ShanghaiTech University

CS240 Algorithm Design and Analysis

Spring 2023

Problem Set 1

Due: 23:59, Mar. 6, 2023

1. Submit your solutions to the course Blackboard.
2. If you want to submit a handwritten version, scan it clearly.
3. Late homeworks submitted within 24 hours of the due date will be marked down 25%. Homeworks submitted more than 24 hours after the due date will not be accepted unless there is a valid reason, such as a medical or family emergency.
4. You are required to follow ShanghaiTech's academic honesty policies. You are allowed to discuss problems with other students, but you must write up your solutions by yourselves. You are not allowed to copy materials from other students or from online or published resources. Violating academic honesty can result in serious penalties.

Problem 1:

Assume you have functions f and g such that $f(n) = O(g(n))$. For each of the following statements, decide whether you think it is true or false and give a proof or counterexample.

a) $\log_2 f(n) = O(\log_2 g(n))$

b) $2^{f(n)} = O(2^{g(n)})$

c) $f(n)^2 = O(g(n)^2)$

Solution:

a) False: When $g(n) = 1$ for all n, $f(n) = 2$ for all n, then $\log_2 g(n) = 0$, whence we cannot write $\log_2 f(n) \leq c \log_2 g(n)$ for any constant $c > 0$.
On the other hand, if we simply require $g(n) > 2$ for all n beyond some n_1, then the statement holds. Since $f(n) \leq cg(n)$ for all $n \geq n_0$, we have $\log_2 f(n) \leq \log_2 g(n) + \log_2 c \leq (\log_2 c) \log_2 g(n)$ once $n \geq \max(n_0, n_1)$.

b) False: take $f(n) = 2n$ and $g(n) = n$. Then $2^{f(n)} = 4^n$, while $2^{g(n)} = 2^n$.

c) True: Since $f(n) \leq cg(n)$ for all $n \geq n_0$, we have $(f(n))^2 \leq c^2(g(n))^2$ for all $n \geq n_0$.

Problem 2:

Sort the following functions in ascending order of growth.

$$f_1(n) = (\log n)^{\log n} \tag{1}$$
$$f_2(n) = n^2 \log n \tag{2}$$
$$f_3(n) = 2023^{\sqrt{n}} \tag{3}$$
$$f_4(n) = n^{987} \tag{4}$$
$$f_5(n) = 2^n \tag{5}$$
$$f_6(n) = (\log n)^n \tag{6}$$
$$f_7(n) = 16^{42^{223}} \tag{7}$$
$$f_8(n) = n \tag{8}$$
$$f_9(n) = n^{\sqrt{n}} \tag{9}$$

Solution:

7, 8, 2, 4, 1, 3, 9, 5, 6

Problem 3:

Consider the following method for sorting N elements stored in an array A, called *selection sort*. First, find the smallest element in A and swap it with element $A[1]$. Next, find the second smallest element in A and swap it with the element $A[2]$. Continue this way for the first $N-1$ elements in A. Analyze its best and worst case time complexity using big-O notation.

Solution:

The running time of the algorithm is $\Theta\left(n^2\right)$ for all cases.

```
n = A.length
for j = 1 to n - 1
    smallest = j
    for i = j + 1 to n
        if A[i] < A[smallest]
            smallest = i
    exchange A[j] with A[smallest]
```

Problem 4:

Suppose you have n identical-looking cards. Each card has an ID, which you cannot see, and there may be multiple cards with the same ID. You are given a machine which can take two cards at a time and determine whether the cards have the same ID.

Give an algorithm which uses the machine $O(n \log n)$ times, and determines whether there exists an ID which occurs on more than $\lfloor n/2 \rfloor$ cards. For example, if the cards have IDs $\{1, 2, 2, 2\}$, then the algorithm should return yes, while for $\{1, 2, 3, 3\}$ it should return no. You can assume that n is a power of 2.

Solution:

Let $c_1, \ldots, c_n$ denote the IDs of the cards: cards i and j are equivalent if $c_i = c_j$. We are looking for a value x so that more than $\lfloor n/2 \rfloor$ of the indices have $c_i = x$.

Divide the set of cards into two roughly equal piles: a set of $\lfloor n/2 \rfloor$ cards and a second set for the remaining $\lceil n/2 \rceil$ cards. We will recursively run the algorithm on the two sides, and will assume that if the algorithm finds an equivalence class containing more than half of the cards, then it returns a sample card in the equivalence class.

Note that if there are more than $n/2$ cards that are equivalent in the whole set, say have equivalence class x, then at least one of the two sides will have more than half the cards also equivalent to x. So at least one of the two recursive calls will return a card that has equivalence class x.

To analyze the running time, let $T(n)$ denote the maximum number of tests the algorithm does for any set of n cards. The algorithm has two recursive calls, and does at most $2n$ tests outside of the recursive calls. So we get the following recurrence:

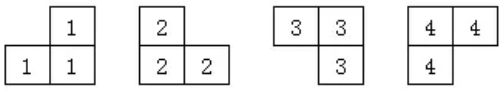
$$T(n) \leq 2T(n/2) + 2n$$

The recurrence implies that $T(n) = O(n \log n)$.

Problem 5:

Jack wants to use carpets to cover the floor of the Student Activity Center. However, he can only find small carpets with certain special shapes. He also needs to leave a given cell of the floor uncovered, which will be used for placing a bulletin board.

- The floor of the Student Activity Center is a square shape where each side has length n, giving a total area of n^2 units. You can assume $n = 2^k$ for some $k \geq 2$.
- There are four shapes of carpets, and each covers three unit cells of the floor. The shapes are shown below.
- The bulletin board will be put in one cell of the floor, and takes 1 unit of area.
- All other cells without the bulletin board need to be covered by exactly one carpet. In another word, two carpets cannot overlap.



As simple example, suppose the floor is a 2×2 square, and the bulletin board is placed in cell $(0, 0)$. Then we can use one carpet with shape 1 to cover the floor.

Given an algorithm for Jack to solve the problem with an $n \times n$ floor, where the bulletin board is placed in cell (x, y), for $0 \leq x, y \leq n - 1$. Clearly describe your algorithm and why it is correct, and use pseudocode if necessary.

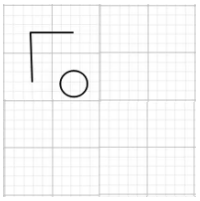
Hint: Using divide-and-conquer, and consider the problem for a 4×4 floor first.

Solution:

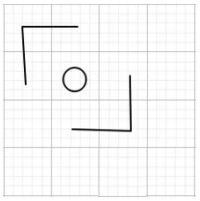
Consider the basic issue, when $n = 2$, no matter where the bulletin board is, we can use one carpet to cover the left area. Which means, when we have exactly one obstacle, the 2×2 area can be solved.

Now consider a 4×4 case with a quarter of its area being covered already. We

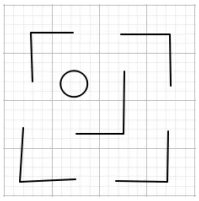
put an extra carpet on its center and the left part becomes 3 sub-part of 2×2 area with an obstacle. Then we fulfill it to get a general solution for 4×4 issues.



(a) 4×4 issue

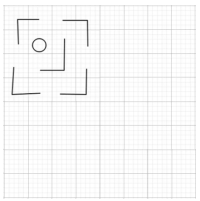


(b) Adding a carpet.

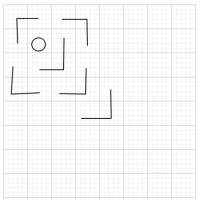


(c) Full-filling.

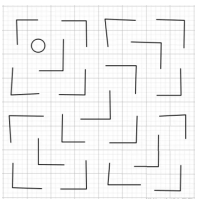
Similarly we get the result of 8×8 by adding an extra carpet at the center. By applying this methodology all $n = 2^k$ issues could be done by merging 4 quarter-problems together with an extra carpet at the center.



(d) 8×8 issue



(e) Adding a carpet.



(f) Full-filling.

Problem 6:

Suppose you have n identical-looking integrated-circuit chips, each of which may be good or bad (i.e. work correctly or not). To test the chips, you have a machine where you can place two chips at a time, and then each chip reports whether it thinks the other chip is good or bad. A good chip always accurately reports whether the other chip is good or bad, but a bad chip can give any answer. Thus, the four possible outcomes each time you test a pair of chips are as follows:

Chip A says	Chip B says	Conclusion
B is good	A is good	both are good, or both are bad
B is good	A is bad	at least one is bad
B is bad	A is good	at least one is bad
B is bad	A is bad	at least one is bad

Give an algorithm to find a single good chip from among the n chips, assuming that more than $\lfloor n/2 \rfloor$ of the chips are good. Use as few tests as you can.

Hint: Show that $\lceil n/2 \rceil$ tests are sufficient to reduce the problem to one of at most half the size.

Solution:

We start with n chips and know that there are more good chips than bad chips. Divide all the chips into [n/2] pairs, with 1 chip set aside in the event that n is odd. For each of the pairwise tests, if either chip reports the other is bad, discard both chips. If both chips report each other as good, discard one chip. Combine the set aside chip (if there was one) with all remaining chips and repeat this process. You will eventually be left with one chip and it is guaranteed that that chip is good.

At each point when you are discarding one or more chips, the property that there are more good chips than bad chips is always held. When you discard both chips, the table above tells us that at least one of them was bad. Discarding one chip from a pair that reports both are good could result in both actually being good, but this would always be offset by the discarding of bad chips from other pairs. In a similar way, this property guarantees that we will never reach a situation wherein we discard all remaining chips unless there is one set aside. This algorithm will continue until there is only one chip remaining, and when

this happens the property that there are more good chips than bad chips will still hold, and thus that final remaining chip must be a good chip.