# Week 3 Workshop: Correctness and performance of programs

- **Resources**: Week 1 and 2 code bundles.
- **To submit this week's work:** Submit your solution to Exercise 2 to your teacher in class. You should hand in a single piece of paper with your solution on it. Be sure you include your name, student number, and the week number in the top-right of your submission.

# Exercises

## 1 Problem Solving: Warmup

How much does the ice in a hockey rink weigh?

For this question, getting exactly the right answer is not the point (although it's good when it's in the "ball park"). Questions like these help you to practise thinking about a problem and to define the assumptions that need to be made to give a reasonable answer. In your group, discuss some assumptions you can make to enable you to do a "back of the envelope" calculation. In your solution list the assumptions that you made in order to come up with your answer.

Note also that in "real life" most problems are ill defined (such as this one) and to be a good problem solver and critical thinker, you need to be able to formulate the problem in a way that gives something of a meaningful (if not always useful!) answer.

## 2 Invariants: Basics - Submission

Consider the following program code. Assume that there is a given linked list with first node `head`, and a variable `x` initialised to some value.

```
char x = <<?>>; LinkedList head = <<?>>;
LinkedList temp = head;
while(temp != null && temp.value != x) {
   temp= temp.next;
}
```

For each of the example lists in the diagram (each containing 4 nodes) diagrammatically show which object/node `temp` points to after running this algorithm.

`char x = 'b';`

$$\begin{tikzcd}\text{head} \arrow[r] & \boxed{\text{'a'}} \arrow[r] & \boxed{\text{'b'}} \arrow[r] & \boxed{\text{'c'}} \arrow[r] & \boxed{\text{'d'}} \arrow[r] & \phi\end{tikzcd}$$

`char x = 'b';`

$$\begin{tikzcd}\text{head} \arrow[r] & \boxed{\text{'a'}} \arrow[r] & \boxed{\text{'a'}} \arrow[r] & \boxed{\text{'b'}} \arrow[r] & \boxed{\text{'b'}} \arrow[r] & \phi\end{tikzcd}$$

`char x = 'd';`

$$\begin{tikzcd}\text{head} \arrow[r] & \boxed{\text{'a'}} \arrow[r] & \boxed{\text{'b'}} \arrow[r] & \boxed{\text{'c'}} \arrow[r] & \boxed{\text{'b'}} \arrow[r] & \phi\end{tikzcd}$$

## 3 Invariants: Any old thing

Predicates belong between two lines of code. Create six predicates for each possible slot (before line one, between line one and two, ...., between line four and five, after line five) of the code in question 2.

Don't worry about how useful your predicates are, just that they are true.

## 4 Invariants: Simple Program

Predicates are "Invariants" if they are true in every slot of the program. Consider the following invariant of the program above:

> For all nodes `n` lying between `head` and `temp` (but not including `temp`) it is the case that `n.value != x`.

In the case that `temp` is `null` because it has traversed the whole list, then take this statement to mean "all nodes in the list".

1. If the loop ends with `temp == null`, what does the invariant tell us about the contents of the list?
2. If the loop ends with `temp != null`, what does the invariant tell us about the contents of the list?
3. Hence or otherwise, suggest a suitable post-condition for this loop to tell you what it does.

## 5 Loop Invariants: Post-conditions

Consider the variations below of the sum algorithm discussed in lectures. In each case use the loop invariant given and work out the post-condition by putting the invariant together with the "negation of the guard". Once you have the post-condition, put into words what the loops do. **You may assume that the loop invariant given has been checked and that it is indeed and loop invariant.**

1.
   ```
   int i=0;  int sum=0;
   while ( i <= N ) {
      i= i+1;
     sum= sum + i;
   ```

```
}
```
Use loop invariant sum $= \sum_{j=0}^{j=i} j$

2.
   ```
   int i=1;  int sum=0;
   while ( i < N ) {
     sum= sum + i;
      i= i+1;
   }
   ```
   Use loop invariant $\sum_{j=0}^{j=i-1} j = sum$

3.
   ```
   int i=1;  int sum=0;
   while ( i < N+1 ) {
     sum= sum + i;
      i= i+1;
   }
   ```
   Use loop invariant $\sum_{j=0}^{j=i-1} j = sum$

## 6 Loop Invariants: Program Correctness

Consider the following small program:

```
//  pre  :  X <= Y
//  post : returns the sum X + (X+1) + ... + Y
int sumBetween (int X, int Y){
  int sum = 0;
  for (int i=X; i!=Y+1; i++)
      sum += i;
  return sum;
}
```

1. State the loop invariant. (Hint: Use the example for summing the numbers between 0 and N discussed in lectures as a starting point.)
2. Show that the loop invariant is established by the initialization.
3. Verify that the loop invariant is preserved by each iteration of the loop.
4. Show that the loop invariant and the termination condition imply the post-condition.
5. What is the complexity in big oh notation of this program?

## 7 Performance: Big Oh

Recall that Big Oh is a *shorthand notation* for the rate of growth based in input size. For any program, you can work out a relationship between the input size ($n$) and the run-time. This relationship is then given as a function. If a program gets 1 cycle slower every time one new element goes into the input, it's performance function is $n$. If a program gets 10 cycles slower for each new input, its performance function is $10n$

Recall also that Big Oh is a *rough* measure, which captures only the *shape* of the function. Both the $n$ and $10n$ performance functions are straight lines and thus should have the same Big Oh value, in this case $O(n)$.

Work out the big Oh equivalent of the following performance functions.

1. $n^2 + 100n$
2. $n \times (4n + 20)$
3. $400$
4. $n^2 + \log n + 11$

**Hint**

Here is the table from lectures to help

| **iterative** | **recursive** | **Big Oh** |
|---|---|---|
| No loop | $T(n)) = C_1$ | $O(1)$ |
| Normal loop | $T(n) = C_1 + C_2 * T(n-1)$ | $O(n)$ |
| Nested loop | $T(n) = C_1 + T(n-1) * T(n-1)$ | $O(n^2)$ |
| Halving loop | $T(n) = C_1 + C_2 * T(n/2)$ | $O(\log n)$ |
| Halving Nested Loop | $T(n) = C_1 + T(n-1) * T(n/2)$ | $O(n \log n)$ |

# Self Study Exercises

## 1 Invariants: Insertion Sort

Consider the implementation below for Insertion Sort, which sorts an array into descending order.

1. Trace the algorithm for the array `A = {2,1,5,0,6}`, and `n=5`; make sure you keep track of the array after each inner and outer iteration. In your trace, draw a rectangle around the array after each complete iteration of the OUTER loop (see (++) in the code).
2. Verify that the array portion `A[0..j-1]` is sorted after each complete iteration of the outer loop, i.e. underline that portion of the array and check that it is sorted. (Note that `A[0..j-1]` means "the portion of the array between, and including, indices `0` and `j-1`.)
3. Now check that the invariant for the outer loop is established (made to be true) by the initialisation of the variable j; i.e. substitute the initial values of the relevant variables into the invariant statement and observe that it is true.
4. Finally deduce that the postcondition is satisfied after the outer loop has terminated.

   [Hint: what is the value of `j` on termination? Substitute into the invariant.]

```
// PRE:
// POST: array is sorted
void insertion_sort(int array[]) {
     int i, j, key;
     for(j = 1; j < array.length; j++) {    //Notice starting with 1 (not 0)
     // Invariant for the outer loop: array[0..j-1] is sorted, descending order
           key = array[j];
           for(i = j - 1; (i >= 0) && (array[i] < key); i--){
              //Move smaller values up one position
              array[i+1] = array[i];
           }
           array[i+1] = key;    //Insert key into proper position (++)
       }
      return;
}
```

## 2 Performance: Big Oh

Work out the big Oh equivalent of the following performance functions.

1. $n^2 + n \log n$
2. $n \times (4n + \log n)$
3. $4 \times 10^126$
4. $n^2 * \log n + 11$

(*Hint:* Your answer should look like one of the following: $O(1)$, $O(\log n)$, $O(n)$, $O(n \log n)$, $O(n^2)$, $O(n^2 \log n)$, $O(n^3)$, $O(n^3 \log n)$, or $O(2^n)$.)

## 3 Invariants: Recursive Function

Consider the following function on linked lists (use the definition of `LinkedList` discussed in class and in the code bundle).

```
LinkedList splitAt(char c){
  if (next == null){
      return null;
  } else {
      if (value == c){
          LinkedList ret = next;
          next = null;
          return ret;
      } else {
          return next.splitAt(c);
      }
  }
}
```

Come up with a post-condition for this function. You know what it is trying to do, so come up with something *precise* that expresses that and relates to variables in the program.

## 4 Invariants: Iterative Function

Consider the following function on linked lists (use the definition of `LinkedList` discussed in class and in the code bundle).

```
  LinkedList splitAt_iterative(char c){
    LinkedList curr = this;
    while(curr != null){
        if (curr.value == c){
            LinkedList ret = curr.next;
            curr.next = null;
            return ret;
        }
        curr = curr.next;
    }
    return null;
  }
```

Come up with a post-condition for this function. You know what it is trying to do, so come up with something *precise* that expresses that and relates to variables in the program. Note that this is the same function as above, but written in an iterative style

## 5 Invariants: Basic Concepts

Have a conversation with your preferred Generative AI tool to find out if it can accurately explain *how can you use loop invariants to prove the correctness of a program snippet*. Try and find an error in it's responses if you can.