Problem Set 1 Solutions

Problem 1-1. Asymptotic Notation

For each of the following statements, decide whether it is **always true**, **never true**, or **sometimes true** for asymptotically nonnegative functions f and g. If it is **always true** or **never true**, explain why. If it is **sometimes true**, give one example for which it is true, and one for which it is false.

(a) $f(n) = O(f(n)^2)$

Solution: Sometimes true: For $f(n) = n$ it is true, while for $f(n) = 1/n$ it is not true. (The statement is always true for $f(n) = \Omega(1)$, and hence for most functions with which we will be working in this course, and in particular all time and space complexity functions).

(b) $f(n) + g(n) = \Theta\left(\max\left(f(n), g(n)\right)\right)$

Solution: Always true: $\max(f(n), g(n)) \leq f(n) + g(n) \leq 2\max(f(n), g(n))$.

(c) $f(n) + O(f(n)) = \Theta(f(n))$

Solution: Always true: Consider $f(n) + g(n)$ where $g(n) = O(f(n))$ and let c be a constant such that $0 \leq g(n) < cf(n)$ for large enough n. Then $f(n) \leq f(n) + g(n) \leq (1 + c)f(n)$ for large enough n.

(d) $f(n) = \Omega(g(n))$ and $f(n) = o(g(n))$ (note the little-o)

Solution: Never true: If $f(n) = \Omega(g(n))$ then there exists positive constant c_Ω and n_Ω such that for all $n > n_\Omega$, $cg(n) \leq f(n)$. But if $f(n) = o(g(n))$, then for any positive constant c, there exists $n_o(c)$ such that for all $n > n_o(c)$, $f(n) < cg(n)$. If $f(n) = \Omega(g(n))$ and $f(n) = o(g(n))$, we would have that for $n > \max(n_\Omega, n_o(c_\Omega))$ it should be that $f(n) < c_\Omega g(n) \leq f(n)$ which cannot be.

(e) $f(n) \neq O(g(n))$ and $g(n) \neq O(f(n))$

Solution: Sometimes true: For $f(n) = 1$ and $g(n) = \|n * \sin(n)\|$ it is true, while for any $f(n) = O(g(n))$, e.g. $f(n) = g(n) = 1$, it is not true.

Problem 1-2. Recurrences

Give asymptotic upper and lower bounds for $T(n)$ in each of the following recurrences. Assume that $T(n)$ is constant for $n \le 3$. Make your bounds as tight as possible, and justify your answers.

(a) $T(n) = 2T(n/3) + n \lg n$

Solution: By Case 3 of the Master Method, we have $T(n) = \Theta(n \lg n)$.

(b) $T(n) = 3T(n/5) + \lg^2 n$

Solution: By Case 1 of the Master Method, we have $T(n) = \Theta(n^{\log_5(3)})$.

(c) $T(n) = T(n/2) + 2^n$

Solution: Case 3 of master's theorem, (check that the regularity condition holds), $\Theta(2^n)$.

(d) $T(n) = T(\sqrt{n}) + \Theta(\lg \lg n)$

Solution: Change of variables: let $m = \lg n$. Recurrence becomes $S(m) = S(m/2) + \Theta(\lg m)$. Case 2 of master's theorem applies, so $T(n) = \Theta((\lg \lg n)^2)$.

(e) $T(n) = 10T(n/3) + 17n^{1.2}$

Solution: Since $\log_3 9 = 2$, so $\log_3 10 > 2 > 1.2$. Case 1 of master's theorem applies, $\Theta(n^{\log_3 10})$.

(f) $T(n) = 7T(n/2) + n^3$

Solution: By Case 3 of the Master Method, we have $T(n) = \Theta(n^3)$.

(g) $T(n) = T(n/2 + \sqrt{n}) + \sqrt{6046}$

Solution: By induction, $T(n)$ is a monotonically increasing function. Thus, for large enough n, $T(n/2) \le T(n/2 + \sqrt{n}) \le T(3n/4)$. At each stage, we incur constant cost $\sqrt{6046}$, but we decrease the problem size to atleast one half and at most three-quarters. Therefore $T(n) = \Theta(\lg n)$.

(h) $T(n) = T(n-2) + \lg n$

Solution: $T(n) = \Theta(n \log n)$. This is $T(n) = \sum_{i=1}^{n/2} \lg 2i \geq \sum_{i=1}^{n/2} \lg i \geq (n/4)(\lg n/4) = \Omega(n \lg n)$. For the upper bound, note that $T(n) \leq S(n)$, where $S(n) = S(n-1) + \lg n$, which is clearly $O(n \lg n)$.

(i) $T(n) = T(n/5) + T(4n/5) + \Theta(n)$

Solution: Master's theorem doesn't apply here. Draw recursion tree. At each level, do $\Theta(n)$ work. Number of levels is $\log_{5/4} n = \Theta(\lg n)$, so guess $T(n) = \Theta(n \lg n)$ and use the substitution method to verify guess.

In the $f(n) = \Theta(n)$ term, let the constants for $\Omega(n)$ and $O(n)$ be n_0, c_0 and c_1, respectively. In other words, let for all $n \geq n_0$, we have $c_0 n \leq f(n) \leq c_1 n$.

- First, we show $T(n) = O(n)$.
  For the base case, we can choose a sufficiently large constant d_1 such that $T(n) < d_1 n \lg n$.
  For the inductive step, assume for all $k < n$, that $T(k) < d_1 n \lg n$. Then for $k = n$, we have

$$\begin{aligned} T(n) &\leq T\left(\frac{n}{5}\right) + T\left(\frac{4n}{5}\right) + c_1 n \\ &\leq d_1 \frac{n}{5} \lg\left(\frac{n}{5}\right) + d_1 \frac{4n}{5} \lg\left(\frac{4n}{5}\right) + c_1 n \\ &= d_1 n \lg n - \frac{d_1 n}{5} \lg 5 - \frac{4 d_1 n}{5} \lg\left(\frac{5}{4}\right) + c_1 n \\ &= d_1 n \lg n - n\left(\left(\frac{\lg 5 + 4\lg(5/4)}{5}\right) d_1 - c_1\right). \end{aligned}$$

  The residual is negative as long as we pick $d_1 > 5c_1/(\lg 5 + 4\lg(5/4))$. Therefore, by induction, $T(n) = O(n \lg n)$.
- To show that $T(n) = \Omega(n)$, we can use almost the exact same math.
  For the base case, we choose a sufficiently small constant d_0 such that $T(n) > d_0 n \lg n$.
  For the inductive step, assume for all $k < n$, that $T(k) > d_0 n \lg n$. Then, for $k = n$, we have

$$\begin{aligned} T(n) &\geq T\left(\frac{n}{5}\right) + T\left(\frac{4n}{5}\right) + c_0 n \\ &\geq d_0 \frac{n}{5} \lg\left(\frac{n}{5}\right) + d_0 \frac{4n}{5} \lg\left(\frac{4n}{5}\right) + c_0 n \\ &= d_0 n \lg n + n\left(c_0 - \left(\frac{\lg 5 + 4\lg(5/4)}{5}\right) d_0\right). \end{aligned}$$

The residual is positive as long as $d_0 < 5c_0/(\lg 5 + 4\lg(5/4))$. Thus, $T(n) = \Omega(n \lg n)$.

(j) $T(n) = \sqrt{n}\, T(\sqrt{n}) + 100n$

Solution: Master's theorem doesn't apply here directly. Pick $S(n) = T(n)/n$. The recurrence becomes $S(n) = S(\sqrt{n}) + 100$. The solution of this recurrece is $S(n) = \Theta(\lg\lg n)$. (You can do this by a recursion tree, or by substituting $m = \lg n$ again.) Therefore, $T(n) = \Theta(n \lg\lg n)$.

Problem 1-3. Unimodal Search

An array $A[1\,..\,n]$ is ***unimodal*** if it consists of an increasing sequence followed by a decreasing sequence, or more precisely, if there is an index $m \in \{1, 2, \ldots, n\}$ such that

- $A[i] < A[i+1]$ for all $1 \le i < m$, and
- $A[i] > A[i+1]$ for all $m \le i < n$.

In particular, $A[m]$ is the maximum element, and it is the unique "locally maximum" element surrounded by smaller elements ($A[m-1]$ and $A[m+1]$).

(a) Give an algorithm to compute the maximum element of a unimodal input array $A[1\,..\,n]$ in $O(\lg n)$ time. Prove the correctness of your algorithm, and prove the bound on its running time.

Solution: Notice that by the definition of unimodal arrays, for each $1 \le i < n$ either $A[i] < A[i+1]$ or $A[i] > A[i+1]$. The main idea is to distinguish these two cases:

1. By the definition of unimodal arrays, if $A[i] < A[i+1]$, then the maximum element of $A[1..n]$ occurs in $A[i+1..n]$.
2. In a similar way, if $A[i] > A[i+1]$, then the maximum element of $A[1..n]$ occurs in $A[1..i]$.

This leads to the following divide and conquer solution (note its resemblance to binary search):

```
a, b ← 1, n
while a < b
    do mid ← ⌊(a + b)/2⌋
        if A[mid] < A[mid + 1]
          then a ← mid + 1
        if A[mid] > A[mid + 1]
          then b ← mid
return A[a]
```

The precondition is that we are given a unimodal array $A[1..n]$. The postcondition is that $A[a]$ is the maximum element of $A[1..n]$. For the loop we propose the invariant "The maximum element of $A[1..n]$ is in $A[a..b]$ and $a \leq b$".

When the loop completes, $a \geq b$ (since the loop condition failed) and $a \leq b$ (by the loop invariant). Therefore $a = b$, and by the first part of the loop invariant the maximum element of $A[1..n]$ is equal to $A[a]$.

We use induction to prove the correctness of the invariant. Initially, $a = 1$ and $b = n$, so, the invariant trivially holds. Suppose that the invariant holds at the start of the loop. Then, we know that the maximum element of $A[1..n]$ is in $A[a..b]$. Notice that $A[a..b]$ is unimodal as well. If $A[mid] < A[mid + 1]$, then the maximum element of $A[a..b]$ occurs in $A[mid+1..b]$ by case 1. Hence, after $a \leftarrow mid+1$ and b remains unchanged in line 4, the maximum element is again in $A[a..b]$. The other case is symmetric.

To complete the proof, we need to show that the second part of the invariant $a \leq b$ is also true. At the start of the loop $a < b$. Therefore, $a \leq \lfloor (a + b)/2 \rfloor < b$. This means that $a \leq mid < b$ such that after line 4 or line 5 in which a and b get updated $a \leq b$ holds once more.

The divide and conquer approach leads to a running time of $T(n) = T(n/2)+\Theta(1) = \Theta(\lg n)$.

A polygon is ***convex*** if all of its internal angles are less than $180°$ (and none of the edges cross each other). Figure 1 shows an example. We represent a convex polygon as an array $V[1 \,..\, n]$ where each element of the array represents a vertex of the polygon in the form of a coordinate pair (x, y). We are told that $V[1]$ is the vertex with the minimum x coordinate and that the vertices $V[1 \,..\, n]$ are ordered counterclockwise, as in the figure. You may also assume that the x coordinates of the vertices are all distinct, as are the y coordinates of the vertices.

(b) Give an algorithm to find the vertex with the maximum x coordinate in $O(\lg n)$ time.

Solution: Notice that the x-coordinates of the vertices form a unimodal array and we can use part (a) to find the vertex with the maximum x-coordinate in $\Theta(\lg n)$ time.

(c) Give an algorithm to find the vertex with the maximum y coordinate in $O(\lg n)$ time.

Solution: After finding the vertex $V[max]$ with the maximum x-coordinate, notice that the y-coordinates in $V[max], V[max + 1], \ldots, V[n - 1], V[n], V[1]$ form a unimodal array and the maximum y-coordinate of $V[1..n]$ lies in this array. Again part (a) can be used to find the vertex with the maximum y-coordinate. The total running time is $\Theta(\lg n)$.

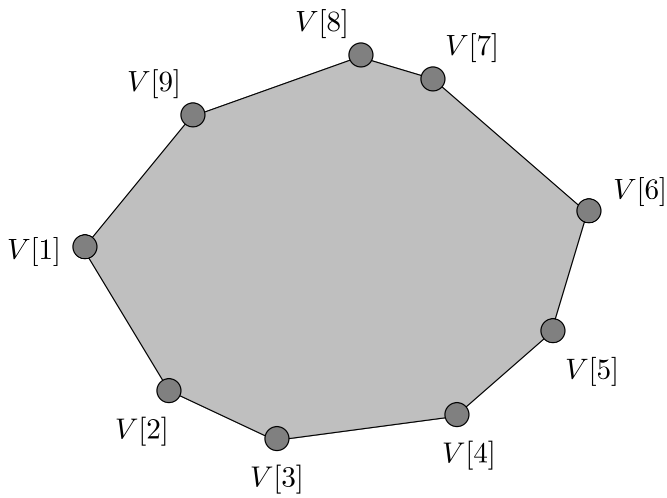

Figure 1: An example of a convex polygon represented by the array $V[1\,..\,9]$. $V[1]$ is the vertex with the minimum x-coordinate, and $V[1\,..\,9]$ are ordered counterclockwise.