CS 330, Fall 2020, Homework 1
Due Wednesday, September 9, 2020, 11:59 pm EST, via Gradescope

Homework Guidelines

Collaboration policy Collaboration on homework problems, with the exception of programming assignments and reading quizzes, is permitted, but not encouraged. If you choose to collaborate on some problems, you are allowed to discuss each problem with at most 5 other students currently enrolled in the class. Before working with others on a problem, you should think about it yourself for at least 45 minutes. Finding answers to problems on the Web or from other outside sources (these include anyone not enrolled in the class) is strictly forbidden.

You must write up each problem solution by yourself without assistance, even if you collaborate with others to solve the problem. You must also identify your collaborators. If you did not work with anyone, you should write "Collaborators: none." It is a violation of this policy to submit a problem solution that you cannot orally explain to an instructor or TA.

Solution guidelines For problems that require you to provide an algorithm, you must give the following:

1. a precise description of the algorithm in English and, if helpful, pseudocode,
2. a proof of correctness,
3. an analysis of running time and space.

You may use algorithms from class as subroutines. You may also use any facts that we proved in class.

You should be as clear and concise as possible in your write-up of solutions. A simple, direct analysis is worth more points than a convoluted one, both because it is simpler and less prone to error and because it is easier to read and understand. Points might be subtracted for illegible handwriting and for solutions that are too long. Incorrect solutions will get from 0 to 30% of the grade, depending on how far they are from a working solution. Correct solutions with possibly minor flaws will get 70 to 100%, depending on the flaws and clarity of the write up.

1. **Array operations and asymptotic running times** (10 points)

   (a) Consider the following pseudocode:

```
Algorithm 1: TestAlg($A$)
  Input: $A$ is an array of real numbers, indexed from 1 to $n$
  $n \leftarrow length(A)$ ;
  for $j \leftarrow 1$ to $\lfloor \frac{n}{2} \rfloor$ do
      $k = n + 1 - j$;
      $A[j] \leftarrow A[j] + A[k]$ ;
      $A[k] \leftarrow A[j] - A[k]$ ;
      $A[j] \leftarrow A[j] - A[k]$ ;
```

   Which of the following statements are true at the end of every iteration of the **for** loop?

   i. The sub-array $A[1 \dots j]$ contains its original contents in their original order
   ii. The sub-array $A[1 \dots j]$ contains the original contents of sub-array $A[(n+1-j) \dots n]$ in reverse order.
   iii. The sub-array $A[1 \dots j]$ contains its original contents in reverse order.
   iv. The sub-array $A[(n+1-j) \dots n]$ contains its original contents in their original order.
   v. The sub-array $A[(n+1-j) \dots n]$ contains the original contents of sub-array $A[1 \dots j]$ in reverse order.
   vi. The sub-array $A[(n+1-j) \dots n]$ contains its original contents in reverse order.

   For the ones that you select as always true, write a one sentence justification.

   Aside: Statements that hold on every iteration of a loop, like the correct options above, are called *loop invariants*. We will use them a lot when we analyze algorithms.

   (b) Consider the following *SwapAlg*( ) algorithm.

```
Algorithm 2: SwapAlg($A$)
  Input: $A$ is an array containing the first $n$ positive integers. It is indexed 1 to $n$.
  $n \leftarrow length(A)$ ;
  $swaps \leftarrow 0$;
  for $i = 1$ to $n$ do
      for $j = 1$ to $n - i$ do
          if $A[j] > A[j+1]$ then
              $A[j] = A[j] + A[j+1]$;
              $A[j+1] = A[j] - A[j+1]$;
              $A[j] = A[j] - A[j+1]$;
              $swaps++$;
  Output: $swaps$
```

   i. Give the best asymptotic upper bound you can on the running time of *SwapAlg*( ) as a function of n using big-Oh notation. Explain your computation.
   ii. Explain in one or two sentences what value the variable *swaps* is tracking. What is the number that this algorithm returns?

(c) Consider the *decAlg*( ) algorithm.

```
Algorithm 3: decAlg($A$)
  Input: $A$ is an array of integers. It is indexed 1 to $n$.
  $n \leftarrow length(A)$ ;
  $dec \leftarrow 0$;
  for $i = 2$ to $n$ do
      $j \leftarrow i - 1$;
      while $j > 1$ AND $A[j-1] > A[j]$ do
          $temp \leftarrow A[j-1]$;
          $A[j-1] \leftarrow A[j]$;
          $A[j] \leftarrow temp$;
          $j--$;
          $dec++$;
  Output: dec
```

i. Give the best asymptotic upper bound you can on the running time of *decAlg*( ) as a function of n using big-Oh notation. Explain your computation.

ii. Observe, that both in *SwapAlg*( ) and in *decAlg*( ) pairs of values in A are swapped. Both variables *swap* and *dec* keep track of the number of swaps performed in their respective algorithms. State and give an exact proof of what the relationship between the two variables are when the algorithms are run on the same input A. (The answer we are looking for is that *swap* is always smaller/equal/larger than *dec* or that it varies depending on the input array.)(*Hint: One way of proving it is to show that for any two values – initially at index $A[x]$ and $A[y]$ – if they are swapped at any given time in swapAlg( ) then they eventually will be swapped in decAlg( ) and vice verse.*)

(d) Consider the *CountAlg*( ) algorithm.

```
Algorithm 4: CountAlg($A$)
  Input: $A$ is an array containing the first $n$ positive integers. It is indexed 1 to $n$.
  $n \leftarrow length(A)$ ;
  $count \leftarrow 0$;
  for $i = 1$ to $n$ do
      for $j = i + 1$ to $n$ do
          if $i < j$ AND $A[i] > A[j]$ then
              $count++$
  Output: $count$
```

i. Give the best asymptotic upper bound you can on the running time of *countAlg*( ) as a function of n using big-Oh notation. Explain your computation.

ii. In fact, *count* in *countAlg*( ) and *swap* in *SwapAlg*( ) are closely related to each other. This is a non-trivial relationship that we're going to cover in detail later this semester. For now, give your best guess based on your intuition of what this relationship might be. Make sure you clearly state what your observation is and provide some justification. For the latter, show us how you came up with the observation.

(e.g. by showing your computation for the two values on some specific examples) (*Note: tracing algorithms on specific examples is very helpful in understanding the algorithm. But you need to do it by yourself to truly benefit from this.*)

2. **Group assignments** (10 points)

   In CS330, the instructors decide to divide the students in to small groups that will be assigned projects. In order to encourage students to get to know their classmates, no two students living on the same floor in the dorms can be assigned to the same group. (For simplicity assume that every student resides on campus.) The size of the individual groups doesn't matter, however the instructors want to create as few groups as possible.

   Denote the total number of students by n and the number of floors by k.

   (a) True or False? the number of groups one needs is always at least the maximum number of students living on any given floor. Justify your answer by giving a one-line proof (if it's true) or a counterexample (if it's false).

   (b) Describe a simple algorithm, $Floors(A)$, that takes an array A as its input, such that $A[i]$ contains the floor ID for student i. The output is an array or hash table (dictionary in Python) $floor$, such that $floor[j]$ consists of the list of students living on floor j. Write concise and clear *pseudocode* for your algorithm. (*Note: make sure you clearly indicate in your code what data structure(s) you are using*) Give a one sentence explanation on how your algorithm works.

   (c) Find a *simple* algorithm, $AssignStudents(A)$, that takes an array A as its input, such that $A[i]$ contains the floor ID for student i. The output is data structure (e.g. array, hash table) B, where $B[j]$ contains a *list* of students assigned to group j. Write concise and clear *pseudocode* for your algorithm. (You may want to call the $Floors(A)$ function as a subroutine in your algorithm – it's of course not required.) Give a short explanation in English on how your algorithm works.

   (d) Give the best asymptotic upper bound you can on the running time of $AssignStudents(\,)$ as a function of the number of students, n, using big-Oh notation. Explain your computation.

   Note: Your running time analysis will always be graded based on your ability to analyze *your* algorithm. That is, you have to give an analysis of the exact algorithm that you have written. That requires that you be specific about the data structure you use and what the running time of certain operations are on that structure. You may simply state your assumptions, as long as they are reasonable (e.g. you may reference what you know from CS112 about each structure.) Do not forget to include the running time of the $Floors(A)$ subroutine if you're using it!