 \leq b_2 \leq \cdots \leq b_n$.

(2) Initialize $T = \emptyset$ and $U = \{1, \ldots, n\}$. (U keeps track of the unseen movies and T records the days that visits to the theatre will be made.)

(3) While $U \neq \emptyset$ do the following:

(i) Let j^* be the smallest element in U and define $b^* = b_{j^*}$.

(ii) Set $T = T \cup \{b^*\}$.

(ii) Delete all elements j from U such that $b^* \in [a_j, b_j]$.

Here is more detailed pseudocode (not the most efficient solution, but it is acceptable), where we use arrays to store the sets T and U:

Algorithm: *Movies*$([a_1, b_1], \ldots, [a_n, b_n])$

Sort the movies $M_j = [a_j, b_j]$ in increasing order of b_j
$T \leftarrow [0, \ldots, 0]$
$\ell \leftarrow 0$
comment: ℓ denotes the number of elements in the set T
$U \leftarrow [1, \ldots, 1]$
comment: $U[j] = 1$ if the jth movie has **not** been seen
$j^* \leftarrow 1$
comment: j^* is the smallest j such that $U[j] = 1$
while $j^* \leq n$
do
  $b^* \leftarrow b_{j^*}$
  $\ell \leftarrow \ell + 1$
  $T[\ell] \leftarrow b^*$
  for $j \leftarrow j^*$ **to** n
    do if $b^* \geq a_j$
    then $U[j] \leftarrow 0$
  $j \leftarrow j^*$
  $j^* \leftarrow n + 1$
  while $j \leq n$
    do
      if $U[j] = 1$
        then $j^* \leftarrow j$; $j \leftarrow n + 1$
        else $j \leftarrow j + 1$
return (T, ℓ)

Remark: Since we did not *specifically* ask for the algorithm to record the list of days during which the theatre will be visited, we will accept for full marks a solution that does not incorporate the array T. Such an algorithm would only return the value ℓ as its output.

Remark: It is also possible to devise a correct greedy algorithm that sorts the movies in *decreasing* order of a_j and starts from the last movie, proceeding backwards to the first movie.

Complexity Analysis: The complexity of the above implementation is clearly $O(n \log n + n^2) = O(n^2)$.

Induction proof: We can prove correctness of this algorithm by the "greedy stays ahead" strategy. Denote the greedy solution by $\mathcal{G} = \{d_1, \ldots, d_\ell\}$. Let $\mathcal{O} = \{e_1, \ldots, e_m\}$ be any optimal solution. (We use set notation for $\mathcal{G}$ and $\mathcal{O}$ in this proof.) We can assume that $d_1 < \cdots < d_\ell$ and $e_1 < \cdots < e_m$.

Since $\mathcal{O}$ is optimal, we have $m \leq \ell$ because the problem is a minimization problem. We want to prove that $m = \ell$. This will be a proof by contradiction, so we assume that $m < \ell$.

We prove by induction that $d_j \geq e_j$ for $j = 1, 2, \ldots, m$. In the base case, the greedy algorithm always chooses $d_1 = b_1$. $\mathcal{O}$ must have $e_1 \leq b_1$, otherwise it misses the first movie $M_1 = [a_1, b_1]$. Hence $d_1 \geq e_1$.

Now, as an inductive hypothesis, assume that $d_{j-1} \geq e_{j-1}$ for some $j \geq 2$. Suppose that $d_j < e_j$. There is a movie $M_i = [a_i, b_i]$ with $b_i = d_j$. This movie has $a_i > d_{j-1}$ since it is not shown on day d_{j-1}. But $d_{j-1} \geq e_{j-1}$, so this movie is not shown on day e_{j-1}. So we have $e_{j-1} < a_i \leq b_i = d_j < e_j$, so it follows that the movie $M_i = [a_i, b_i]$ is not shown on any day in $\mathcal{O}$. This is a contradiction.

By induction, it follows that $d_j \geq e_j$ for $j = 1, 2, \ldots, m$. Now $\mathcal{G}$ contains a day $d_{m+1} > d_m \geq e_m$. There must be a movie that is not shown on any of the days $d_1, \ldots, d_m$, otherwise $\mathcal{G}$ would not contain an $(m+1)$st day. So there is a movie $M_i = [a_i, b_i]$ with $b_i = d_{m+1}$ and $a_i > d_m$. Since $d_m \geq e_m$, this movie is not shown on any of the days in $\mathcal{O}$, which contradicts the fact that $\mathcal{O}$ is a feasible solution. This contradiction shows that $m = \ell$ and hence $\mathcal{G}$ is optimal.

Slick Proof: Consider the greedy solution $\mathcal{G} = \{d_1, \ldots, d_\ell\}$. Each d_j is the last day that a certain movie $M_i = [a_i, b_i]$ is shown (i.e., $d_j = b_i$). The intervals associated with these ℓ movies must be *disjoint*, because whenever a point d_j is added to $\mathcal{G}$, the corresponding movie starts *later than* d_{j-1}, which is the last day that the previous movie in $\mathcal{G}$ is shown. Therefore it requires at least ℓ days to see the ℓ movies in $\mathcal{G}$, so there cannot exist any feasible solution consisting of fewer than ℓ visits to the theatre.

(b) [*6 marks*] Illustrate the execution of your algorithm step-by-step on the problem instance consisting of the following movies $M_j = [a_j, b_j]$:

$$[7,9], \quad [14,23], \quad [18,32], \quad [32,36], \quad [2,6], \quad [10,19], \quad [15,34], \quad [8,15],$$
$$[22,31], \quad [33,37], \quad [4,7], \quad [7,24], \quad [24,38], \quad [3,10], \quad [8,18], \quad [30,40].$$

Answer: We sort and rename the movies as follows:

i	$[a_i, b_i]$	i	$[a_i, b_i]$	i	$[a_i, b_i]$	i	$[a_1, b_i]$
1	$[2,6]$	2	$[4,7]$	3	$[7,9]$	4	$[3,10]$
5	$[8,15]$	6	$[8,18]$	7	$[10,19]$	8	$[14,23]$
9	$[7,24]$	10	$[22,31]$	11	$[18,32]$	12	$[15,34]$
13	$[32,36]$	14	$[33,37]$	15	$[24,38]$	16	$[30,40]$

Then we construct the optimal solution as follows (we use set notation for T and U):

- $T = \emptyset$, $U = \{1, 2, \ldots, 16\}$.
- We set $j^* = 1$, $b^* = 6$, $T = \{6\}$ and $U = \{3, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16\}$.
- We set $j^* = 3$, $b^* = 9$, $T = \{6, 9\}$ and $U = \{7, 8, 10, 11, 12, 13, 14, 15, 16\}$.
- We set $j^* = 7$, $b^* = 19$, $T = \{6, 9, 19\}$ and $U = \{10, 13, 14, 15, 16\}$.
- We set $j^* = 10$, $b^* = 31$, $T = \{6, 9, 19, 31\}$ and $U = \{13, 14\}$.
- We set $j^* = 13$, $b^* = 36$, $T = \{6, 9, 19, 31, 36\}$ and $U = \emptyset$.
- The optimal solution (found by the greedy algorithm) is $\ell = 5$ and $T = \{6, 9, 19, 31, 36\}$.