**-----------------------------------Array------------------------------------**

Arrays are fixed size sequence containers. They holds a specific number of elements ordered in a linear sequence.

An array is a collection of items stored at contiguous memory locations.

**Member functions:**

**Iterators:** begin, end, rbegin, rend, cbegin, cend, crbegin, crend

**Capacity:** size, max\_size, empty

**Element access:** operator[], at, front, back, data

**Why do we use array:**

Arrays are used to store multiple values in a single variable instead of declaring separate variables for each value.

**Advantage:**

* Arrays represent multiple data items of the same type using a single name.
* In arrays, the elements can be accessed randomly by using the index number.
* Arrays allocate memory in contiguous memory locations for all its elements. Hence there is no chance of extra memory being allocated in case of arrays. This avoids memory overflow or shortage of memory in arrays.
* Using arrays, other data structures like linked lists, stacks, queues, trees, graphs etc can be implemented.
* Two-dimensional arrays are used to represent matrices.

**Disadvantage:**

* The number of elements to be stored in an array should be known in advance.
* An array is a static structure (which means the array is of fixed size). Once declared the size of the array cannot be modified. The memory which is allocated to it cannot be increased or decreased.
* Insertion and deletion are quite difficult in an array as the elements are stored in consecutive memory locations and the shifting operation is costly.
* Allocating more memory than the requirement leads to wastage of memory space and less allocation of memory also leads to a problem.

**Applications:**

Array stores data elements of the same data type.

Maintains multiple variable names using a single name. Arrays help to maintain large data under a single variable name. This avoid the confusion of using multiple variables.

Arrays can be used for sorting data elements. Different sorting techniques like Bubble sort, Insertion sort, Selection sort etc use arrays to store and sort elements easily.

Arrays can be used for performing matrix operations. Many databases, small and large, consist of one-dimensional and two-dimensional arrays whose elements are records.

Arrays can be used for CPU scheduling.

Lastly, arrays are also used to implement other data structures like Stacks, Queues, Heaps, Hash tables etc.

**-----------------------------------Vector------------------------------------**

Vectors are sequence containers representing arrays that can change their size during runtime.

**Member functions:**

**Iterators:** begin, end, rbegin, rend, cbegin, cend, crbegin, crend

**Capacity:** size, max\_size, empty, resize, capacity, reserve, shrink\_to\_fit

**Element access:** operator[], at, front, back, data

**Modifiers:** assign, push\_back, pop\_back, insert, erase, swap, clear, emplace, emplace\_back

**Why do we use vector:**

They have a dynamic structure and provide programmers with the ability to allocate container size and memory space quickly.

**Advantage:**

Size of the vector can be changed.

Multiple objects can be stored

Elements can be deleted from a vector

Elements can be inserted and deleted easily

It is very easy to copy vectors from one to another by just using assignment operator

**Disadvantage:**

A vector is an object, memory consumption is more.

It doesn’t use contiguous memory

It is not indexed.

**Applications:**

Used in computing in computer graphics and simulating physical systems.

**Array v/s Vector:**

Vector can never be faster than an array.

Vector occupies much more memory in exchange for managing storage and growing dynamically, whereas arrays are a memory efficient data structure.

**Linear Search (sequential search) - application problems**

* + It is one of the searching algorithms which we have some data and we have to search a particular element in which is known as key.
  + By traversing the whole data structure elements from start to end one by one to find key comparing with each data structure element to the key.

Advantage:

* + When a key element matches the first element in array, the linear search algorithm is best case because executing time of linear search algorithm is O (n).

Disadvantage:

* + When a key element matches the last element in the array or a key element doesn’t matches any element then linear search algorithm is a worst case.

Application:

* Linear search can be applied to both single-dimensional and multi-dimensional arrays.
* Linear search is easy to implement and effective when the array contains only a few elements.
* Linear Search is also efficient when the search is performed to fetch a single search in an unordered-List.

Time Complexity: Best case: O (1)

Worst case: O (n)

Average case: O (n / 2)

Space Complexity: O (1)

**Binary Search - application problems**

* + Binary search is a searching algorithm used in a sorted array by repeatedly dividing the search interval in half.
  + The idea of binary search is to use the information that the array is sorted and reduce the time complexity O (log n).

Advantage:

* + It eliminates half of the list from further searching by using the result of each comparison.
  + It indicates whether the element being searched is before or after the current position in the list.
  + This information is used to narrow the search.
  + For large lists of data, it works significantly better than linear search.

Disadvantage:

* + It employs recursive approach which requires more stack space.
  + Programming binary search algorithm is error prone and difficult.
  + The interaction of binary search with memory hierarchy

Application:

* + Find an element in a sorted array
  + Dictionary
  + Debugging a linear piece of code
  + Figuring out resource requirements for a large system
  + Find values in sorted collection
  + Numerical solutions to an equation

Time Complexity:

Worst case: O (log n)

Best case: O (1)

Average case: O (log n)

Space complexity:

O (1)

**Two pointer approach**

* + It is used for searching pairs in a sorted array.
  + Two pointer is a pattern in which two pointers iterate across the data structure until one or both of them satisfy the necessary condition.
  + Two approaches:
    - Naïve approach O (n2)
    - Optimal approach O (n)

**Three pointer approach**

* + It is used for searching the 3-pairs in a sorted array.
  + 3 approach:
    - Naïve approach O (n3) space: O (1)
    - Optimal approach O (n2) space: O (1)
    - Hashing based approach O (n2) space: O (n)

**Reversal algorithm**

* + In this algorithm subarrays are created and reversed to perform the rotation of the array.
  + 3 approach:
    - Rotate one by one Time: O (n \* d) Space: O (1)
    - Juggling algorithm Time: O (n) Space: O (1)

Instead of moving one by one, divide the array in different sets where number of sets is equal to GCD of n and d and move the elements within sets.

* + - Reversal algorithm. Time: O (n) Space: O (1)

**DNF (Dutch National Flag) algorithm**

The flag of the Netherlands consists of three colors: white, red, and blue. The task is to randomly arrange balls of white, red, and blue in such a way that balls of the same color are placed together. For DNF, we sort an array of 0, 1, and 2's in linear time that does not consume any extra space. We have to keep in mind that this algorithm can be implemented only on an array that has three unique elements.

Time: O (n) Space: O (1)

Algorithm:

* + Take three pointer – low, mid, high
  + We use low and mid pointers at the start, and the high pointer will point at the end of the given array.

**Sliding Window**

* + Sliding window algorithm can be used to solve problems that involve linear sequence such as arrays.
  + The condition to use the sliding window technique is that the problem asks to find the maximum or minimum value for a function that calculates the answer repeatedly for a set of ranges from the array.
  + it is considered as a technique that could reduce the time complexity from O(n²) to O(n).

Example:

We are given an array of size N*N*; we need to calculate the maximum sum of K*K* consecutive elements.

The most obvious solution would be to find all possible blocks of K*K* elements, and pick the one with the largest sum.

The optimal solution, however, uses the sliding window algorithm. Here is how it works:

1. Add the first K*K* elements together, and store the sum in the variable currentSum. Since this is the first sum, it is also the current maximum, so store it in variable maximumSum as well.
2. Since the window size is K*K*, we ***slide*** the window one place to the right, and compute the sum of the elements in the window.
3. If the currentSum is larger than the maximumSum, then update the maximum and repeat step 2. This is illustrated below:

Time: O (n)

Space: O (1)

**Moore voting algorithm**

* + Given an integer array of length N. Check if there exists an integer which occurs more than floor(N/2) times.
  + It is used to find the majority element in the given array.

floor(): returns the largest possible integer value which is less than or equal to the given argument

Time: O (n) and space: O (1)

**Pigeon hole principal**

* + Pigeon hole sorting algorithm that is suitable for sorting lists of element where the number of elements and the number of possible key values are approximately the same.
  + Time: O(n + range) and space: O(1)

**Frequency based and prefix sum based problems**