CSE530: Algorithms & Complexity
Assignment 1: Suggested Answers

Antoine Vigneron

March 25, 2018

1. For each pair $f(n), g(n)$ of functions below, which statement is correct: $f(n) = o(g(n))$, or $f(n) = \Omega(g(n))$? You do not need to prove your answer.

(a)	$f(n) = 1000n^2 + 100$	$g(n) = n^3$
(b)	$f(n) = n^2 + 2$	$g(n) = 10n^2 - n + 1$
(c)	$f(n) = 1/\log n$	$g(n) = \pi$
(d)	$f(n) = n^2 + n \log n$	$g(n) = n^2$
(e)	$f(n) = \sqrt{n}$	$g(n) = n^{1/3} \log n$

Answer. (a) $f(n) = o(g(n))$ (b) $f(n) = \Omega(g(n))$ (c) $f(n) = o(g(n))$ (d) $f(n) = \Omega(g(n))$ (e) $f(n) = \Omega(g(n))$

2. In this problem, we consider two functions $f(n)$ and $g(n)$ such that $f(n) = \Theta(g(n))$, and we want to determine whether $f(n) + g(n) = \Theta(g(n))$.

(a) Prove that if $f(n) \geqslant 0$ and $g(n) \geqslant 0$ for all n, and $f(n) = \Theta(g(n))$, then $f(n) + g(n) = \Theta(g(n))$.

(b) Give an example of two functions f and g such that $f(n) = \Theta(g(n))$ is true, but $f(n) + g(n) = \Theta(g(n))$ is false.

Answer a. As $f(n) = \Theta(g(n))$, there exist constants n_0, $c_1 > 0$, $c_2 > 0$ such that $n \geqslant n_0$ implies $c_1|g(n)| \leqslant |f(n)| \leqslant c_2|g(n)|$. As $f(n) \geqslant 0$ and $g(n) \geqslant 0$, it means that $c_1 g(n) \leqslant f(n) \leqslant c_2 g(n)$. We add $g(n)$ to these inequalities and obtain

$$(c_1 + 1)g(n) \leqslant f(n) + g(n) \leqslant (c_2 + 1)g(n) \qquad \text{for all } n \geqslant n_0.$$

As c_1, c_2, $f(n)$ and $g(n)$ are non-negative, we have $(c_1 + 1)g(n) = (c_1 + 1)|g(n)|$, $f(n) + g(n) = |f(n) + g(n)|$, and $(c_2 + 1)g(n) = (c_2 + 1)|g(n)|$. It implies that

$$(c_1 + 1)|g(n)| \leqslant |f(n) + g(n)| \leqslant (c_2 + 1)|g(n)| \qquad \text{for all } n \geqslant n_0.$$

As $c_1 + 1 > 0$ and $c_2 + 1 > 0$, it proves that $f(n) + g(n) = \Theta(g(n))$.

Answer b. Take the constant functions $f(n) = 1$ and $g(n) = -1$ for all $n \in \mathbb{N}$.

3. Given an array $A[1 \dots n]$ of n numbers, the 2-MIN problem is to find the two smallest numbers in A. For instance, if $n = 5$ and $A = [4, 3, 7, 2, 8]$ then we should return the pair $2, 3$. More formally, we should return the smallest number $A[i^*]$, followed by the smallest number $A[j^*]$ such that $j^* \neq i^*$. We use Algorithm 1 to solve this problem.

Algorithm 1

```
1: procedure SLOW 2-MIN(A[1...n])
2:     x ← A[1], y ← A[2]
3:     for i ← 1, n − 1 do
4:         for j ← i + 1, n do
5:             if A[i] + A[j] < x + y then
6:                 x ← A[i]
7:                 y ← A[j]
8:     return min(x, y), max(x, y)
```

(a) Prove that Algorithm 1 is correct using the loop invariant method. (You only need to prove a loop invariant for the outer loop, not for the inner loop.)

(b) Assume that each execution of Line i takes time c_i, where c_i is a constant. Give the running time of Algorithm 1 as a function of n, and using the constants $c_2, c_3, \dots, c_7$.

(c) Give the running time of Algorithm 1 using the $\Theta(\cdot)$ notation, and justify your answer.

(d) Give an asymptotically faster algorithm for the 2-MIN problem. (You should give its pseudocode and its running time using the $\Theta(.)$ notation.)

Answer a. We use the invariant below:

- At the beginning of the ith iteration of the outer loop, if $i > 1$, then $x = A[i']$ and $y = A[j']$ where $A[i'], A[j']$ is the pair that minimizes $A[i'] + A[j']$ among all pairs that satisfy $1 \leqslant i' < i$ and $i' < j' \leqslant n$. If $i = 1$, then $A[1] = x$ and $A[2] = y$.

We now prove the three properties of this invariant: Initialization, maintenance and termination.

Initialization. At the beginning of the first iteration, we have $i = 1$, $x = A[1]$ and $y = A[2]$ so our invariant is true.

Maintenance. Suppose that $i > 1$. Then at the beginning of the iteration, $x = A[i']$ and $y = A[j']$ where $1 \leqslant i' < i - 1$, $i < j' \leqslant n$ and $A[i'] + A[j']$ is minimum. If there is no pair $A[i], A[j'']$ such that $i < j''$ and $A[i] + A[j''] < A[i'] + A[j']$, then x and y are not updated during the iteration of the loop and the invariant is maintained. If there is such a pair $A[i], A[j'']$, then inner loop will update x, y with the smallest such pair, so our invariant is also maintained.

The case $i = 1$ is similar: At the end of the execution of the inner loop, x, y record the pair $A[1], A[j']$ that minimizes $A[1] + A[j']$ for $j' > 1$.

Termination. The loop terminates when $i = n$. At this point, our invariant says that the pair x, y records the pair $A[i'], A[j]$ such that $1 \leqslant i' < n$ and $i' < j \leqslant n$ and $A[i'] + A[j']$ is minimum. This pair is chosen among all possible pair of distinct elements, and the sum is minimized, so it is formed by the two smallest numbers in the array.

Answer b. In the worst case, the contributions of each lines are

2: c_2

3: $c_3 n$

4: $c_4(n + n - 1 + n - 2 + \cdots + 2) = c_4\left(\frac{n(n+1)}{2} - 1\right)$

5–7: $(c_5 + c_6 + c_7)\binom{n}{2} = (c_5 + c_6 + c_7)\frac{n(n-1)}{2}$

8: c_8

so the overall running time is

$$\frac{c_4 + c_5 + c_6 + c_7}{2} n^2 + \left(c_3 + \frac{c_4 - c_5 - c_6 - c_7}{2}\right) n + c_2 - c_4 + c_8.$$

Remark: If we are interested in the best case running time, we can just delete c_6 and c_7 from the formula above. It remains $\Theta(n^2)$.

Answer c. The expression found in (c) is a degree-2 polynomial in n, so it is $\Theta(n^2)$.

Answer d. The pseudocode is given as Algorithm 2. Its running time is $\Theta(n)$.

Algorithm 2

```
1: procedure FAST 2-MIN(A[1...n])
2:     x ← min(A[1], A[2])
3:     y ← max(A[1], A[2])
4:     for i ← 3, n do
5:         if A[i] < x then
6:             y ← x
7:             x ← A[i]
8:         else if A[i] < y then
9:             y ← A[i]
10:    return x, y
```

Other possible answer: First run MERGE SORT on A, and then return $A[1], A[2]$.

4. We consider the Dynamic Time Warping (DTW) problem with outliers. More precisely, given two point sequences $A = (a_1, \ldots, a_m)$ and $B = (b_1, \ldots, b_n)$, and an integer $p < m$, we consider the problem of computing the smallest possible value of $DTW(A', B)$ where A' is obtained by deleting p points from A. (See Figure 1.) So we need to minimize over all possible choices of these p points of A, and over all possible couplings for each such choice of p points. These p points are called the *outliers*. The motivation could be that A contains p erroneous points that we would like to identify and discard.

(a) Give an $O(mnp^2)$-time algorithm for this problem: Given A, B and p, it should return the minimum value of $DTW(A', B)$ over all possible sequences A' obtained by deleting p points from A.

(b) Show how to modify the algorithm from (a) so that it returns the p outliers efficiently.

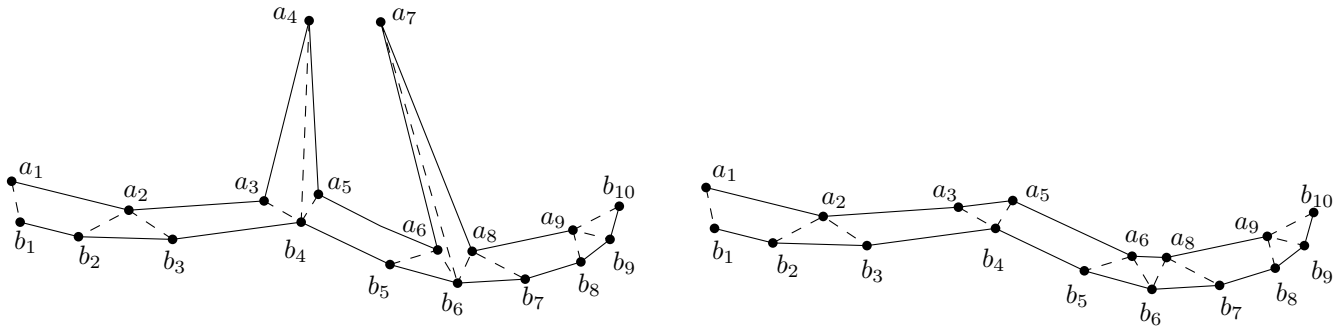


Figure 1: (Left) two point sequences $A = (a_1, \ldots, a_9)$ and $b = (b_1, \ldots, b_{10})$ and the optimal coupling (dashed) corresponding to $DTW(A, B)$. (Right) If we allow $p = 2$ outliers, then the points a_4 and a_7 are deleted from A, thus obtaining $A' = (a_1, a_2, a_3, a_5, a_6, a_8, a_9)$. The optimal coupling for $DTW(A', B)$ is represented by dashed line segments.

Answer a. As we did in class, we denote $A_i = (a_1, \ldots, a_i)$ and $B_j = (b_1, \ldots, b_j)$. Let $D(i, j, k)$ denote the optimal value of $DTW(A_i, B_j)$ with k outliers, under the constraint that the last pair in the coupling is (i, j). In other words, it is the smallest value of $DTW(A'_i, B_j)$ where A'_i is a subsequence of A_i with $i - k$ elements, and whose last element is a_i. Then $D(i, j, k)$ satisfies the following recurrence relation whenever $0 \leqslant k \leqslant i - 2$ and $j \geqslant 2$:

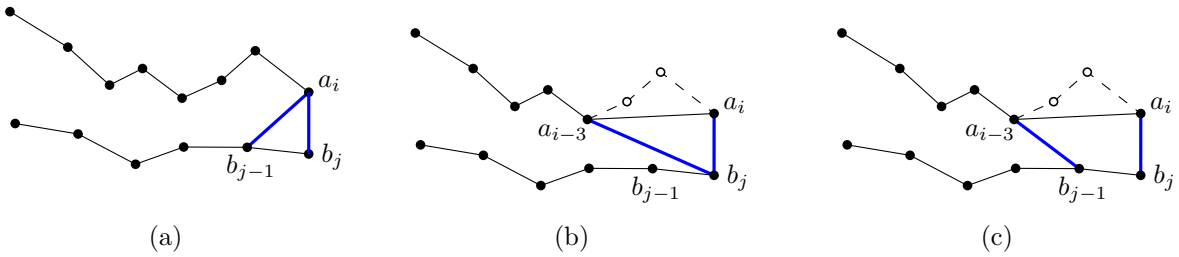
$$D^k_{i,j} = d(a_i, b_j) + \min\left(D^k_{i,j-1}, \min_{\ell=0,\ldots,k}\left(D^{k-\ell}_{i-\ell-1,j}\right), \min_{\ell=0,\ldots,k}\left(D^{k-\ell}_{i-\ell-1,j-1}\right)\right) \tag{1}$$

This formula follows from the case analysis below (Figure 2):

(a) The second last pair in the coupling is $(i, j-1)$ and then $D^k_{i,j} = d(a_i, b_j) + D^k_{i,j-1}$.

(b) The second last pair is $(i-1-\ell, j)$ because we skipped ℓ outliers before a_i. Then $D^k_{i,j} = d(a_i, b_j) + D^{k-\ell}_{i-\ell-1,j}$.

(c) The second last pair is $(i-1-\ell, j-1)$ because we skipped ℓ outliers before a_i, and we moved to b_{j-1} at the same time. Then $D^k_{i,j} = d(a_i, b_j) + D^{k-\ell}_{i-\ell-1,j-1}$.



(a) (b) (c)

Figure 2: The three cases in the recurrence relation.

When $k = i - 1$, then $a_1, \ldots, a_{i-1}$ are outliers and thus

$$D_{i,j}^{i-1} = \sum_{q=1}^{j} d(a_i, b_q). \tag{2}$$

When $j = 1$ then only case (b) applies and we have

$$D_{i,1}^{k} = d(a_i, b_1) + \min_{\ell=0,\ldots,k} \left(D_{i-\ell-1,1}^{k-\ell} \right). \tag{3}$$

In the end, we just need to return the smallest value among $D[m-k, n, p-k]$ over $k = \{0, \ldots, p\}$, to account for the fact that A can end with k outliers $a_{m-k+1}, \ldots, a_m$.

We obtain an $O(mnp^2)$-time algorithm by just implementing these formula using dynamic programming.

Answer b. Similar to what we did in class for the step function problem, We modify the Algorithm 3 above so that it recalls the indices of the optimal subproblem during a recursive call.

Algorithm 3 DTW with outliers

function DTW($(a_1, \ldots, a_m), (b_1, \ldots, b_n), p$)
  $D \leftarrow$ new $m \times n \times p$ array
  $D[1, 1, 0] \leftarrow d(a_1, b_1)$
  for $i \leftarrow 1, m$ **do** ▷ Implementing Equation (2)
    for $j \leftarrow 1, n$ **do**
      $D[i, j, i-1] \leftarrow \sum_{q=1}^{j} d(a_i, b_q)$
  for $k \leftarrow 0, p$ **do** ▷ Implementing Equation (3)
    for $i \leftarrow k+2, m$ **do**
      $t \leftarrow \infty$
      for $\ell \leftarrow 0, k$ **do**
        $t \leftarrow \min(t, D[i-\ell-1, 1, k-\ell])$
      $D[i, 1, k] \leftarrow d(a_i, b_1) + t$
  for $k \leftarrow 0, p$ **do** ▷ Implementing Equation (1)
    for $i \leftarrow k+1, m$ **do**
      for $j \leftarrow 2, n$ **do**
        $t \leftarrow D[i, j-1, k]$ ▷ Case (a)
        for $\ell \leftarrow 0, k$ **do**
          $t \leftarrow \min(t, D[i-\ell-1, j, k-\ell])$ ▷ Case (b)
          $t \leftarrow \min(t, D[i-\ell-1, j-1, k-\ell])$ ▷ Case (c)
        $D[i, j, k] \leftarrow t + d(a_i, b_j)$
  return $\min_{k=0,\ldots,m} D[m-k, n, p-k]$

Algorithm 4 Computing the p outliers

function DTW($(a_1, \ldots, a_m), (b_1, \ldots, b_n), p$)
  $D \leftarrow$ new $m \times n \times p$ array of numbers
  $P \leftarrow$ new $m \times n \times p$ array of integer triples
  $D[1, 1, 0] \leftarrow d(a_1, b_1)$
  for $i \leftarrow 1, m$ **do** ▷ Implementing Equation (2)
    for $j \leftarrow 1, n$ **do**
      $D[i, j, i-1] \leftarrow \sum_{q=1}^{j} d(a_i, b_q)$
      $P[i, j, i-1] \leftarrow (0, 0, 0)$
  for $k \leftarrow 0, p$ **do** ▷ Implementing Equation (3)
    for $i \leftarrow k+2, m$ **do**
      $D[i, 1, k] \leftarrow \infty$
      for $\ell \leftarrow 0, k$ **do**
        if $D[i-\ell-1, 1, k-\ell]) < D[i, 1, k]$ **then**
          $D[i, 1, k] \leftarrow D[i-\ell-1, 1, k-\ell]$
          $P[i, 1, k] \leftarrow (i-\ell-1, 1, k-\ell)$
      $D[i, 1, k] \leftarrow d(a_i, b_1) + D[i, 1, k]$
  for $k \leftarrow 0, p$ **do** ▷ Implementing Equation (1)
    for $i \leftarrow k+1, m$ **do**
      for $j \leftarrow 2, n$ **do**
        $D[i, j, k] \leftarrow D[i, j-1, k]$ ▷ Case (a)
        $P[i, j, k] \leftarrow (i, j-1, k)$
        for $\ell \leftarrow 0, k$ **do**
          if $D[i-\ell-1, j, k-\ell] < D[i, j, k]$ **then** ▷ Case (b)
            $D[i, j, k] \leftarrow D[i-\ell-1, j, k-\ell]$
            $P[i, j, k] \leftarrow (i-\ell-1, j, k-\ell)$
          if $D[i-\ell-1, j, k-\ell] < D[i, j, k]$ **then** ▷ Case (c)
            $D[i, j-1, k] \leftarrow D[i-\ell-1, j-1, k-\ell]$
            $P[i, j-1, k] \leftarrow (i-\ell-1, j-1, k-\ell)$
        $D[i, j, k] \leftarrow D[i, j, k] + d(a_i, b_j)$
  $i \leftarrow m, j \leftarrow n, k \leftarrow p$
  while $k \geqslant 0$ **do** ▷ Printing the outliers in reverse order
    $(i', j', k') \leftarrow P[i, j, k]$
    if $k > k'$ **then**
      for $q \leftarrow 1, k-k'$ **do**
        PRINT a_{i-q}
    $(i, j, k) \leftarrow (i', j', k')$