CSE 21 Fall 2018
Homework 3
Due: Sunday, October 21, 2018 at 10:00pm

Instructions

Homework should be done in groups of **one to three** people. You are free to change group members at any time throughout the quarter. Problems should be solved together, not divided up between partners. Homework must be submitted through **Gradescope** by **a single representative**. Submissions must be received by **10:00pm** on the due date, and there are no exceptions to this rule.

You will be able to look at your scanned work before submitting it. Please ensure that your submission is **legible** (neatly written and not too faint) or your homework may not be graded.

Students should consult their textbook, class notes, lecture slides, instructors, TAs, and tutors when they need help with homework. Students should not look for answers to homework problems in other texts or sources, including the Internet. You may ask questions about the homework in office hours, but **not on Piazza**.

Your assignments in this class will be evaluated not only on the correctness of your answers, but on your ability to present your ideas clearly and logically. You should **always explain** how you arrived at your conclusions and **justify your answers** with mathematically sound reasoning. Whether you use formal proof techniques or write a more informal argument for why something is true, your answers should always be well-supported. Your goal should be to **convince the reader** that your results and methods are sound.

For questions that require pseudocode, you can follow the same format as the textbook, or you can write pseudocode in your own style, as long as you specify what your notation means. For example, are you using "=" to mean assignment or to check equality? You are welcome to use any algorithm from class as a subroutine in your pseudocode. For example, if you want to sort list A using InsertionSort, you can call InsertionSort(A) instead of writing out the pseudocode for InsertionSort.

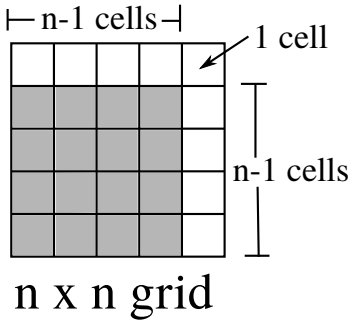
Required Reading Rosen Chapter 5 and Sections 8.1-8.3.

Key Concepts: Recursive algorithms (definitions, proofs of correctness, runtime calculations), recursive structure of a problem, recurrence relations, guess-and-check method for solving recurrences, unraveling recurrences, Master Theorem.

---

1. (16 points total - 2 points each) In each situation, write a recurrence relation, including base case(s), that describes the recursive structure of the problem. Briefly explain how you obtain your recurrence. You do not need to solve for the closed-form formula.

   (a.) Let $C(n)$ be the number of 1×1 cells in an $n \times n$ grid. Write a recurrence for $C(n)$.
   Solution: We can think of each $n \times n$ grid as containing an $(n-1) \times (n-1)$ grid in the lower left corner. The larger $n \times n$ grid contains all the cells of the $(n-1) \times (n-1)$ grid plus an extra strip around the outside which contains $(n-1)+1+(n-1) = 2n-1$ additional cells.



   Therefore, the recurrence is

$$C(n) = C(n-1) + 2n - 1, \text{ with } C(1) = 1$$

   or equivalently,

$$C(n) = C(n-1) + 2n - 1, \text{ with } C(0) = 0.$$

   (b.) A bunch of motorcycles and cars want to parallel park on a street. The street can fit n motorcycles, but cars take up three motorcycle spaces. Let $A(n)$ be the number of arrangements of cars and motorcycles on a street that fits n motorcycles. For example, $A(5) = 4$ because there are four ways to park vehicles on a street with five motorcycle spaces. If M stands for motorcycle, and C stands for car, then the four arrangements are: MMC, MCM, CMM, and MMMMM. Write a recurrence for $A(n)$.
   Solution: If the first vehicle parked is a motorcycle, then the other vehicles form an arrangement that fills $n-1$ spots. If the first vehicle parked is a car, then the other vehicles form an arrangement that fills $n-3$ spots. Therefore, the recurrence is

$$A(n) = A(n-1) + A(n-3), \text{ with } A(0) = 1, A(1) = 1, A(2) = 1$$

   or equivalently,

$$A(n) = A(n-1) + A(n-3), \text{ with } A(1) = 1, A(2) = 1, A(3) = 2.$$

   (c.) Let $B(n)$ be the number of length n binary strings that have no three consecutive 0's. Write a recurrence for $B(n)$.
   Solution: Any bit string that has no 000 must have a 1 in at least one of the first three positions. Break up all bit strings avoiding 000 by when the first 1 occurs. That is, each bit string of length n avoiding 000 falls into exactly one of these cases:
   (i) 1 followed by any bit string of length $n-1$ avoiding 000.
   (ii) 01 followed by any bit string of length $n-2$ avoiding 000.

(iii) 001 followed by any bit string of length $n-3$ avoiding 000.

Therefore, the recurrence is

$$B(n) = B(n-1) + B(n-2) + B(n-3), \text{ with } B(0) = 1, B(1) = 2, B(2) = 4$$

or equivalently,

$$B(n) = B(n-1) + B(n-2) + B(n-3), \text{ with } B(1) = 2, B(2) = 4, B(3) = 7.$$

(d.) Any binary string can be broken into contiguous chunks of the same character, called runs. For example, 001110100000 has:

- a run of 0s of length 2
- a run of 1s of length 3
- a run of 0s of length 1
- a run of 1s of length 1
- a run of 0s of length 5

Let $B(n)$ be the number of length n binary strings where each run of 1s has even length. Find a recurrence for $B(n)$.

Solution: Every binary string either starts with 0 or 1. Suppose we have a binary string where each run of 1s has even length, and the string starts with 0. Then the remainder of the string is a length $n-1$ binary string where each run of 1s has even length, and there are $B(n-1)$ such strings. Now, suppose we have a binary string where each run of 1s has even length, and the string starts with 1. Then the second bit of the string must also be 1, otherwise the string would start with a run of length one, which is odd. After these first two 1s, the remainder of the string is a length $n-2$ binary string where each run of 1s has even length, and there are $B(n-2)$ such strings. Therefore, the recurrence is

$$B(n) = B(n-1) + B(n-2), \text{ with } B(0) = 1, B(1) = 1$$

or equivalently,

$$B(n) = B(n-1) + B(n-2), \text{ with } B(1) = 1, B(2) = 2.$$

(e.) Let $S(n)$ be the number of subsets of $\{1, 2, \ldots, n\}$ having the following property: no two elements in the subset are consecutive integers. The empty set with no elements should be included in your count. Write a recurrence for $S(n)$.

Solution: If the subset contains 1, then the rest of the subset should be a subset of $\{3, 4, \ldots, n\}$ with the property that no two elements are consecutive. There are $S(n-2)$ such subsets because if we think of just subtracting two from each of the integers, it is the same problem as before with a set of size $n-2$. If the subset does not contain 1, then the rest of the subset should be a subset of $\{2, 3, \ldots, n\}$ with the property that no two elements are consecutive. There are similarly $S(n-1)$ such subsets. Therefore, the recurrence is

$$S(n) = S(n-1) + S(n-2), \text{ with } S(0) = 1, S(1) = 2$$

or equivalently,

$$S(n) = S(n-1) + S(n-2), \text{ with } S(1) = 2, S(2) = 3.$$

(f.) A ternary string is like a binary string except it uses three symbols, 0, 1, and 2. For example, 12210021 is a ternary string of length 8. Let $T(n)$ be the number of ternary strings of length n with the property that there is never a 2 appearing **anywhere** after a 0. For example, 12120110 has this property but 10120012 does not. Write a recurrence for $T(n)$.

Solution: Any ternary string of length n can start with 0, 1, or 2. There are three cases for the first letter in the string:

- If the first letter is 0, then since we cannot have 2 anywhere after a 0, the remaining letters form a binary string, that is a string of length $n-1$ containing all 0s and 1s. The contribution of this case to $T(n)$ is 2^{n-1}.
- If the first letter is either 1 or 2 then the remaining of the string can be any ternary string of length $n-1$ having the property that there is never a 2 anywhere after a 0. The contribution of this case to $T(n)$ is $2T(n-1)$.

For the base case, note that any ternary string of length one satisfies the required property.

Therefore, our recurrence is: $T(n) = 2T(n-1) + 2^{n-1},\ T(1) = 3$.

Note: For parts (g) and (h) of this problem, to "tile a rectangle" is to cover it with tiles so that no tiles overlap, no tiles are hanging off the edge of the rectangle, and every space on the rectangle is covered by some tile.

(g.) Let $T(n)$ be the number of ways to tile a $1 \times n$ rectangle with 1×1 square tiles that are either red or blue, and 1×2 domino tiles that are green, white, or yellow. Write a recurrence for $T(n)$.

Solution: When $n = 0$, there is only one tiling, the empty tiling, and thus, $T(0) = 1$.

When $n = 1$, we can only tile a a 1×1 rectangle with a 1 square tile. However, there are two ways to choose a color for this tile. Thus, $T(1) = 2$.

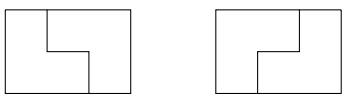
For $n \geq 2$, consider a tile placement counted by $T(n)$. We shall classify this tiling based on the first tile used, when reading the rectangle from left to right. There are two possibilities.

- If the first tile is a 1×1 square, then there are **two** ways to pick a color for this square tile. For each of the color choice, the remaining cells can be any tiling of a $1 \times (n-1)$ rectangle with colored squares and dominoes. Thus, the contribution in this case to $T(n)$ is given by $2T(n-1)$.
- If the first tile is a 1×2 domino, then there are **three** ways to pick a color for this domino tile. For each of the color choice, the remaining cells can be any tiling of a $1 \times (n-2)$ rectangle with colored squares and dominoes. Thus, the contribution in this case to $T(n)$ is given by $3T(n-2)$.

Therefore, the recurrence is $T(n) = 2T(n-1) + 3T(n-2)$, with $T(0) = 1, T(1) = 2$.

(h.) Let $L(n)$ be the number of ways to tile a $2 \times n$ rectangle with L-shaped tiles of area 3 (trominoes). Write a recurrence for $L(n)$.

Solution: First, notice that if n is not a multiple of 3, there will be no way to tile the rectangle. Now if n is a multiple of 3, then there are two ways to tile the first three columns:



The rest of the tiling is a tiling of a $2 \times (n-3)$ rectangle, of which there are $L(n-3)$. Therefore, the recurrence is

$$L(n) = \begin{cases} 1 & \text{if } n = 0 \\ 2L(n-3), \text{with} & \text{if } n \equiv 0 \mod 3 \\ 0 & \text{otherwise.} \end{cases}$$

Equivalently, this is the recurrence

$$L(n) = 2L(n-3), \text{ with } L(0) = 1, L(1) = 0, L(2) = 0.$$

2. (5 points) At a Phi Beta Kappa party with n people, everyone high-fives once with everybody else. Let $H(n)$ be the total number of high-fives among the n people.

   (a.) Write a recurrence for $H(n)$.
   Solution: Since each person high-fives with $n-1$ other people, the recurrence is

   $$H(n) = H(n-1) + (n-1), \text{ with } H(2) = 1$$

   or equivalently,

   $$H(n) = H(n-1) + (n-1), \text{ with } H(1) = 0.$$

   (b.) Use the guess-and-check method to obtain a closed-form formula for $H(n)$. That is, first guess a formula for $H(n)$ and use induction to prove that your guess is correct.
   Solution: The first few values of $H(n)$ is given by

n	1	2	3	4	5	6	7
H(n)	0	1	3	6	10	15	21

   The table above suggests that a formula for $H(n)$ may be $H(n) = \dfrac{n(n-1)}{2}$. We shall prove this guess by induction on n.

   The base case is when $n = 1$. We have $H(1) = 0$ and $\dfrac{1(1-1)}{2} = 0$, so the formula holds.

   Assume that $H(n) = \dfrac{n(n-1)}{2}$ as our inductive hypothesis. Then $H(n+1) = H(n) + n$ by the recurrence found in part (a), and so

   $$H(n+1) = \frac{n(n-1)}{2} + n = \frac{n(n-1) + 2n}{2} = \frac{n^2 + n}{2} = \frac{(n+1)(n)}{2}$$

   by applying the inductive hypothesis and simplifying. Thus, our formula is correct.

3. Use unraveling method to obtain a closed-form formula for each of the following recurrences. Your final answer should be a very simple formula involving n. For part (c), you may want to use a formula from calculus for the sum of a finite geometric series.

(a) (5 points) $R(n) = \dfrac{n}{n-2}R(n-2),\ R(1) = 3,\ R(2) = 6.$

Solution:

$$\begin{aligned} R(n) &= \frac{n}{n-2}R(n-2) \\ &= \frac{n}{n-2}\left(\frac{n-2}{n-4}R(n-4)\right) = \frac{n}{n-4}R(n-4) \\ &= \frac{n}{n-4}\left(\frac{n-4}{n-6}R(n-6)\right) = \frac{n}{n-6}R(n-6) \\ &= \cdots = \frac{n}{n-k}R(n-k) \end{aligned}$$

There are two base cases, and as we unravel, we will hit $R(1)$ or $R(2)$ based on whether n is odd or even, since we subtract 2 each time. When n is odd, let $k = (n-1)$, so

$$R(n) = \frac{n}{n-k}R(n-k) = \frac{n}{1}R(1) = 3n.$$

When n is even, let $k = (n-2)$, so

$$R(n) = \frac{n}{n-k}R(n-k) = \frac{n}{2}R(2) = n/2 \cdot 6 = 3n.$$

Hence, $R(n) = 3n$ for all positive integers n.

(b) (5 points) $S(n) = S(n-1) + 2n + 1,\ S(1) = 3$

Solution:

$$\begin{aligned} S(n) &= S(n-1) + 2n + 1 \\ &= (S(n-2) + 2(n-1) + 1) + 2n + 1 \\ &= [(S(n-3) + 2(n-2) + 1) + 2(n-1) + 1] + 2n + 1 \\ &\ \ \vdots \\ &= S(n-k) + 2\left((n-k+1) + \cdots + (n-1) + n\right) + k \\ &\ \ \vdots \\ &= S(1) + 2\left(2 + \cdots + (n-1) + n\right) + (n-1) \\ &= 3 + 2\left(n(n+1)/2 - 1\right) + (n-1) \\ &= 3 + n(n+1) - 2 + n - 1 \\ &= n(n+2) \end{aligned}$$

Here we are letting $k = n-1$ because that is the value of k that reaches the base case $S(1) = 3$.

(c) (5 points) $T(n) = 3T(n-1) + 8,\ T(0) = 2$

Solution:

$$\begin{aligned} T(n) &= 3T(n-1) + 8 \\ &= 3(3T(n-2) + 8) + 8 \\ &= 3^2T(n-2) + 3(8) + 8 \\ &= 3^2(3T(n-3) + 8) + 3(8) + 8 \\ &= 3^3T(n-3) + 3^2(8) + 3(8) + 8 \\ &= 3^kT(n-k) + (3^{k-1} + 3^{k-2} + \cdots + 3^0)(8) \end{aligned}$$

Here we can apply the formula for the finite geometric series $\displaystyle\sum_{r=0}^{k-1} 3^r = \frac{1-3^k}{1-3} = \frac{3^k-1}{2}$ so that

$$\begin{aligned} T(n) &= 3^kT(n-k) + \frac{3^k-1}{2}(8) \\ &= 3^kT(n-k) + 4(3^k-1) \\ &= 3^nT(0) + 4(3^n-1) \qquad \text{(letting } k = n\text{)} \\ &= 3^n \cdot 2 + 4 \cdot 3^n - 4 \\ &= 6 \cdot 3^n - 4 \end{aligned}$$

4. (5 points) The following algorithm **has access to a global list of distinct integers** $a_1, a_2, \ldots, a_n$. When called with parameters i, j, x, Search(i, j, x) returns the location of the target value x among $\{a_i, \ldots, a_j\}$, or 0 if the target value is not present in $\{a_i, \ldots, a_j\}$.

procedure Search$(i, j, x : i, j, x$ integers, $1 \le i \le j \le n)$

1. **if** $a_i = x$ **then**
2. **return** i
3. **else if** $i = j$ **then**
4. **return** 0
5. **else**
6. **return** Search$(i + 1, j, x)$

Let $T(n)$ be the running time of this algorithm. Write a recurrence relation that $T(n)$ satisfies. Then solve the recurrence and write the solution in Θ notation.

Solution: We start with a list of size n and with each recursive call, we search a list of size one smaller than before. All the other work in the algorithm is just basic operations, which take a constant amount of time. In the base case, which is for a list of size 1, we also do a constant amount of work in lines 1 through 4. Therefore, the recurrence is

$$T(n) = T(n-1) + c, \text{ with } T(1) = d,$$

where c and d are constants. Note that we cannot determine exactly how much time (in seconds or minutes) these basic operations take, so c and d must be arbitrary constants.

We will use the unravel method since we don't have particular constants to guess a pattern.

$$\begin{aligned} T(n) &= T(n-1) + c \\ &= T(n-2) + c + c \\ &= T(n-3) + c + c + c \\ &\vdots \\ &= T(n-k) + ck \\ &\vdots \\ &= T(1) + c(n-1) \\ &= d + c(n-1) \\ &= cn + (d-c) \end{aligned}$$

Here we are letting $k = n-1$ because that is the value of k that reaches the base case $T(1) = d$. In Θ notation, we have $\Theta(cn + (d-c)) = \Theta(n)$ since c and d are constants. Therefore, this algorithm is linear in the size of the list we are searching.

5. The following algorithm determines whether a word is a palindrome, that is, if the word is the same read left to right as right to left. An example of a palindrome is *racecar*.

   procedure Palindrome($s_1 s_2 s_3 \dots s_n$)

   1. **if** ($n = 0$ or $n = 1$) **then return** $true$
   2. **if** $s_1 = s_n$ **then return** Palindrome($s_2 \dots s_{n-1}$)
   3. **else return** $false$

   Note: Writing $s_1 s_2 s_3 \dots s_n$ denotes a string of length n whose characters are s_1, s_2, s_3, etc. These characters are being concatenated (not multiplied) to form a string.

   (a) (5 points) Prove that this algorithm is correct, i.e., that it returns true if and only if $s_1 s_2 s_3 \dots s_n$ is a palindrome.

   Solution: Here we will use strong induction on n, the length of the input word.

   For the base cases, when $n = 0$ or $n = 1$, all words of length 0 or 1 are palindromes, so the algorithm is correct since it returns $true$ in these cases.

   Now suppose that $n \geq 2$ and the algorithm is correct on all input words of length $1, 2, \dots, n-1$. This is our strong inductive hypothesis. We must show that the algorithm is correct on input words of length n.

   If the algorithm takes as input a word $s_1 s_2 s_3 \dots s_n$ of length n, $n \geq 2$, where the first and last letters are not the same, then the word cannot be a palindrome and so the algorithm correctly returns $false$.

   If instead, the first and last letters are the same, then the word is a palindrome if and only if when we remove the first and last letters, we still have a palindrome. Thus, $s_1 s_2 s_3 \dots s_n$ is a palindrome if and only if $s_2 s_3 \dots s_{n-1}$ is also a palindrome.

Since the strong inductive hypothesis says that Palindrome will return the correct answer on input $s_2 s_3 \ldots s_{n-1}$ (which is of size $n-2$), and the algorithm simply returns that answer, the algorithm will be correct on input words of length n.

(b) (2 points) Let $C(n)$ be the number of times this algorithm compares two characters s_i and s_j for some i, j. Write a recurrence relation that $C(n)$ satisfies.

Solution: With each recursive call, one comparison is done in line 2 (between the first and last characters of the word), and then Palindrome is called again with an input having length two smaller than before. No comparisons are done in the base cases in line 1. Therefore, the recurrence is

$$C(n) = C(n-2) + 1, \text{ with } C(0) = 0, C(1) = 0.$$

(c) (3 points) Solve the recurrence found in part (b) to get a closed-form formula for $C(n)$.

Solution: To find an exact formula for $C(n)$ we will have to consider whether n is even or odd, since the recursion expresses $C(n)$ in terms of $C(n-2)$. This means that when n is even, we will eventually hit the base case for $n = 0$ and when n is odd, we will eventually hit the base case for $n = 1$. Unraveling the recurrence gives

$$\begin{aligned}
C(n) &= C(n-2) + 1 \\
&= C(n-4) + 2 \\
&= C(n-6) + 3 \\
&\vdots \\
&= C(n-k) + \frac{k}{2} \\
&\vdots \\
&= \begin{cases} C(0) + \frac{n}{2} & \text{if } n \text{ is even} \quad \text{(letting k=n)} \\ C(1) + \frac{n-1}{2} & \text{if } n \text{ is odd} \quad \text{(letting k=n-1)} \end{cases} \\
&= \begin{cases} \frac{n}{2} & \text{if } n \text{ is even} \\ \frac{n-1}{2} & \text{if } n \text{ is odd} \end{cases} \\
&= \left\lfloor \frac{n}{2} \right\rfloor
\end{aligned}$$

6. The following algorithm, CountMultiples, takes as input a list of $n \geq 1$ distinct positive integers and returns the number of list elements that are a multiple of some prior list element. For example, on the input $2, 6, 5, 9, 8, 3, 15$, the algorithm would return 3 because there are 3 list elements that are a multiple of some prior list element:

- 6 is a multiple of 2
- 8 is a multiple of 2
- 15 is a multiple of 5 (and it is a multiple of 3)

procedure CountMultiples($a_1, \ldots, a_n$: $n \geq 1$)

1. **if** $n = 1$ **then return** 0

2. $m = \text{CountMultiples}(a_1, \ldots, a_{n-1})$
3. **for** $i = 1$ **to** $n - 1$
4. **if** a_i divides a_n **then return** $m + 1$
5. **return** m

(a) (5 points) Prove that CountMultiples is correct.

Proof: We can prove that CountMultiples is correct for any input of size n, using (regular) induction on n. For the base case, when $n = 1$, CountMultiples returns 0 in line 1. This is correct because when there is only one list element, it cannot be a multiple of anything before it, as there is nothing before it. So there are 0 list elements that are a multiple of some prior list element.

Now suppose as the inductive hypothesis that CountMultiples is correct on all inputs of size $n - 1$ and we will show that it is correct on all inputs of size n, where $n > 1$. Let $a_1, \ldots, a_n$ be an arbitrary input of size n.

On this input, CountMultiples will not return in the first line, since we are assuming $n > 1$. Then it will call CountMultiples$(a_1, \ldots, a_{n-1})$ and store the result as m. Since $a_1, \ldots, a_{n-1}$ is a list of size $n - 1$, we know by the induction hypothesis that m will be the number of elements in $a_1, \ldots, a_{n-1}$ that are a multiple of some prior list element. Notice that the total number of list elements in $a_1, \ldots, a_n$ that are a multiple of some prior element is the total number of such elements in the list $a_1, \ldots, a_{n-1}$ plus possibly one more, depending on whether the last element a_n is a multiple of some prior element. In the case where the last element a_n is a multiple of some prior list element a_i with $i < n$, then the algorithm will return $m + 1$ because it loops over all values of $i < n$. This is the correct result in this case. Similarly, if the last element a_n is not a multiple of some prior list element, then the entire for loop will execute without the if condition being true, so the algorithm will return m in the very last line, which is also the correct result in this case.

Therefore, since we have shown that for an arbitrary input of size n, CountMultiples returns the number of elements in the input list that are a multiple of some prior list element, we conclude that CountMultiples is correct by induction.

(b) (3 points) Give an example of a best-case input of size $n = 6$. Write a recurrence for the runtime $T(n)$ in the best case. What's the order of the algorithm in Θ notation, in the best case?

Solution: A best-case input is one where the first element is a 1, for example $1, 4, 2, 5, 7, 8$. In this case, the for loop only does one iteration each time it is called, since $a_1 = 1$ is a multiple of each other element. Therefore, on an input of size n, the runtime of the recursive call is $T(n - 1)$ and the runtime of the for loop and other work outside the recursive call is constant-time. Therefore, a recurrence is

$$T(n) = T(n - 1) + c, T(1) = d, \text{ for constants } c, d$$

In this case, the algorithm's runtime is $\Theta(n)$, as this recurrence describes a linear function.

(c) (3 points) Give an example of a worst-case input of size $n = 6$. Write a recurrence for the runtime $T(n)$ in the worst case. What's the order of the algorithm in Θ notation, in the worst case?

Solution: A best-case input is one where no element is a multiple of any prior element, for example $7, 5, 9, 3, 8, 1$. In this case, the for loop does a full $n-1$ iterations each time it is called on an input of size $n-1$, since the if condition will never be true. Therefore, on an input of size n, the runtime of the recursive call is $T(n-1)$ and the runtime of the for loop is linear. Therefore, a recurrence is

$$T(n) = T(n-1) + cn, T(1) = d, \text{ for constants } c, d$$

In this case, the algorithm's runtime is $\Theta(n^2)$, as this recurrence describes a quadratic function.

7. Let n be a positive integer. Use the Master Theorem to obtain the big-O class for the functions that satisfy the following recurrences.

   (a) (4 points) $g(n) = 4g(n/2) + n^2$

   Solution: We can directly apply the master theorem with $a = 4, b = 2$, and $d = 2$ (since $n^2 \in O(n^2)$) . Since $a = 4 = 2^2 = b^d$, we are in the "steady-state" case, and $T(n) \in O(n^d \log n) = O(n^2 \log n)$.

   (b) (4 points) $f(n) = 2f(n/3) + O(n)$.

   Solution: In this case, $a = 2, b = 3$ and $d = 1$. Since $a = 2 < 3^1 = b^d$, we are in the "top-heavy" case, and the function $f(n) \in O(n^d) = O(n)$.