

ECE 374 B ✧ Spring 2023

🌀 Homework 4 🌀

- Groups of up to three people can submit joint solutions. Each problem should be submitted by exactly one person, and the beginning of the homework should clearly state the Gradescope names and email addresses of each group member. In addition, whoever submits the homework must tell Gradescope who their other group members are.
 - **Submit your solutions electronically on the course Gradescope site as PDF files.** please use the \LaTeX solution template on the course web site. If you must submit scanned handwritten solutions, please use a black pen on blank white paper and a high-quality scanner app (or an actual scanner, not just a phone camera).
-

👉 Some important course policies 👈

- **You may use any source at your disposal**—paper, electronic, or human—but you *must* cite *every* source that you use, and you *must* write everything yourself in your own words. See the academic integrity policies on the course web site for more details.
 - **Avoid the Three Deadly Sins!** Any homework or exam solution that breaks any of the following rules will be given an *automatic zero*, unless the solution is otherwise perfect. Yes, we really mean it. We're not trying to be scary or petty (Honest!), but we do want to break a few common bad habits that seriously impede mastery of the course material.
 - Always give complete solutions, not just examples.
 - Always declare all your variables, in English. In particular, always describe the specific problem your algorithm is supposed to solve.
 - Never use weak induction.
-

See the course web site for more information.

If you have any questions about these policies,
please don't hesitate to ask in class, in office hours, or on Piazza.

For algorithmic problems as a whole, I want to emphasize that we will be grading clarity and conciseness of solutions (in addition to correctness) which admittedly is a subjective measure, but it is necessary. Being able to communicate your code is a far more important skill than being able to code itself.

If you need pages of text to describe your algorithm, you're doing something wrong. You should be editing and rewriting your solution multiple times before submission. **Figures can do more in a smaller space than pages of text, use them.** Before submitting your solution(s) ask yourself: "If I was completely unfamiliar with my algorithm, how long would it take me to understand it from the documentation provided?" Hopefully the answer to that is less than a few minutes.

1. Solve the following recurrence relations. For parts (a) and (b), give an exact solution. For parts (c) and (d), give an asymptotic one. In both cases, justify your solution.

(a) $A(n) = A(n-1) + 2n + 1; A(0) = 0$

(b) $B(n) = B(n-1) + n(n-1) - 1; B(0) = 0$

(c) $C(n) = C(n/2) + C(n/3) + C(n/6) + n$

(d) $D(n) = D(n/2) + D(n/3) + D(n/6) + n^2$

2. Consider the following variants of the Towers of Hanoi. For each of variant, describe an algorithm to solve it in as few moves as possible. **Prove** that your algorithm is correct. Initially, all the n disks are on peg 1, and you need to move the disks to peg 2. In all the following variants, you are not allowed to put a bigger disk on top of a smaller disk.

- (a) **Hanoi 1:** Suppose you are forbidden to move any disk directly between peg 1 and peg 2, and every move must involve (the third peg) 0. Exactly (i.e., not asymptotically) how many moves does your algorithm make as a function of n ?

- (b) **Hanoi 2:** Suppose you are only allowed to move disks from peg 0 to peg 1, from peg 1 to peg 2, or from peg 2 to peg 0.

Provide an upper bound, as tight as possible, on the number of moves that your algorithm uses.

(One can derive the exact upper bound by solving the recurrence, but this is too tedious and not required here.)

- (c) **Hanoi 3:** Finally consider the disappearing Tower of Hanoi puzzle where the largest remaining disk will disappear if there is nothing on top of it. The goal here is to get all the disks to disappear and be left with three empty pegs (in as few moves as possible).

Provide an upper bound, as tight as possible, on the number of moves your algorithm uses.

3. Below is a divide and conquer sorting algorithm called FIFTHSORT.

```

FIFTHSORT( $A[1..n]$ ):
  if  $n < 100$ :
    sort  $A$  by brute force
  else:
     $k = \lceil \frac{n}{5} \rceil$ 
    FIFTHSORT( $A[1..3k]$ )      ⟨First Sort⟩
    FIFTHSORT( $A[2k+1..n]$ )    ⟨Second Sort⟩
    FIFTHSORT( $A[1..3k]$ )      ⟨Third Sort⟩
    FIFTHSORT( $A[k+1..4k]$ )    ⟨Fourth Sort⟩

```

- (a) Prove that FIFTHSORT correctly sorts its input. [Hint: Where can the smallest k elements be?]
 - (b) Would FIFTHSORT still sort correctly if we replaced “if $n < 100$ ” with “if $n < 10$ ”? Justify your answer.
 - (c) Would FIFTHSORT still sort correctly if we replaced “if $n < 100$ ” with “if $n < 13$ ”? Justify your answer.
 - (d) Would FIFTHSORT still sort correctly if we replaced “ $k = \lceil \frac{n}{5} \rceil$ ” with “ $k = \lfloor \frac{n}{5} \rfloor$ ”? If the answer is yes, prove the correctness. If the answer is no, explain in what cases FIFTHSORT would fail.
 - (e) What is the running time of FIFTHSORT? (Set up a running-time recurrence and then solve it, ignoring the floors and ceilings.) Justify your answer.
4. Suppose we are given an array $A[1..n]$ of n *distinct integers*, which could be positive, negative, or zero, sorted in increasing order so that $A[1] \leq A[2] \leq \dots \leq A[n]$. Suppose we wanted to count the number of times some integer value x occurs in A . Describe an algorithm (as fast as possible) which returns the number of elements containing value x .
5. Given an arbitrary array $A[1..n]$, describe an algorithm to determine in $O(n)$ time whether A contains more than $n/4$ copies of any value. **Do not use hashing, or radix sort, or any other method that depends on the precise input values.**