# **Sliding Window Technique**

## **What Is the Sliding Window Technique?**

* The **sliding window algorithm** can be used to solve problems that involve **sequentially iterating** over a **contiguous** **sequence** of elements.
  + For example: Finding the maximum subarray in an array.
* **Contiguous** describes either
  + one object

or

* + two or more objects that are adjacent to each other
* A **contiguous sequence** consists of **elements that occur one after another in a linear sequence of data**.
* Using the technique, we **slide** a **window** (**subset of data)** over a **contiguous sequence** of elements.
* A **window** is a contiguous sequence which is a subset of a larger collection.
* The window is **slides** (**expands or shrinks**) over the collection to satisfy certain conditions.
* Some **operations** are performed on elements within this window, and then the window is slid further down the collection.
* This technique shows how a nested for loop in some problems can be converted to a single for loop to reduce the time complexity.

## **When Should We Use It?**

* Contiguous sequences of elements are any collections that can be linearly iterated over:
  + Strings
  + Arrays
  + Linked lists
  + Trees
  + Etc.
* Algorithms that can be solved using the sliding window technique usually involve finding:
  + Minimum value
  + Maximum value
  + Longest subset/substring
  + Shortest subset/substring
  + Contains value(s)

## **Question Variants**

1. Fixed Length Window Size

Example: Find the max sum subarray of size k

1. Fixed Length with Auxiliary Data Structure

Example: [Sliding Window Maximum [hard]](https://leetcode.com/problems/sliding-window-maximum/)

1. Dynamic Length (window is dynamic is size)

Example: Find the smallest sum subarray >= to some value s

Example: [Maximum subarray [easy]](https://leetcode.com/problems/maximum-subarray/)

Example: [Minimum Size Subarray Sum [medium]](https://leetcode.com/problems/minimum-size-subarray-sum/)

// global variable you are tracking (i.e. global max sum or global max window)

// local variable to track current window constraint (i.e. local max sum or local max window)

// tracker variables for front and back of window

for(int i = 0; i < arr.length; i++)

{

// update local variable

while(local variable constraint to meet problem constraint)

update window size by updating start

}

1. Dynamic Length w/ Auxiliary Data Structure (keep aux ds to keep track of variable)

Example: Find the longest substring w/ no more than k distinct characters

Example: [Minimum Window Substring [hard]](https://leetcode.com/problems/minimum-window-substring/)

// global data structure you are tracking (i.e. max sum)

// local variable to track current window constraint (i.e. max sum or max window)

// tracker variables for front and back of window

for(int i = 0; i < arr.length; i++)

{

// update local variable

while(local variable constraint to meet problem constraint)

update window size by updating start

}

1. String Permutations
   1. Everything is grouped sequentially (substring, subarray, etc…)
   2. Longest/smallest/contains/maximum/minimum value/sub
   3. There are some criteria to build the window around

Sample Examples of Each:

<https://gist.github.com/Schachte/87d7c0165a584f26b3ad7845f8010387>