***THINK TWICE, CODE ONCE***

**Good to know:**

* Use the *Set* class if no duplicate object is wanted in our collection
* *HashSet* only takes in 1 type argument, and it has NO get() method! To find an element in a *HashSet* instance, the iterator must be used(or just use a HashMap instead)…
* Integer reverse the **quick** way:

Int x = 0;

While (num != 0){

x \* = 10;

x +=num % 10;

num /= 10;

}

return x;

(This is WAY faster than using parsing num into a StringBuilder instance, then calling reverse().toString() and parsing back into an int!)

* To find the “closer” number to the target, use:

Math.abs(target - num1) < Math.abs(target - num2) !!

**Algorithms**

**Sliding window algorithm:**

<https://stackoverflow.com/questions/8269916/what-is-sliding-window-algorithm-examples>

* Sliding window only works when:
* The input values are all positive/ negative
* The input values are sorted

**Greedy algorithms:**

* Returns the local optimal solution, which is sometimes equal to the global optimal solution!

**Floyd’s cycle detection algorithm:**

<http://onwaier.com/?p=528>

* Time complexity: O(n), Space complexity(with the 2 pointers implementation): O(1)
* Used to detect whether a cycle exists in a linked list

Pseudocode:

* We initialize two pointers, fastPtr and slowPtr, which both points to the start of the list
* Set up a loop. In each iteration, we move fastPtr forward for 2 positions, slowPtr forward for 1 position.
* Two things can happen if we keep the loop running:

1. fastPtr reaches the end of the list, there is NO CYCLE
2. fastPtr meets slowPtr, there is CYCLE

**Kadane’s Algorithm:**

<https://medium.com/@rsinghal757/kadanes-algorithm-dynamic-programming-how-and-why-does-it-work-3fd8849ed73d>

* Time complexity: O(n), Space complexity: O(1)
* Used to find the contiguous subarray with the largest sum

Pseudocode:

* We initialize local\_max[n], where local\_max[i] stores the largest contiguous subarray’s sum up to index i.
* local\_max at index i is the larger of A[i] and the sum of A[i] and local\_max at index i-1

**local\_max[i] = max(A[i], local\_max[i-1] + A[i].**

**String**

* Use the StringBuilder class whenever possible
* To find a palindrome of any String x, simply convert String x to a StringBuilder instance y, and call y.reverse()