

Analysis of Algorithms
CS 376, Spring 2023

Problem Set 2

Due **AT THE BEGINNING OF CLASS** Wednesday, March 15

- From CLRS, please read Chapter 4 (4.1 is skippable).
- The style guidelines for writing algorithms given on PS0 continue to apply. In general, unless explicitly instructed otherwise, these guidelines apply to all exercises in CS376 (Problem Sets, Smaller Assignments, ...) whether or not they explicitly appear as part of the assignment itself.
- For this assignment and future CS376 assignments, there may be small deductions for excessively inelegant solutions (e.g., algos, proofs) to exercises. Answers *absolutely do not need to be optimally elegant for full credit*, but when an excessively inelegant answer misses some of the central concepts of a clean solution, there is room for improvement to the answer, which may entail a small deduction. Please feel free to ask me questions about this and how it might apply to your specific HW solutions!
- *A general note for CS376:* When writing up your homework, please write neatly and **explain your answers clearly**, giving all details needed to make your answers easy to understand. Clear communication *is* the point, on every assignment.

In general in CS376, unless explicitly specified otherwise, answers should be accompanied by explanations; as part of this, in general, when you present an algorithm, give an accompanying correctness argument (i.e., proof). Answers without explanations may not receive full credit. Please feel free to ask me any questions about explanations that might come up!

Exercises

1. Using the *unwinding* method from class, solve the following recurrence relations and give the Θ class of the solution.

(a) $T(n) = T(n - 1) + n$ for $n > 0$; $T(0) = 0$.

(b) $T(n) = T(n/3) + 1$ for $n > 1$; $T(1) = 1$. (Assume n is of the form 3^i for this exercise.)

Please show your work in doing the “unwinding”: To show a full understanding of the unwinding method, please be sure to show a k ’th step—a step at some representative k ’th step in the unwinding, in terms of k rather than some specific number, as demonstrated in lecture and shown in lecture notes—which shows the pattern upon which your solution is based.

2. Solve the following recurrence **showing your work, using the recursion tree method** and give the Θ class of the solution.

$$T(n) = 3T(n - 1) \text{ for } n > 1; T(1) = 4.$$

Please show your work by including a table of the following form. The table should include one row for each level in the tree; be sure to show a representative k 'th level to illustrate the relevant pattern, similar to the k 'th step in the unwinding method.

level	# nodes	work per node	total work
0	nodes on level 0	work per node on level 0	total work on level 0
1	nodes on level 1	work per node on level 1	total work on level 1
2	nodes on level 2	work per node on level 2	total work on level 2
...	...	...	...
k	nodes on level k	work per node on level k	total work on level k
...	...	...	...
$\langle last\ level \rangle$	nodes on $\langle last\ level \rangle$	work per node on $\langle last\ level \rangle$	total work on $\langle last\ level \rangle$

Then, explain how you use information from your table to arrive at a final answer. Recall that you can use high-level explanations for Θ analysis; you don't need to give witnesses for the values of the relevant constants.

NOTES: A table like this is generally a good way to show your work when using the recursion tree method; sometimes, you might want to explicitly include the second-to-last level in your table, as well as the last level, because the last level is sometimes so different from the remainder of the table that it does not fit into the pattern. Also, you are also welcome to include a picture of the recursion tree as part of your answer if you'd like, but it is not required for the exercise.

L^AT_EX Note: There are, I'm sure, multiple ways to get a table like the one for this exercise using L^AT_EX, but I used the `tabular` environment. For instance, the header of the above table would be generated by:

```
\begin{tabular}{c c c c}
  level & \# nodes & work per node & total work\\
  \hline\\
\end{tabular}
```

Note that the `&` symbols are column delimiters. Please look up the `tabular` environment for more information (the first parts of the tutorial at <https://www1.cmc.edu/pages/faculty/aaksoy/latex/latexfour.html> or the very beginning of <https://www.tug.org/TUGboat/tb28-3/tb90hoeppner.pdf> may be all you need to get off to a great start), and if you still have questions, as always, feel free to ask!

3. Give Θ bounds for the following recurrences. *Be sure to use the Master Theorem for these exercises.* As always, be sure to give a brief explanation of your answers—here, that will include the values for each relevant variable in the Master Theorem, what case of the Theorem you are applying, and a brief explanation of how you know what case to apply.

(a) $T(n) = 4T(n/2) + n^2$, $T(1) = 1$.

(b) $T(n) = 4T(n/2) + n^3$, $T(1) = 1$.

4. Give a Θ bound for the following recurrence, using the method of your choice from among these three: unwinding, recursion tree, Master method.

$$T(n) = 3T(n/2) + n \lg n, T(1) = 1$$

As always, be sure to give a brief explanation (with appropriate details) of your answer.

5. An algorithm to compute 2^n would be defined by the following specifications:

```
// Input: n, a non-negative integer (i.e., 0 or greater)
// Output: 2^n (i.e., 2 to the n'th power)
```

- (a) Write a recursive algorithm for computing 2^n for any non-negative integer n **that is based on the formula** $2^n = 2^{n-1} + 2^{n-1}$. (This may not be the most natural way to think of an algorithm to compute 2^n , but please be sure to use it for this exercise!)
- (b) Set up a recurrence for the *number of additions made by the algorithm* on input n and solve it **using the recursion tree method**.
As above, please show your work by including a table of the form illustrated in the previous exercise; once again, the table should include one row for each level in the tree and a representative k 'th level to illustrate the relevant pattern, similar to the k 'th step in the unwinding method. Then, explain how you use information from your table to arrive at a final answer. (You are also welcome to include a picture of the recursion tree as part of your answer, if you'd like.)
- (c) What is the Θ complexity class of this algorithm? If the above recurrence is helpful in this, explain why; if not, set up a different recurrence for the runtime of this algorithm and solve it **using a method of your choice** to get the complexity class.
- (d) Is this a good algorithm for computing 2^n ? Please explain your answer! (A couple of sentences could well be a sufficient explanation for this exercise.)
6. Prof. Nigma at the Portland Institute of Technology (which continues to not exist!; motto: "What's an acronym?") hired you to analyze the algorithm given here in pseudocode (written by one of the Prof.'s Ph.D. students), but as usual, Prof. Nigma neglected to explain what the algorithm does.

```
M(x, y) :
if y = 0 return 0 else
return M(2x, ⌊y/2⌋) + x · (y mod 2)
```

- (a) What does this algorithm do? Give a clean, concise specification for the algorithm, and then show the algorithm satisfies that specification; as usual, explaining how the algorithm works will be part of your argument. (You do not need to give examples as part of your answer, but you are welcome to do so if it would make your answer clearer.)

- (b) Express the running time of the algorithm in a recurrence *as a function of the length of y only*—i.e., treat x as if it were of fixed length for this analysis, solve the recurrence using a method of your choice (from those shown in class), and give an informative asymptotic bound on the running time as the length of y changes.

For this exercise, assume the following:

- x and y are positive integers
- checking if a number is equal to 0 is a constant-time operation
- the addition of two integers is a constant-time operation
- the operation of finding an integer mod 2 is a constant-time operation
- the operation of integer division by 2 (e.g., $\lfloor y/2 \rfloor$) is a constant-time operation

If you're not sure what makes any of these assumptions reasonable in this context, please ask me!