Problem Set 1 Solutions

Problem 1-1. Asymptotic Notation
For each of the following relationships, find two nonnegative functions $f(n)$ and $g(n)$ that satisfy it. If no such functions exist, write “NONE” and briefly justify your answer.

(a) $f(n) = O(g(n))$ and $g(n) = O(f(n))$.

**Solution:** $f(n) = g(n) = n$.

(b) $f(n) = o(g(n))$ and $g(n) = o(f(n))$.

**Solution:** NONE: If $f(n) = o(g(n))$, then there exists an $n_0$ such that, for all $n > n_0$, $f(n) < \frac{1}{2}g(n)$. Similarly, $g(n) = o(f(n))$ implies that there exists an $n_1$ such that, for all $n > n_1$, $g(n) < \frac{1}{2}f(n)$. Thus, for all $n > \max\{n_0, n_1\}$, we have $f(n) < \frac{1}{4}f(n)$, which cannot hold if $f(n)$ is nonnegative.

(c) $f(n) = \omega(f(n))$

**Solution:** NONE: $f(n) = \omega(f(n))$ implies that there exists an $n_0$ such that, for all $n > n_0$, $f(n) > f(n)$. But this is clearly impossible.

(d) $f(n) = \Omega(g(n))$ and $g(n) = \omega(f(n) + g(n))$.

**Solution:** NONE: If we add $g(n)$ to both sides of $f(n) = \Omega(g(n))$, we obtain that $f(n) + g(n) = \Omega(g(n))$. Substituting this relation into the right-hand side of $g(n) = \omega(f(n) + g(n))$, we obtain that $g(n) = \omega(g(n))$, which is impossible by part (c).

(e) $f(n) = \Theta(1) = o(f(n))$

**Solution:** $f(n) = 1, g(n) = 1/n$.

(f) $f(n) = O(1) = o(f(n) + g(n))$.

**Solution:** $f(n) = 1, g(n) = 1/n$. 

Problem 1-2. Recurrences

Give asymptotic upper and lower bounds for $T(n)$ in each of the following recurrences. Assume that $T(n)$ is a nonnegative constant for $n \leq 10$. Make your bounds as tight as possible, and justify your answers.

(a) $T(n) = 3T(n/3) + n$

Solution: Using the notation of the Master Theorem, we have $a = 3$, $b = 3$, so $n^{\log_b a} = n$, and $f(n) = n$. Thus, by Case 2 of the Master Theorem, $T(n) = \Theta(n \lg n)$.

(b) $T(n) = 11T(n/7) + n^3$

Solution: Using the notation of the Master Theorem, we have $a = 11$, $b = 7$, so $n^{\log_b a} \approx n^{1.23}$, and $f(n) = n^3$. The regularity condition is easy to verify: $af(n/b) = 11(n/7)^3 = (11/343)n^3 \leq \frac{1}{2}n^3$. Thus, by Case 3 of the Master Theorem, $T(n) = \Theta(n^3)$.

(c) $T(n) = 16T(n/2) + \binom{n}{3} \lg^4 n$

Solution: Using the notation of the Master Theorem, we have $a = 16$, $b = 2$, so $n^{\log_b a} = n^4$, and $f(n) = \binom{n}{3} \lg^4 n$. Now $f(n) = O(n^{3+\epsilon})$ for any $0 < \epsilon < 1$. Thus, by Case 1 of the Master Theorem, $T(n) = \Theta(n^4)$.

(d) $T(n) = \frac{4}{3}T(\frac{3}{4}n) + \frac{4}{3}n$

Solution: Using the notation of the Master Theorem, we have $a = \frac{4}{3}$, $b = \frac{4}{3}$, so $n^{\log_b a} = n$, and $f(n) = \frac{1}{\frac{3}{4}}n$. Clearly, $f(n) = \Theta(n)$. Thus, by Case 2 of the Master Theorem, $T(n) = \Theta(n \lg n)$.

(e) $T(n) = 2T(n/2) + \lg(n!)$

Solution: Using the notation of the Master Theorem, we have $a = 2$, $b = 2$, so $n^{\log_b a} = n$, and $f(n) = \lg(n!)$. By Sterling’s Formula, $f(n) = \Theta(n \lg n)$. Thus, by the generalized Case 2 of the Master Theorem covered in lecture, $T(n) = \Theta(n \lg^2 n)$.

(f) $T(n) = T(n - 10) + n$

Solution: $T(n) = n + T(n-10) = n + (n-10) + T(n-20) = \cdots = \sum_{i=0}^{n/10} (n-10i) = n(n/10) - 10 \sum_{i=0}^{n/10} i = n^2/10 - n(n/10 + 1)/2 = \Theta(n^2)$. 

(g) \( T(n) = 2T(n/4) + 6.046 \sqrt{n} \)

**Solution:** Using the notation of the Master Theorem, we have \( a = 2, b = 4 \), so \( n^{\log_b a} = n^{1/2} = \sqrt{n} \), and \( f(n) = \sqrt{n} \). Thus, by Case 2 of the Master Theorem, \( T(n) = \Theta(\sqrt{n} \log n) \).

(h) \( T(n) = T(n/3) + T(n/4) + n \)

**Solution:** Draw the recursion tree. Level \( i \) contributes at most \( (1/3 + 1/4)^i - 1 \) \( n \) to the total value of \( T(n) \). This bound suggests an \( \sum_{i=0}^{\infty} (1/3 + 1/4)^i - 1 \) \( n = \Theta(n) \) guess for the value of \( T(n) \). In fact, \( T(n) = \Omega(n) \) immediately from the recursion. Assume that \( T(n) \leq c n \) for an appropriate constant \( c \) and for all \( n \geq n_0 \). We have that \( T(n) \leq c(n/3) + c(n/4) + n = (7c/12 + 1) n \leq cn \) for \( c \geq 12/5 \). Thus \( T(n) = \Theta(n) \).

Alternatively, we can observe that \( T(n) \geq n \) and that \( T(n) \leq 2T(n/3) + n \). The latter recursion solves by the Master Method to \( T(n) = O(n) \).

(i) \( T(n) = 3T(n^{1/3}) + \lg \lg n \)

**Solution:** After the substitution \( n = 2^k \), we obtain the recurrence \( T(2^k) = 3T(2^{k/3}) + \lg k \). Set \( S(k) = T(2^k) \). Then we obtain that \( S(k) = 3S(k/3) + \lg k \). By Case 1 of the Master Theorem, \( S(k) = \Theta(k) \). Therefore \( T(n) = \Theta(\lg n) \).

**Problem 1-3. Lightest Segment Problem**

Given an input array \( A[1..n] \) of numbers, a **segment** of \( A \) is a consecutive interval of one or more elements from \( A \) of the form \( A[i..j] \) where indices \( i \) and \( j \) satisfy \( 1 \leq i \leq j \leq n \). The **weight** of the segment \( A[i..j] \) is defined to be \( \sum_{k=i}^{j} A[k] \). The **absolute weight** of \( A[i..j] \) is \( |\sum_{k=i}^{j} A[k]| \). The absolute weight is always nonnegative. In the **lightest segment problem**, the goal is find the minimum absolute weight of a segment in the given array \( A[1..n] \).

(a) Give a simple \( O(n^3) \)-time algorithm for the lightest segment problem.

**Solution:** A simple algorithm considers each interval \( A[i..j] \) in turn, computes the absolute sum directly, and returns the best interval found over this brute-force search:
Problem Set 1 Solutions

1. best ← ∞
2. for i ← 1 to n
3.   do for j ← i to n
4.     do w ← 0
5.     for k ← i to j
6.       do w ← w + A[k]
7.     if |w| < best
8.       then best ← |w|
9. return best

Each loop has a range of at most n, so the running time of the algorithm is $O(n^3)$.

In the rest of this problem, we develop a more efficient solution using the divide-and-conquer paradigm.

(b) Give an $O(\lg n)$-time algorithm that, given a sorted array $B$ of $n$ numbers and given an integer $x$, determines the minimum value attained by $|B[i] + x|$ where $i$ varies between 1 and $n$.

Solution: Binary search for $-x$ in $B$ to find an index $i$ such that $B[i] \leq -x \leq B[i+1]$. The closest element of $B$ to $-x$ is either $B[i]$ and $B[i+1]$, and we can determine which it is in $O(1)$ time. Minimizing the distance between $B[j]$ and $-x$ is equivalent to minimizing $|B[j] - (-x)| = |B[j] + x|$, solving the problem. The running time is dominated by the binary search, which takes $O(\lg n)$ time.

Alternatively, we can design an explicit divide-and-conquer algorithm as follows. Let $\text{sgn}(y)$ denote the sign of $y$: $\text{sgn}(y) = -1$ for $y < 0$, $\text{sgn}(y) = 1$ for $y > 0$, and $\text{sgn}(0) = 0$. Let $1 \leq m \leq n - 1$. The fact that $B$ is a sorted array implies that

- If $\text{sgn}(B[m] + x) = \text{sgn}(B[m + 1] + x) = -1$ then the minimum value attained by $|B[i] + x|$ is attained for some $m + 1 \leq i \leq n$;
- If $\text{sgn}(B[m] + x) = \text{sgn}(B[m + 1] + x) = 1$ then the minimum value attained by $|B[i] + x|$ is attained for some $1 \leq i \leq m$;
- If $\text{sgn}(B[m] + x) \neq \text{sgn}(B[m + 1] + x)$ then the minimum value attained by $|B[i] + x|$ is either $|B[m] + x|$ or $|B[m + 1] + x|$.
- If $\text{sgn}(B[m] + x) = 0$ then the minimum value attained by $|B[i] + x|$ is zero.

These observations suggest the following simple divide-and-conquer algorithm; note the resemblance to binary search.
The running time $T(n)$ satisfies the recurrence $T(n) = T(n/2) + O(1)$, which solves to $O(\lg n)$ as with binary search.

(c) Give an $O(n \lg n)$-time algorithm that, given two arrays $B_1[1..n]$ and $B_2[1..n]$ of numbers, determines the minimum value attained by $|B_1[i] + B_2[j]|$, where $i$ and $j$ vary between 1 and $n$. **Hint:** Use the algorithm from part (b).

**Solution:** Sort the array $B_1$ using an $O(n \lg n)$-time algorithm. For each $1 \leq j \leq n$, run the algorithm from part (b) to determine the minimum value attained by $|B_1[i] + B_2[j]|$ over all possible choices of $i$. Return the best such minimum value found over all choices of $j$. Because there are only $n$ choices for $j$, and it takes $O(\lg n)$ time to process each of them, the total running time is $O(n \lg n)$.

We distinguish three different kinds of segments $A[i..j]$. Define $m$ to be $\lfloor n/2 \rfloor$. If $i, j \leq m$, we call the segment **left**. If $i, j > m$, we call the segment **right**. Otherwise, $i \leq m$ and $j > m$, and we call the segment **middle**.

(d) Give an $O(n \lg n)$-time algorithm that, given an array $A[1..n]$ of numbers, determines the minimum absolute weight of a middle segment $A[i..j]$ (where $1 \leq i \leq m$ and $m < j \leq n$).

**Hint:** The weight of a middle segment $A[i..j]$ can be written as the sum of the weights of two subsegments, $A[i..m]$ and $A[m+1..j]$. There are only $O(n)$ such subsegments; avoid recomputing their weights.

**Solution:** In $O(n)$ time, we can compute the two arrays $B_1[1..m]$ and $B_2[1..n - m]$ described in the hint:
1. \( B_1[m] \leftarrow A[m] \)
2. \textbf{for} \( k \leftarrow m - 1 \) \textbf{downto} 1
3. \hspace{1em} \textbf{do} \( B_1[k] \leftarrow B_1[k + 1] + A[k] \)
4. \( B_2[1] \leftarrow A[m + 1] \)
5. \textbf{for} \( k \leftarrow 2 \) \textbf{to} \( n - m \)
6. \hspace{1em} \textbf{do} \( B_2[k] \leftarrow B_2[k - 1] + A[m + k] \)

Then we run the \( O(n \log n) \)-time algorithm from part (c) on \( B_1 \) and \( B_2 \) to minimize the value attained by \( |B_1[i] + B_2[j]| = |A[i \ldots m] + A[m + 1 \ldots (m + j)]| = |A[i \ldots (m + j)]| \) where \( 1 \leq i \leq m \) and \( 1 \leq j \leq n - m \). This is equivalent to minimizing \( |A[i \ldots j]| \) where \( 1 \leq i \leq m \) and \( m + 1 \leq j \leq n \).

\( e \) Give an \( O(n \log^2 n) \)-time divide-and-conquer algorithm for the lightest segment problem.

\textbf{Solution:} Again let \( m = \lfloor n/2 \rfloor \). Divide the array \( A \) into two parts \( A[1 \ldots m] \) and \( A[m+1 \ldots n] \). The lightest segment is either left (contained in the first part), right (contained in the right part), or middle. The algorithm from part (d) finds the minimum absolute weight of a middle segment. We can find the minimum absolute weight of a left segment by recursion, and similarly for the right segments. The running time \( T(n) \) satisfies the recurrence \( T(n) = 2T(n/2) + O(n \log n) \). By Case 2 of the Master Theorem, \( T(n) = \Theta(n \log^2 n) \).

\( f \) \textbf{Bonus Part.} This part is optional. If you solve this part, you do not need to write up your solutions to the other parts of this problem.

Give an \( O(n \log n) \)-time algorithm for the lightest segment problem.

\textbf{Solution:} Compute, for \( i = 0 \) to \( n \), the sum \( S[i] \) of the first \( i \) elements of the array. This can be done in time \( O(n) \) since \( S[0] = 0 \) and \( S[i + 1] = S[i] + A[i + 1] \) for \( 0 \leq i < n \). The weight of the segment \( A[i \ldots j] \) is equal to \( S[j] - S[i - 1] \); thus, all is left to do is find the pair of sums with the minimum difference. To do so, find a sorted permutation of the \( n + 1 \) sums in time \( O(n \log n) \). The minimum difference between two sums must be between two sums that are adjacent in the sorted array (since if \( S[h] \geq S[j] \geq S[i] \) then \( S[h] - S[i] \geq S[j] - S[i] \) and can thus be found in time \( O(n) \). The total running time is \( O(n) + O(n \log n) + O(n) = O(n \log n) \).