Questions and/or Exercises to work out and turn in:

Grading Guidelines (See Appendix At The Bottom Of The document):

A right answer will get full credit when:

1. It is right (worth 25%)
2. It is right **AND** neatly presented making it easy and pleasant to read. (worth an **extra** 15%)
3. There is an **obvious and clear link[[1]](#footnote-1)** between 1) the information provided in the exercise and in class and 2) the final answer. A clear link is built by properly writing, justifying, and documenting an answer (worth an **extra** 60%).
4. Calculation mistakes will be minimally penalized (2 to 5% of full credit) while errors on units will be more heavily penalized.

You are welcome/encouraged to discuss exercises with other students or the instructor. But, ultimately, **personal** writing is expected.

* USE THIS FILE AS THE STARTING DOCUMENT YOU WILL TURN IN. **DO NOT DELETE ANYTHING FROM THIS FILE:** JUST **INSERT** EACH ANSWER **RIGHT AFTER ITS QUESTION/PROMPT**.
* IF USING HAND WRITING (STRONGLY DISCOURAGED), **USE THIS FILE** BY CREATING SUFFICIENT SPACE AND WRITE IN YOUR ANSWERS.
* FAILING TO FOLLOW TURN IN DIRECTIONS /GUIDELINES WILL COST **A 30% PENALTY.**

Objectives of this assignment:

* to use and manipulate the concepts presented in this module
* to use appropriately the counterexample and proofs by contradiction or induction.
* to use loop invariants to prove the correctness of an algorithm.

What you need to do:

Answer the questions and/or solve the exercises described below.

Exercise 1 (30 points) Proof by induction

Let us prove this formula:

1. (5 points) Try this expression with n = 3. Evaluate *S(3)* using each of the two expressions (the sum and the closed form and check whether they yield the same result).

With the Summation form of the expression, it would look like this:

S(3) = 0 + 2 + 4 + 6

S(3) = 12 would be the answer using summation.

With the Closed form of the expression, it would like this:

Now to simply express S(3) while using the closed form version of the expression, we can simply multiply n(n + 1). This would lead to the following expression.

S(3) = 3 \* (3 + 1)

S(3) = 3 \* 4

S(3) = 12 would be the answer using the closed form equation.

Both the summation form and the closed form version of the expression yield the result of S(3) = 12. This in turn proves that for n = 3 that the formula does in fact hold true.

1. Let us prove this expression using induction (proof must use induction)
   1. (5 points) Show the base (basis) case (propose the base and show that the formula above works by evaluating using each of the two expressions for the base, i.e., both expressions yield the same result)

With the Summation form of the expression, it would look like this:

First, we can propose the base as (n – 0) since i = 0. So using the same structure as the formula above and plugging in our proposed base of n – 0 would give the following evaluation:

Summation Form Expression

S(0) = 2 \* 0 = 0

Closed Form Expression

Then using just the closed form version of the expression using the base n – 0 would yield:

S(0) = 0(0 + 1)

S(0) = 0 + 0 = 0

* 1. Show the induction step by answering these questions:
     1. (4 points) What is your hypothesis to use for the induction step?

The hypothesis to use for the induction step is if the formula holds for n, then it should also hold true for n + 1.

This would make the summation formula as follows:

* + 1. (16 points) Now, complete the induction step

Now assuming the formula is true for n, such that:

Then next we need to prove for n + 1 as follows:

First, we use the induction hypothesis from above and then add the next term for the series. We replace i with n + 1 on both sides which gives us:

S(n) + 2(n + 1) = n(n + 1) + 2(n + 1). This then can simplify into.

S(n + 1) = n2 + n + 2 + 2n. And then further simplifies it too.

S(n + 1) = n2 + 3n + 2. Then we can factor the polynomial on the right side to the following.

S(n + 1) = (n + 1)(n + 2).

So, this shows if the formula ends up holding n, then it should also hold for n + 1. This completes the step of induction.

Exercise 2 (70 points) Loop Invariant

Consider the algorithm *getIndexMaximum(A,k)* that takes a sequence A as an input and returns the index of the largest number (maximum) in the range [k-A.length] in Sequence A.

For example, let A ={200, 22, 14, 5, 22, 7}. *getIndexMaximum(A,3) will return 5 because 5 is the index of the element 22 and 22 is the largest number in A in the range [3-6].*

Consider the following sorting algorithm that sorts a sequence A in decreasing order:

Sort-Array(A)

for i = 1 to A.length

IndexMax = getIndexMaximum(A,i)

// swap A[i] and A[IndexMin]

buffer = A[IndexMax]

A[IndexMax] = A[i]

A[i] = buffer

The objective is to prove that the above sorting algorithm is correct. Use the textbook in Section 2.1: the authors show that the *InsertSort* algorithm is correct using loop invariants. Their proof should help you with this exercise. It is expected and strongly advised that you follow their steps in Section 2.1.

Advice: get familiar with Sort-Array by executing the algorithm Sort-Array(A) when A ={7, 3, 20, 100}

1. (5 points) Express the property that Sort-Array(A) must satisfy to be correct:

The sorting algorithm “Sort-Array(A) must satisfy the following condition or correctness property in order to be considered correct. After the array named “A” is finished sorting, the values should be in decreasing order such that both indices “i” and “j” are “1 <= i < j <= A.length, giving us A[i] >= A[j]

This inherently says that for every element in the array must be greater than or equal to every other element that follows it. This, in fact, will insure that the array is sorted in descending order in values.

1. (12 points) List four loop invariants (even if they are not that helpful for our ultimate proof of correctness of Sort-Array). Specify whether your loop invariant applies before or after the iteration.
2. Subarray Maximum Element (Before): Before each iteration of the loop, the element at A[i] is the maximum within the subarray A[i..A.length]. This invariant applies before the swap operation, ensuring that the correct element is selected for positioning.
3. Stable Elements (After): After each iteration, all elements in A[1..i] have not been moved again. The finality of each element's position once it has been placed, shows the ability of the algorithm to do work and move through the sorting process.
4. Remaining Unsorted Elements (Before): Before each iteration, the subarray A[i + 1..A.length] consists of all elements not yet sorted into their final position. This invariant progresses the sorting algorithm , focusing on the difference between sorted and unsorted portions of the array.
5. Invariant Size Reduction (After): After each iteration, the size of the unsorted portion of the array A[i+1..A.length] decreases by one. This invariant is a direct consequence of the sorting process, highlighting the iterative reduction of the array's unsorted section.
6. (15 points) Propose a loop invariant for the outer loop that is the closest to our ultimate objective: Sort-Array is correct.

Right before we begin each iteration of the outer loop, indicated by i, the section of the array A[1 .. i - 1] is already sorted in a descending manner. Additionally, each element in this part of the array is greater than or equal to all elements in A[I .. A.length]. This loop invariant is crucial for proving the Sort-Array algorithm works because it demonstrates how elements are progressively placed in their correct positions. It ensures the first part of the array is not just sorted but also correctly ranked compared to the rest, confirming these are the top elements up to that point.

1. Use the three steps:
   1. (8 points) Initialization

At the very start, with i = 1, the algorithm hasn't yet performed any swaps, so A[1..i-1] (the empty segment) is in descending order, meeting the invariant. This step sets the stage for the algorithm to begin sorting by identifying and moving the maximum elements to their proper positions, starting from the first element.

* 1. (18 points) Maintenance

As the algorithm progresses from i = 1 to A.length, for each i, it will call getIndexMaximum(A,i) to find the index IndexMax of the maximum element in the unsorted portion of the array starting from index i. It will then swap the element at A[i] with the element at A[IndexMax], using a buffer variable for the swap. This ensures that after each iteration, segment A[1..i] is sorted in descending order, as the largest element from the remaining unsorted array has been correctly placed at the start of this unsorted segment. This action upholds the invariant by confirming the segment A[1..i-1] is sorted and each element in this part is greater than or equal to elements in A[i..A.length].

* 1. (12 points)Termination

When i exceeds A.length, the loop finishes, indicating that every element has been compared, and the maximum elements have been successively placed from the start to the end of the array. At this point, the entire array A[1 .. A.length] is sorted in descending order. The final array state demonstrates the successful application of our invariant throughout the sorting process, ensuring that the Sort-Array algorithm correctly sorts the array.

**Appendix**: Grading: What is an OBVIOUS and CLEAR LINK?

Here is an example to explain what an **obvious and clear link** is and how we grade your work.

Consider the following problem:

"(100 points) John travels from Auburn to Atlanta in his car at a speed of 60 mph. Leaving at 8am, at what time will John reach Atlanta".

Here are the answers of three students and their scores:

* **Student 1** answers: "9:48am". Student 1 will get 25 points.
* **Student 2**answers : "John will reach Atlanta at 9:48am". Student 2 will get 25+15 = 40 points
* **Student 3** answers: "The time t to travel a distance d at speed v is equal to d/v = d/60mph. The problem does not provide the distance d from Auburn to Atlanta. Based on GoogleMaps, the distance from Auburn to Atlanta is approximately 108 miles (**document is attached**).



Therefore, the time t = 108 miles/60mph \* 60 minutes/hour= 108 minutes. Since John left at 8am, he will then reach Atlanta at 8am + 108 minutes = 8 am + 60 minutes + 48 minutes = 9:48".

**Student 3** will get 25 + 15 + 60 = 100 points

Do you see the **direct** **link** going from the data provided in the question to the final answer, using general knowledge/formula and documents?.... Can you now solve the following problem and get 100 points?

"(100 points) Alice travels from Auburn to Atlanta in her car at a speed of 60 mph. Leaving at 8am, at what time will Alice reach Atlanta assuming that she had a flat tire that delayed her 30 minutes".

1. See Appendix to know more about an obvious and clear link. [↑](#footnote-ref-1)