CS 170
Fall 2016

Efficient Algorithms and Intractable Problems
Christos Papadimitriou and Luca Trevisan

# HW 1

## Due Sep 5, 10:00pm

**Instructions:** You are welcome to form small groups (up to 4 people total) to work through the homework, but you **must** write up all solutions by yourself. List your study partners for homework on the first page, or "none" if you had no partners.

If using LaTeX (which we recommend), use the homework template linked on this Piazza post to get started.

Begin each problem on a new page. Clearly label where each problem and subproblem begin. The problems must be submitted in order (all of P1 must be before P2, etc). For questions asking you to give an algorithm, respond in what we will refer to as the *four-part format* for algorithms: main idea, pseudocode, proof of correctness, and running time analysis.

Read the Homework FAQ Piazza post on Piazza before doing the homework for more explanation on the four-part format and other clarifications for our homework expectations.

No late homeworks will be accepted. No exceptions. This is not out of a desire to be harsh, but rather out of fairness to all students in this large course. Out of a total of approximately 13 homework assignments, the lowest two scores will be dropped.

This homework is due Monday, September 5, at 10:00pm via Gradescope. Please submit via PDF.

**1. (5 pts.) Course policies recap**

For each statement, write *OK* if it is allowed by the course policies and *Not OK* if it constitutes cheating.

(a) You ask your friend who's taken the class to help you with the homework. She gives you some general pointers (she's seen some of these problems before).

(b) You had 5 midterms on the same day and are behind on your homework. You decide to ask your classmate, who's already done the homework, for help. He tells you how to do the first three problems.

(c) You were looking up Dijkstra's on the internet, and run into a website with a problem very similar to one on your homework. You read it, including the solution, and then you close the website, write up your solution, and cite the website URL in your homework writeup.

**2. (10 pts.) Compare growth rates**

In each of the following, indicate whether $f = O(g)$, $f = \Omega(g)$, or both (in which case $f = \Theta(g)$). Briefly justify each of your answers.

| | $f(n)$ | $g(n)$ |
|---|---|---|
| (a) | $2^{n-1}$ | $2^n$ |
| (b) | $3^n$ | $n2^n$ |
| (c) | $(\log n)^{100}$ | $n^{0.1}$ |
| (d) | $n$ | $(\log n)^{\log\log n}$ |
| (e) | $n+\log n$ | $n+(\log n)^2$ |
| (f) | $\log n!$ | $n\log n$ |

**3. (15 pts.) Recurrence relations**

Solve the following recurrence relations and give a $\Theta$ bound for each of them.

(a) (i) $T(n) = 3T(n/4) + 4n^2$

(ii) $T(n) = 45T(n/3) + .1n^3$

(iii) $T(n) = 2T(\sqrt{n}) + 5$, and $T(2) = 5$. (Hint: this means the recursion tree stops when the problem size is 2)

(b) (i) Consider the recurrence relation $T(n) = 2T(n/2) + n\log n$. We can't plug it directly into the Master theorem, so solve it by adding the size of each layer.
*Hint: split up the* $\log(n/(2^i))$ *terms into* $\log n - \log(2^i)$*, and use the formula for arithmetic series.*

(ii) A more general version of Master theorem, like the one on Wikipedia, incorporates this result. The case of the master theorem which applies to this problem is:
*If* $T(n) = aT(n/b) + f(n)$ *where* $a \geq 1$*,* $b > 1$*, and* $f(n) = \Theta(n^c \log^k n)$ *where* $c = \log_b a$*, then* $T(n) = \Theta(n^c \log^{k+1} n)$.
Use the general Master theorem to solve the following recurrence relation:
$T(n) = 9T(n/3) + n^2 \log^3 n$.

**4. (15 pts.) Complex numbers review**

(i) *(No explanation necessary)* Write each of the following numbers in the form $\rho(\cos\theta + i\sin\theta)$ (for real $\rho$ and $\theta$):

(a) $-\sqrt{3}+i$

(b) The three 3-rd roots of unity

(c) The sum of the seven 7th roots of unity

(ii) Let $A(x) = ax^2+bx+c$ and $B(x) = dx^2+ex+f$. Define $C(x) = A(x)B(x)$. If $A(3) = 7$ and $B(3) = 2$, do you have enough information to determine $C(3)$? If so, briefly justify and give the value of $C(3)$. If not, explain why not.

**5. (20 pts.) Election Scandal**

This question guides you through how to answer algorithm questions and write proofs. We've done three of the four parts of the solution for you; you will write the proof of correctness.

Protests have started in Columbia, after losing Presidential candidate Ronald Frump rejected the election results. Ronald claims that the votes were counted inaccurately. In Columbia, whoever wins a majority of the popular vote becomes President. You are a prominent computer scientist, and the country is relying on you to design a provably correct algorithm to determine who won the election.

More precisely, we say that $v$ is the *majority value* for the list $L$ if it comprises $> 50\%$ of the elements of $L$. Given a list, your algorithm should compute its majority value (if one exists; if not, it is allowed to output any value).

**Main Idea**

We loop through the list, maintaining a candidate value $v$ and a counter $c$. At each iteration, we increment/decrement the counter based on whether another copy of $v$ is seen. A candidate is replaced if its counter falls to zero. Intuitively, we attempt to pair off the candidate $v$ to a different value $w \neq v$; if we instead see another $v$, we store it. After all iterations, we return the current candidate.

**Pseudocode**

```
1:  function MAJORITY(A[1...n])
2:      c ← 0
3:      v ← null

4:      for i ← 1,...,n do
5:          if c = 0 then v ← A[i]
6:          if v = A[i] then
7:              c = c + 1
8:          else
9:              c = c − 1

10:     return v
```

**Running Time Analysis**

The algorithm runs in $O(n)$ time. After some constant-time initialization, it loops over all $n$ elements; each iteration performs only constant-time operations.

**Proof of Correctness**
This is where you get involved. We now guide you through a proof that this algorithm correctly outputs the majority element (if one exists).

(a) Briefly show that $c$ never becomes negative.

(b) Prove by induction that the following invariant holds at the start of each loop iteration:

At the start of the $i$th iteration, the elements of $A[1..i]$ can be partitioned into two groups:

1. A group $U_i$ of exactly $c$ instances of the value $v$
2. A group $P_i$ of elements that can be paired off so that the two elements in each pair differ.

*Example.* Let $A = [2,3,3]$. On iteration 1, $[2]$ can be partitioned into $U_1 = [2]$ and $P_1 = []$. On iteration 2, $[2,3]$ can be partitioned into $U_2 = []$ and $P_2 = [2,3]$. On iteration 3, $[2,3,3]$ can be partitioned into $U_3 = [3]$ and $P_3 = [2,3]$.

(c) Using the invariant from part (b), prove that the algorithm is correct.

**6. (15 pts.) Two sorted arrays** You are given two sorted arrays, each of size $n$. Give as efficient an algorithm as possible to find the $k$-th smallest element in the union of the two arrays. What is the running time of your algorithm as a function of $k$ and $n$?

**7. (20 pts.) Vacation serendipity**
$n$ Berkeley students independently plan vacations to Kauai. Each student $i$ arrives on Kauai on date $s_i$ and leaves on date $t_i$ (both of which are represented as integers, with $s_i < t_i$). They neglected to coordinate their trips, so they're asking you to tell them which pair of student vacations have the longest overlap (ie, number of days during which they're both in Kauai).

For instance, if Amanda arrives on date 2 and leaves on date 4, Ban arrives on date 1 and leaves on date 10, and Chris arrives on date 6 and leaves on date 18, then Ban and Chris's vacations have the longest overlap.

Devise an algorithm that, when given $n$ students and each student $i$'s vacation arrival/departure dates $(s_i, t_i)$, finds and outputs the pair of students whose vacations have the longest overlap (you may resolve ties arbitrarily). A trivial $\Theta(n^2)$ algorithm can be achieved by comparing all pairs of intervals; look for something better.

*Hint: it would be useful to sort the students somehow.*