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

Assignment 1 Sample Solutions

1. [*10 marks*] .

   (a) [*6 marks*] Give a proof from first principles (not using limits) that $n^3 - 100n + 1000 \in \Theta(n^3)$.

   Answer: We have $n^3 - 100n + 1000 > n^3 - 100n = n(n^2 - 100)$ for all n. Clearly $n^2 - 100 > .5n^2$ if $.5n^2 > 100$, or $n > \sqrt{200} \approx 14.14$. So $n^3 - 100n + 1000 \geq .5n^3 > 0$ for $n \geq 15$. Also, $n^3 - 100n + 1000 < n^3 + 1000$ and $n^3 + 1000 < 2n^3$ if $n^3 > 1000$, i.e., $n > 10$. So we have
   $$0 < .5n^3 < n^3 - 100n + 1000 < 2n^3$$
   for all $n \geq 15$.

   (b) [*4 marks*] Suppose that $f(n)$, $g(n)$ and $h(n)$ are positive-valued functions such that $f(n) \in O(h(n))$ and $g(n) \in O(h(n))$. Prove that $2.72f(n) + 3.14g(n) \in O(h(n))$.

   Answer: There are constants c_0 and n_0 such that $0 \leq f(n) \leq c_0h(n)$ for all $n \geq n_0$. Further, there are constants c_1 and n_1 such that $0 \leq f(n) \leq c_1h(n)$ for all $n \geq n_1$. Then
   $$0 \leq 2.72f(n) + 3.14g(n) \leq (2.72c_0 + 3.14c_1)h(n)$$
   for $n \geq \max\{n_0, n_1\}$. So, if we define $c_2 = 2.72c_0 + 3.14c_1$ and $n_2 = \max\{n_0, n_1\}$, we have
   $$0 \leq 2.72f(n) + 3.14g(n) \leq c_2h(n)$$
   for all $n \geq n_2$.

2. [*12 marks*] For each pair of functions $f(n)$ and $g(n)$, fill in the correct asymptotic notation among Θ, o, and ω in the statement $f(n) \in \sqcup\, (g(n))$. Formal proofs are not necessary, but provide brief justifications for all of your answers. (The default base in logarithms is 2.)

   (a) $f(n) = \sum_{i=1}^{n-1}(i+1)/i^2$ vs. $g(n) = \log(n^{100})$

   Answer: We have $f(n) = \sum_{i=1}^{n-1}(1/i + 1/i^2)$. $\sum_{i=1}^{n-1} 1/i \in \Theta(\log n)$ and $\sum_{i=1}^{n-1} 1/i^2 \in \Theta(1)$ because $\sum_{i=1}^{\infty} 1/i^2$ is finite. Thus $f(n) \in \Theta(\log n)$. We have $g(n) = 100 \log n \in \Theta(\log n)$ so $f(n) \in \Theta(g(n))$.

   (b) $f(n) = n^{3/2}$ vs. $g(n) = (n+1)^9/(n^3-1)^2$.

   Answer: We have
   $$g(n) = \frac{n^9 + \text{ lower order terms}}{n^6 + \text{ lower order terms}} \in \Theta\left(\frac{n^9}{n^6}\right) = \Theta(n^3).$$
   Therefore $f(n) \in o(g(n))$.

(c) $f(n) = (32768)^{n/5}$ vs. $g(n) = (6561)^{n/4}$

Answer: We have $f(n) = (32768)^{n/5} = (32768^{1/5})^n = 8^n$ and $g(n) = (6561)^{n/4} = (6561^{1/4})^n = 9^n$. We now consider

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{8^n}{9^n} = \lim_{n\to\infty} \left(\frac{8}{9}\right)^n = 0.$$

Thus $f(n) \in o(g(n))$.

(d) $f(n) = (\log n)^{\log n}$ vs. $g(n) = n^{\log\log n}$. [Hint: take logarithms.]

Answer: We have $\log f(n) = \log n \log\log n$ and $\log g(n) = \log\log n \log n$, so $\log f(n) = \log g(n)$. Therefore $f(n) = g(n)$ and $f(n) \in \Theta(g(n))$.

3. [*10 marks*] Analyze the following pseudocode and give a tight Θ bound on the running time as a function of n. Carefully show your work.

(a) [*5 marks*]

```
s = 0
for i = 1 to n do {
    j = i
    while j ≤ n do {
        j = j + i
        s = s + j
    }
}
```

Answer: For a given value of i, j takes on the values $i, 2i, 3i, \ldots$ in the **while** loop. Therefore there are n/i iterations of the **while** loop that are executed in the ith iteration of the **for** loop. it follows that the ith iteration of the **for** loop takes time $\Theta(n/i)$. Therefore the total time is

$$\sum_{i=1}^{n} \Theta(n/i) = \Theta\left(\sum_{i=1}^{n} n/i\right) = \Theta\left(n \sum_{i=1}^{n} 1/i\right) = \Theta(n \log n).$$

(b) [*5 marks*]

```
k = 1
s = 0
for i = 1 to n do {
    for j = 1 to 2k do
        s = s + j
    k = 2k
}
```

Answer: The inner **for** loop takes time $\Theta(k)$. The value of k at the beginning of the ith iteration of the outer **for** loop is 2^{i-1}. Therefore the total time is

$$\sum_{i=1}^{n} \Theta(2^{i-1}) = \Theta\left(\sum_{i=1}^{n} 2^{i-1}\right) = \Theta(2^n - 1) = \Theta(2^n).$$

4. [*6 marks*] Consider the following problem named M3SUM: Given an array of n **positive, distinct** integers, $S[1], \ldots, S[n]$, determine if there exist three array elements $S[i], S[j]$ and $S[k]$ such that

$$S[i] + S[j] = S[k]$$

(where $1 \le i, j, k \le n$ and i, j, k are all distinct). Define $T[\ell] = 4S[\ell] - 1$ for $1 \le \ell \le n$ and define $T[\ell + n] = -4S[\ell] + 2$ for $1 \le \ell \le n$. Show that solving 3SUM on the array T (of length $2n$) will solve M3SUM on the array S (so this is a reduction from M3SUM to 3SUM). [Important: you need to show that there is a solution for M3SUM for the instance S **if and only if** there is a solution for 3SUM for the instance T.]

Answer: Suppose that $S[i] + S[j] = S[k]$, where $1 \le i, j, k \le n$ and i, j, k are all distinct. Then

$$T[i] + T[j] + T[n + k] = 4S[i] - 1 + 4S[j] - 1 - 4S[k] + 2 = 4(S[i] + S[j] - S[k]) = 0.$$

Conversely, suppose that $T[i] + T[j] + T[k] = 0$, where the array T is constructed from S as described above. We consider several cases:

(a) If $i, j, k \le n$, then $T[i] + T[j] + T[k] \equiv -1 - 1 - 1 \equiv -3 \bmod 4$, so $T[i] + T[j] + T[k] \ne 0$, a contradiction.

(b) If precisely one of i, j, k is $\le n$, then $T[i] + T[j] + T[k] \equiv -1 + 2 + 2 \equiv -1 \bmod 4$, so $T[i] + T[j] + T[k] \ne 0$, a contradiction.

(c) If $i, j, k > n$, then $T[i] + T[j] + T[k] \equiv 2 + 2 + 2 \equiv 2 \bmod 4$, so $T[i] + T[j] + T[k] \ne 0$, a contradiction.

The only case remaining is that precisely two of i, j, k are $\le n$. WLOG assume $i, j \le n$ and $k > n$. Then

$$0 = T[i] + T[j] + T[k] = 4S[i] - 1 + 4S[j] - 1 - 4S[k - n] + 2 = 4(S[i] + S[j] - S[k - n]).$$

Therefore $S[i] + S[j] = S[k - n]$. We have $i \ne j$ because i, j, k are distinct. Is it possible that $i = k - n$? This would imply that $S[j] = 0$ which is not possible since S consists only of positive integers. Similarly, $j \ne k - n$. Therefore we have a solution to M3SUM.

5. [*10 marks*] Suppose Alice spends a_i dollars on the ith day and Bob spends b_i dollars on the ith day, for $1 \le i \le n$. We want to determine whether there exists some set of t consecutive days during which total amount spent by Alice is exactly the same as the total amount spent by Bob in some (possibly different) set of t consecutive days. That is, we want to determine if there exist i, j, t (with $0 \le i, j \le n - t$ and $1 \le t \le n$) such that

$$a_{i+1} + a_{i+2} + \cdots + a_{i+t} = b_{j+1} + b_{j+2} + \cdots + b_{j+t}.$$

For example, for the inputs $10, 21, 11, 12, 19, 15$ and $12, 9, 2, 31, 21, 8$, the answer is “yes” because $11 + 12 + 19 = 9 + 2 + 31$.

(a) [*5 marks*] First design and analyze an algorithm that solves the problem in $\Theta(n^3)$ time by “brute force”.

Answer:

Algorithm: *EqualSpending*$(a_1, \ldots, a_n, b_1, \ldots, b_n)$
for $t \leftarrow 1$ **to** n

$$\textbf{do} \begin{cases} \textbf{for } i \leftarrow 0 \textbf{ to } n-t \\ \quad \textbf{do} \begin{cases} S[i] \leftarrow 0 \\ T[i] \leftarrow 0 \\ \textbf{for } j \leftarrow i+1 \textbf{ to } i+t \\ \quad \textbf{do} \begin{cases} S[i] \leftarrow S[i] + a_j \\ T[i] \leftarrow T[i] + b_j \end{cases} \end{cases} \\ \textbf{for } i \leftarrow 0 \textbf{ to } n-t \\ \quad \textbf{do} \begin{cases} \textbf{for } j \leftarrow 0 \textbf{ to } n-t \\ \quad \textbf{do} \begin{cases} \textbf{if } S[i] = T[j] \\ \quad \textbf{then return } (i, j, t) \end{cases} \end{cases} \end{cases}$$

For a given value of t, the two **for** loops (on i) have complexity $\Theta(t(n-t))$ and $\Theta((n-t)^2)$, respectively. Since $1 \leq t \leq n$, these are both $O(n^2)$ and the algorithm has complexity $O(n^3)$.

For the Ω-bound, we observe that

$$\sum_{t=1}^{n} (n-t)^2 \geq \sum_{t=1}^{n/2} (n-t)^2 \geq \sum_{t=1}^{n/2} (n/2)^2 = n^3/8.$$

Therefore the algorithm has complexity $\Omega(n^3)$.

Remark: The first **for** loop on i can be made to run in time $O(n)$ by a simple optimization. This is used in part (b); however, this optimization is not required for this part of the question.

(b) [*5 marks*] Design and analyze a better algorithm that solves the problem in $\Theta(n^2 \log n)$ time. [Hint: use sorting.]

Answer: We need to make two improvements.

First, we compute all the the sums $S[0], \ldots, S[n-t]$ and $T[0], \ldots, T[n-t]$ in $\Theta(n)$ time. The idea is that any $S[i]$ with $i > 0$ can be computed from $S[i-1]$ in $O(1)$ time by using the formula $S[i] = S[i-1] + a_{i+t} - a_i$, and $S[0]$ can be computed in time $\Theta(t)$. A similar optimization can be done for the computation of $T[0], \ldots, T[n-t]$.

Second, after computing the sums $S[0], \ldots, S[n-t]$ and $T[0], \ldots, T[n-t]$, we sort these two lists (separately) using a $\Theta((n-t)\log(n-t))$ sorting algorithm such as *MergeSort*. Then we can search for $S[i] = T[j]$ with a single pass through each of these two lists.

Algorithm: *FasterEqualSpending*$(a_1, \dots, a_n, b_1, \dots, b_n)$

$$
\begin{array}{l}
\textbf{for } t \leftarrow 1 \textbf{ to } n \\
\quad \textbf{do} \begin{cases}
S[0] \leftarrow 0 \\
T[0] \leftarrow 0 \\
\textbf{for } i \leftarrow 1 \textbf{ to } t \\
\quad \textbf{do} \begin{cases} S[0] \leftarrow S[0] + a_i \\ T[0] \leftarrow T[0] + b_i \end{cases} \\
\textbf{for } i \leftarrow 1 \textbf{ to } n - t \\
\quad \textbf{do} \begin{cases} S[i] \leftarrow S[i-1] + a_{i+t} - a_i \\ T[i] \leftarrow T[i-1] + b_{i+t} - b_i \end{cases} \\
\mathit{MergeSort}(S[0], \dots, S[n-t]) \\
\mathit{MergeSort}(T[0], \dots, T[n-t]) \\
i \leftarrow 0 \\
j \leftarrow 0 \\
\textbf{while } i \le n - t \textbf{ and } j \le n - t \\
\quad \textbf{do} \begin{cases} \textbf{if } A[i] = B[j] \\ \quad \textbf{then return } (i, j, t) \\ \quad \textbf{else if } A[i] < B[j] \quad \textbf{then } i \leftarrow i + 1 \\ \quad \textbf{else } j \leftarrow j + 1 \end{cases}
\end{cases}
\end{array}
$$

Now, the complexity of the tth iteration of the outer **for** loop is clearly

$$\Theta(n) + \Theta((n-t)\log(n-t)) + \Theta(n-t) = \Theta(n + (n-t)\log(n-t)).$$

Since $1 \le t \le n$, this is $O(n \log n)$ and the entire algorithm runs in time $O(n^2 \log n)$. To get the corresponding Ω-bound, we can observe that

$$\sum_{t=1}^{n}(n-t)\log(n-t) \ge \sum_{t=1}^{n/2}(n-t)\log(n-t) \ge \sum_{t=1}^{n/2}(n/2)\log(n/2) = (n^2/4)(\log n - 1).$$

6. [*21 marks*] Suppose we are given an array of n integers, $A[1], \dots, A[n]$, and a positive integer k. We want to find the maximum value of $A[i] + A[j]$ subject to the condition that $1 \le i < j \le \min\{i + k, n\}$. That is, we want the maximum sum of two array elements that are at most k apart in the array. For example, for the inputs $10, 2, 0, 8, 1, 7, 1, 0, 11$ and $k = 2$, the maximum sum is $A[4] + A[6] = 8 + 7 = 15$ (the two array elements are $A[4]$ and $A[6]$, which are two apart).

   (a) [*4 marks*] Design and analyze a simple "brute-force" algorithm for this problem that runs in $O(kn)$ time.

Algorithm: *BruteForce*$(A[1], \ldots, A[n])$

$$
\begin{array}{l}
\mathit{Max} \leftarrow 0\\
\textbf{for } i \leftarrow 1 \textbf{ to } n-1\\
\quad \textbf{do} \begin{cases} L \leftarrow \min\{i+k, n\}\\ \textbf{for } j \leftarrow i+1 \textbf{ to } L\\ \quad \textbf{do} \begin{cases} \textbf{if } A[i]+A[j] > \mathit{Max}\\ \quad \textbf{then } \mathit{Max} \leftarrow A[i]+A[j] \end{cases} \end{cases}\\
\textbf{return } (\mathit{Max})
\end{array}
$$

Note that the inner **for** loop has at most k iterations and the outer **for** loop has n iterations. Thus the complexity is $O(kn)$ because $O(1)$ work is done in each iteration.

(b) [*8 marks*] Design a divide-and-conquer algorithm for this problem in which you split the array into two equal pieces, where the "combine" operation of the algorithm runs in time $O(k)$.

Answer: We recursively compute the maximum value within the subarray $A[1], \ldots, A[n/2]$ and the maximum value within the subarray $A[n/2+1], \ldots, A[n]$. Then we also need to compute the maximum of $A[i] + A[j]$ subject to the conditions that $1 \le i \le n/2$, $n/2 + 1 \le j \le \min\{n, i + k\}$. Clearly this requires that $i \ge n/2 - k + 1$. We need to consider the following sums:

$$
\begin{array}{l}
A[n/2-k+1] + A[n/2+1]\\
A[n/2-k+2] + A[n/2+1], A[n/2-k+2] + A[n/2+2]\\
A[n/2-k+3] + A[n/2+1], A[n/2-k+3] + A[n/2+2], A[n/2-k+3] + A[n/2+3]\\
\ldots\\
A[n/2] + A[n/2+1], A[n/2] + A[n/2+2], \ldots, A[n/2] + A[n/2+k]
\end{array}
$$

For $1 \le j \le k$, let

$$R_j = \max\{A[n/2+1], \ldots, A[n/2+j]\}.$$

Then the maximum value in the jth row of the above table is $A[n/2 - k + 3] + R_j$ and the value we are trying to compute is

$$M = \max\{A[n/2-k+j] + R_j : 1 \le j \le k\}.$$

Clearly we can compute M in time $O(k)$ if we can compute each R_j from R_{j-1} in $O(1)$ time. But this is easy, since $R_j = \max\{R_{j-1}, A[n/2 + j]\}$ for $2 \le j \le k$. The initial value $R_1 = A[n/2 + 1]$.

The following algorithm basically follows this approach. In the updating step for R, we have $i = n/2 - k + j$, so $n/2 + j = i + k$. We implicitly assume that $n = 2^t k$ for some integer k, so we can then use the solution for $n = k$ from part (c) as a base case.

Algorithm: *DCMaxSum*$(A[1], \ldots, A[n])$
if $n = k$
then treat this as a base case

$$\textbf{else} \begin{cases} S_1 \leftarrow \mathit{DCMaxSum}(A[1], \ldots, A[n/2]) \\ S_2 \leftarrow \mathit{DCMaxSum}(A[n/2+1], \ldots, A[m]) \\ S_3 \leftarrow -\infty \\ R \leftarrow -\infty \\ \textbf{for } i \leftarrow n/2 - k + 1 \textbf{ to } n/2 \\ \quad \textbf{do} \begin{cases} R \leftarrow \max\{R, A[i+k]\} \\ S_3 \leftarrow \max\{S_3, A[i] + R\} \end{cases} \\ \textbf{return } (\max\{S_2, S_2, S_3\}) \end{cases}$$

However, the above algorithm contains an inefficiency in that it requires copying subarrays. This will contribute a $\Theta(n)$ term to the recurrence relation, which we want to avoid. The solution is to treat the array as a global object and only pass indices to *DCMaxSum* (similar to what is done in a binary search).

We obtain the following modified algorithm:

Algorithm: *DCMaxSum*(lo, hi)
global $[A[1], \ldots, A[n]]$
if $hi - lo = k - 1$
then treat this as a base case

$$\textbf{else} \begin{cases} mid \leftarrow \lfloor (lo + hi)/2 \rfloor \\ S_1 \leftarrow \mathit{DCMaxSum}(lo, mid) \\ S_2 \leftarrow \mathit{DCMaxSum}(mid + 1, hi) \\ S_3 \leftarrow -\infty \\ R \leftarrow -\infty \\ \textbf{for } i \leftarrow mid - k + 1 \textbf{ to } mid \\ \quad \textbf{do} \begin{cases} R \leftarrow \max\{R, A[i+k]\} \\ S_3 \leftarrow \max\{S_3, A[i] + R\} \end{cases} \\ \textbf{return } (\max\{S_2, S_2, S_3\}) \end{cases}$$

(c) [*4 marks*] Show that this problem can be solved directly (not recursively) for an array of length $n = k$ in $O(k)$ time (this will be used as a base case for the divide-and-conquer algorithm).

Answer: It suffices to iterate through the k elements of A, keeping track of the largest and second largest elements. At the end, compute the sum of these two elements.

The following algorithm passes indices of the array A as parameters.

Algorithm: *MaxSumBaseCase*(lo, hi)

comment: assume $1 \le hi - lo \le k-1$

$$
\begin{array}{l}
\textbf{if } A[lo] > A[lo+1] \\
\quad \textbf{then } \begin{cases} M_1 \leftarrow A[lo] \\ M_2 \leftarrow A[lo+1] \end{cases} \\
\quad \textbf{else } \begin{cases} M_1 \leftarrow A[lo+1] \\ M_2 \leftarrow A[lo] \end{cases} \\
\textbf{for } i \leftarrow lo \textbf{ to } hi \\
\quad \textbf{do } \left\{ \begin{array}{l} \textbf{if } A[i] \ge M_1 \\ \quad \textbf{then } \begin{cases} M_2 \leftarrow M_1 \\ M_1 \leftarrow A[i] \end{cases} \\ \quad \textbf{else if } A[i] \ge M_2 \\ \quad \textbf{then } M_2 \leftarrow A[i] \end{array} \right. \\
\textbf{return } (M_1 + M_2)
\end{array}
$$

The complexity is obviously $\Theta(k)$.

(d) [*5 marks*] Using $n = k$ as a base case, the running time $T(n)$ for the divide-and-conquer algorithm satisfies the following recurrence which involves both k and n:

$$T(n) = \begin{cases} 2T(n/2) + O(k) & \text{if } n > k \\ O(k) & \text{if } n = k. \end{cases}$$

Show that the solution to this recurrence is $T(n) \in O(n)$ if $n = k2^t$ for some integer t. [Hint: this can be done using either the recursion tree method or guess-and-check.]

Answer: We use guess-and-check. Suppose we write $T(k) = ck$ for some positive constant c. Then $T(2k) = 2ck + dk$ for some constant d. Computing further values, we have $T(4k) = 4ck + 3dk$ and $T(8k) = 8ck + 7dk$. From these values, we conjecture that $T(2^t k) = 2^t ck + (2^t - 1)dk$, which is easily proven by induction on t.

The base case ($t = 0$) is correct because $2^t ck + (2^t - 1)dk = ck = T(k)$ when $t = 0$.

As an induction assumption, suppose the formula holds for $t = s - 1$. Then

$$\begin{aligned} T(k2^s) &= 2T(k2^{s-1}) + dk \\ &= 2(2^{s-1}ck + (2^{s-1} - 1)dk) + dk \\ &= 2^s ck + 2^s dk - 2dk + dk \\ &= 2^s ck + (2^s - 1)dk. \end{aligned}$$

Thus the formula holds for all $s \ge 0$.

Writing this result in terms of n, we have $T(n) = cn + dn - dk$. Since $k \le n$, we have $T(n) \in \Theta(n)$ because $cn \le T(n) \le (c + d)n$.

7. [*6 marks*] Give a tight asymptotic (i.e., Θ) bound for the solution to the following recurrence by using the recursion-tree method (you may assume that n is a power of 4). Show your work.

$$T(n) = \begin{cases} 3\,T(n/4) + \sqrt{n} & \text{if } n > 1 \\ 5 & \text{if } n \le 1 \end{cases}$$

Let $n = 4^j$. We tabulate the number of nodes at each level of the tree, and their values.

level	# nodes	value at each node	value of the level
j	1	$n^{1/2}$	$n^{1/2}$
$j-1$	3	$(n/4)^{1/2}$	$3(n/4)^{1/2}$
$j-2$	3^2	$(n/4^2)^{1/2}$	$3^2(n/4^2)^{1/2}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
1	3^{j-1}	$(n/4^{j-1})^{1/2}$	$3^{j-1}(n/4^{j-1})^{1/2}$
0	3^j	5	5×3^j

Summing the values at all levels of the recursion tree, we have that

$$\begin{aligned}
T(n) &= 5 \times 3^j + n^{1/2} \sum_{i=0}^{j-1} \left(\frac{3}{4^{1/2}} \right)^i \\
&= 5 \times 3^j + n^{1/2} \sum_{i=0}^{j-1} \left(\frac{3}{2} \right)^i . \\
&= 5n^{\log_4 3} + n^{1/2} \left(\frac{(3/2)^j - 1}{3/2 - 1} \right) \\
&= 5n^{\log_4 3} + 2n^{1/2} \left(\frac{3^j - 2^j}{2^j} \right) \\
&= 5n^{\log_4 3} + 2n^{1/2} \left(\frac{n^{\log_4 3} - n^{1/2}}{n^{1/2}} \right) \\
&= 5n^{\log_4 3} + 2 \left(n^{\log_4 3} - n^{1/2} \right) \\
&= 7n^{\log_4 3} - 2n^{1/2},
\end{aligned}$$

when $n = 4^j$. Note that we use the facts that $3^j = n^{\log_4 3}$ and $2^j = n^{\log_4 2} = n^{1/2}$ in the above simplifications.

Since $\log_4 3 \approx 0.79248 > 1/2$, we have that $T(n) \in \Theta(n^{\log_4 3})$.