*Introduction to Algorithms* September 17, 2002
Massachusetts Institute of Technology 6.046J/18.410J
Professors Erik Demaine and Shafi Goldwasser Handout 8

# Problem Set 1 Solutions

(Exercises were not to be turned in, but we're providing the solutions for your own interest.)

**Exercise 1-1.** See problem 2 for binary search pseudocode.

**Exercise 1-2.** First sort the array $S$ in $\Theta(n \log n)$ time. Next, for each $i = 1, \ldots, n$, do a binary search for $x - S[i]$. If the search returns an index $j$, then $S[i] + S[j] = x$ as desired. If none of the searches succeed, then no two elements in $S$ sum to $x$.

**Exercise 1-3.** Note that, for all $n$,

$$0 \leq \frac{1}{2}(f(n) + g(n)) \leq \max(f(n), g(n)) \leq f(n) + g(n).$$

By definition (with $c_1 = 1/2$, $c_2 = 1$, and $n_0 = 1$): $\max(f(n), g(n)) = \Theta(f(n) + g(n))$.

**Exercise 1-4.** Translating $O(n^2)$ into words, the sentence reads (roughly): "the running time of algorithm $A$ is **at least at most** $cn^2$ for sufficiently large $n$." This is gibberish. To express "at least," use $\Omega$ notation.

**Exercise 1-5.** Let $n = 2^m \iff m = \lg n$, so we get $T(2^m) = 2T(2^{m/2}) + 1$. Let $S(m) = T(2^m)$, so $S(m) = 2S(m/2) + 1$. Then $S(m) = \Theta(m)$ by the Master Theorem; $T(n) = T(2^m) = S(m) = \Theta(m) = \Theta(\lg n)$.

---

**Problem 1-1.** The functions, grouped asymptotically from largest to smallest, are as follows (functions $f, g$ on the same line are such that $f(n) = \Theta(g(n))$):

$$n!$$

$$3^n$$

$$2^n$$

$$(\lg n)^{\lg n} \qquad n^{\lg \lg n}$$

$$n^2 \qquad \sum_{k=1}^{n} k$$

$$n \lg n \qquad \lg(n!)$$

$$(\sqrt{2})^{\lg n}$$

$$\ln n \qquad \sum_{k=1}^{n} \frac{1}{k}$$

$$\lg^* n \qquad \lg^*(\lg n)$$

$$2^{(100^{100})} \qquad 1$$

A few notes about how these results are obtained:

$3^n \neq \Theta(2^n)$ because their ratio is $(3/2)^n$, which grows without bound.

$(\lg n)^{\lg n} = (2^{\lg\lg n})^{\lg n} = (2^{\lg n})^{\lg\lg n} = n^{\lg\lg n}$.

$\sum_{k=1}^{n} k = n(n+1)/2 = \Theta(n^2)$.

$n! \approx \sqrt{2\pi n}(n/e)^n$ by Stirling's approximation, so $\lg n! = \Theta(n \lg n)$.

$(\sqrt{2})^{\lg n} = 2^{(\lg n)/2} = n^{1/2}$.

$\sum_{k=1}^{n} 1/k \approx \ln n$, which can be seen by approximating the integral from above and below.

$\lg^*(\lg n) = \lg^*(2^{\lg n}) - 1 = \lg^* n - 1 = \Theta(\lg^* n)$.

$2^{(100^{100})}$, despite being an unfathomably large number, is still a constant.

**Problem 1-2.** Binary search.

**(a)** Here is one possible pseudocode procedure, which searches for the element $v$ in the range of $A[i, \ldots, j]$ (on an array of $n$ elements, it should initially be called with $i = 1$ and $j = n$):

```
BINARY-SEARCH-REC(A, v, i, j)
 1  if i > j                          ▷ no elements left to search
 2     return ⊥
 3  mid ← ⌊(i + j)/2⌋
 4  if A[mid] = v
 5     return mid
 6  elseif A[mid] > v
 7     return BINARY-SEARCH-REC(A, v, i, mid − 1)
 8  else
 9     return BINARY-SEARCH-REC(A, v, mid + 1, j)
```

**(b)** Here is pseudocode for an iterative version of binary search:

```
BINARY-SEARCH-ITER(A, v)
 1  i ← 1
 2  j ← length[A]
 3  while i ≤ j
 4      mid ← ⌊(i + j)/2⌋
 5      if A[mid] = v
 6          return mid
 7      elseif A[mid] > v
 8          j ← mid − 1
 9      else
10          i ← mid + 1
11  return ⊥
```

**(c)** The pre-condition is that $A$ is sorted, i.e. $A[1] \leq A[2] \leq \cdots \leq A[n]$. The post-condition of the loop is that there is no $k$ such that $A[k] = v$.

The loop invariant is that for all $k > j$, $A[k] > v$ and for all $k < i$, $A[k] < v$. This is trivially true upon entry to the loop (when $i = 1$ and $j = n$). Notice that $i \leq \mathit{mid} \leq j$. Let $i'$ and $j'$ be the values of $i$ and $j$, respectively, at the end of the loop.

If $A[\mathit{mid}] > v$, then $i' = i$, so for all $k < i'$, $A[k] < v$ by the inductive hypothesis. Also, $j' = \mathit{mid} - 1$, so for all $k > j' = \mathit{mid} - 1$, $A[k] \geq A[\mathit{mid}] > v$ by the inductive hypothesis and the sorted-array pre-condition.

If $A[\mathit{mid}] < v$, then $i' = \mathit{mid}+1$, so for all $k < i'$, $A[k] \leq A[\mathit{mid}] < v$ by the inductive hypothesis and sorted-array pre-condition. Also, $j' = j$, so for all $k > j'$, $A[k] > v$ by the inductive hypothesis. Therefore the loop invariant is correct by induction.

We also need to prove that the function terminates with the correct output, and that the function always terminates. If the function ever returns an index $k$, then if must be that $A[k] = v$. If the loop terminates without returning an index, then $i > j$. Then for any index $k$, either $k < i$, which implies $A[k] < v$, or $k > j$, which implies $A[k] > v$ (by the loop invariant). Therefore there is no such $k$ such that $A[k] = v$, which is the post-condition (and $\perp$ is the correct output).

Finally, we prove termination: we claim that $j - i$ is strictly decreasing with each loop iteration, therefore the loop terminates. Because $i \leq j$, $\mathit{mid} + 1 > i$, and $\mathit{mid} - 1 < j$, so with each iteration $j$ strictly decreases or $i$ strictly increases, so $j - i$ is strictly decreasing. This completes the proof of correctness.

**Problem 1-3.** Solving recurrences.

**(a)** Case 3 of the Master Theorem applies ($f(n) = n^3 = \Omega(n^{2+\epsilon})$ for $\epsilon = 0.1 > 0$), so $T(n) = \Theta(n^3)$.

**(b)** The answer is $T(n) = \Theta(n^3)$. We can use the expansion technique to prove both bounds. For the upper bound, note that $T(n) = n^2 + (n-2)^2 + \cdots + 1 \leq n^2 + n^2 +$

$\cdots + n^2 \leq n \cdot n^2 = n^3 = O(n^3)$. For the lower bound, take the first $\lfloor n/4 \rfloor$ terms, which are all at least $(n - n/2)^2 = n^2/4$, so $T(n) \geq n^3/16 = \Omega(n^3)$.

**(c)** Here we must compare $n^{\log_b a} = n^{\log_3 5}$ and $f(n) = n^{3/2}$. In fact, $3/2 > \log_3 5 \iff 3^{3/2} > 5 \iff \sqrt{27} > 5$, which is true. Therefore Case 3 of the Master Theorem applies, and $T(n) = \Theta(n\sqrt{n})$.

**(d)** Here $a = 2$, $b = 8$ (so $\log_b a = 1/3$), and $f(n) = \sqrt[3]{n} \ln n$. Note that *none* of the three cases of the Master Theorem holds[1], because $f(n) \neq \Theta(n^{1/3})$ and $f(n) \neq \Omega(n^{1/3+\epsilon})$ for any constant $\epsilon > 0$. We use the Akra-Bazzi theorem instead: $2(1/8)^p = 1$ implies $p = 1/3$, so the solution is:

$$T(n) = \Theta(n^{1/3}) + \Theta(n^{1/3} \int_1^n \frac{x^{1/3} \ln x}{x^{4/3}}\, dx).$$

Using integration by parts, we find that the integral evaluates to $\frac{\ln^2 n}{2}$, so $T(n) = \Theta(n^{1/3} \ln^2 n)$.

**(e)** Here we use substitution. Let $n = 3^m$, yielding $T(3^m) = 3T(3^{m/3}) + m$. Next, let $S(m) = T(3^m) = T(n)$, yielding $S(m) = 3T(m/3) + m$. By the Master Theorem, $S(m) = \Theta(m \log m)$. Since $T(n) = S(m)$, we substitute $m = \log_3 n$ to get $T(n) = \Theta(\log n \log\log n)$.

**(f)** Here we can use a recursion tree. The leaves of the tree sum to $\Theta(2^n)$, so $T(n) = \Omega(2^n)$. In fact, we claim also that $T(n) = O(2^n)$, so $T(n) = \Theta(2^n)$. Specifically, we will prove (by induction) that there exist positive constants $b, c, n_0$ such that $T(n) \leq b2^n - c(n+1)^7$ whenever $n \geq n_0$.

First we will handle the inductive case, choosing $c$ and $n_0$ appropriately so that the proof goes through, then choosing $b$ so that the base case of $n = n_0$ holds. Assume $T(n) \leq b2^n - c(n+1)^7$ for all $n$, $n_0 \leq n \leq i$. Then

$$\begin{aligned} T(i+1) &= 2T(i) + (i+1)^7 \\ &\leq 2(b2^i - c(i+1)^7) + (i+1)^7 \\ &= b2^{i+1} - (2c-1)(i+1)^7. \end{aligned}$$

Now it is enough to show that $(2c-1)(i+1)^7 \geq c(i+2)^7 \iff \frac{2c-1}{c} \geq (\frac{i+2}{i+1})^7$. If we set $n_0 = 20$, then $(\frac{i+2}{i+1})^7 < 1.4$ because $i \geq n_0$. Then setting $c = 5/3$ means that $\frac{2c-1}{c} = 1.4 > (\frac{i+2}{i+1})^7$, as desired. Finally, we handle the base case of $n = n_0 = 20$: note that $T(20)$ is a constant, so we simply choose $b$ to be a large enough constant such that $T(20) \leq b2^{20} - c(21^7) \iff b \geq (T(20) + c(21^7))/2^{20}$. This completes the proof.

**Problem 1-4.** Monge arrays.

[1] the version of the Master Theorem presented in lecture is stronger and does apply here, but the version in CLRS does not

**(a)** First we prove the "only if" part, i.e. we may assume that array $A$ is Monge. We simply set $k = i + 1$ and $\ell = j + 1$, and by the Monge property, we have $A[i, j] + A[i + 1, j + 1] \leq A[i, j + 1] + A[i + 1, j]$ as desired.

Now we prove the "if" part, i.e. we may assume that $A[i, j] + A[i + 1, j + 1] \leq A[i, j + 1] + A[i + 1, j]$ and we want to prove that the array is Monge. First we do induction on the rows, that is, we will prove that for any $k > i$,

$$A[i, j] + A[k, j + 1] \quad \leq \quad A[i, j + 1] + A[k, j] \tag{1}$$

For the base case, equation (1) is true when $k = i + 1$ by our original assumption. For the inductive case, assume that equation (1) is true for some $k = i' > i$; we will prove it true for $k = (i' + 1)$. By the original hypothesis, we have

$$\begin{aligned} & A[i', j] + A[i' + 1, j + 1] \quad \leq \quad A[i', j + 1] + A[i' + 1, j] \\ \iff & -A[i', j + 1] + A[i' + 1, j + 1] \quad \leq \quad -A[i', j] + A[i' + 1, j]. \end{aligned} \tag{2}$$

Because (1) is true for $k = i'$, we can replace $k$ by $i'$ and add equation (2), yielding:

$$A[i, j] + A[i' + 1, j + 1] \leq A[i, j + 1] + A[i' + 1, j]$$

so equation (1) is also true for $k = i' + 1$ (and therefore all $k > i$) as desired.

A similar argument (induction on the columns) proves that for any $k > i$, $\ell > j$ we have $A[i, j] + A[k, \ell] \leq A[i, \ell] + A[k, j]$. Therefore, $A$ is indeed Monge.

**(b)** Assume by contradiction that the property is untrue. Then let $i$ be the smallest value for which $f(i) > f(i + 1)$. Let $j = f(i + 1)$, $k = i + 1$, and $\ell = f(i)$. Then $i < k$ and $j < l$ as required by the Monge property. Also, $A[i, j] > A[i, \ell]$ because $A[i, \ell]$ is the leftmost minimum element of row $i$, and $j < \ell$. Likewise, $A[k, \ell] \geq A[k, j]$ because $A[k, j]$ is a minimum element of row $k = i + 1$. Then we have $A[i, j] + A[k, \ell] > A[i, \ell] + A[k, j]$, in contradiction of the Monge property.

**(c)** Say the previous step returns an array $F'[1, \ldots, \lfloor m/2 \rfloor]$ in which $F'[i] = f_{A'}(i)$, for all $i = 1, \ldots, \lfloor m/2 \rfloor$. Our algorithm will produce $F[1, \ldots, m]$ such that $F[i] = f_A(i)$ for all $i = 1, \ldots, m$. The algorithm works as follows:

> For $i = 1, \ldots, m$: if $i$ is even, let $F[i] = F'[i/2]$. Else, scan elements $A[i, F'[(i - 1)/2]]$ through $A[i, F'[(i + 1)/2]]$ to discover the leftmost minimum, then set $F[i]$ to be its column index.

By the property in part **(c)**, correctness is clear. For the running time, note that the outer loop is executed $O(m)$ times (once for each odd $i$), and a total of at most $2n$ array elements are examined by the inner loop over the entire run of the algorithm (column $F'[i]$ is examined twice for each $i$). Therefore the running time is $O(m + n)$.

**(d)** Let $T(m, n)$ be the running time of the algorithm on an $m \times n$ array. Then

$$T(m, n) = T(m/2, n) + O(m + n) \Rightarrow T(m, n) \leq T(m/2, n) + c(m + n)$$

for some constant $c > 0$, and $T(1, n) = O(n)$ for all $n$. By expansion, we get a $O(\log m)$ factor of $cn$, and an additional term of $c(m + m/2 + \ldots + 1) \leq 2cm$, so $T(m, n) = O(m + n \log m)$.