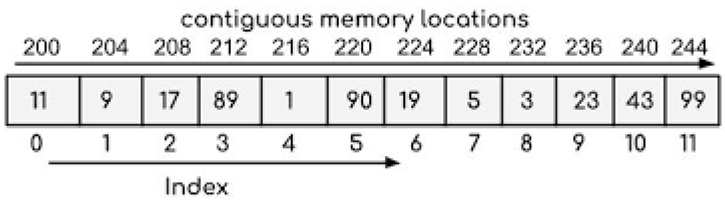
Arrays

What is Array?

- An `array` is a collection of items stored at contiguous memory locations.
- In Arrays multiple items of same type can be stored together which makes it easier to calculate the position of each element by simply adding an offset to a base value.



Defining An Array

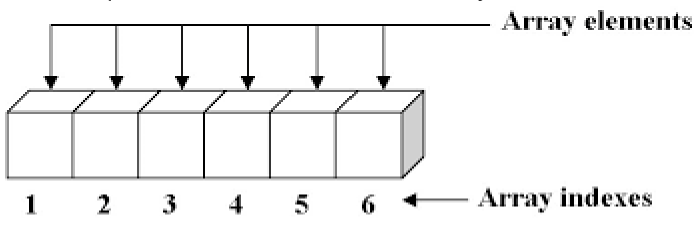
- The size of the array is `fixed` and the memory for an array needs to be allocated before use, `the size of an array cannot be increased or decreased dynamically.`

General Declaration

```
dataType arrayName[arraySize];
```

Accessing array elements

- Indexing of an array starts from 0 to n-1;
- Element present at ith index in the array arr[] can be accessed as arr[i].



One-dimensional array with six elements

Operations:-

1. Searching

- We can use loops to perform the above operation of array traversal and access the elements, using indexes

> Eg :-. For searching the element stored in key in array of size n

> **Example Code**

```
        int a[n]; // Array Declaration
         //Traversing An Array Through Loop

         for(i=0 ;i<n;i++) {

         if(arr[i]==key) {

         print "ELEMENT FOUND";

         }

         else{
         print "ELEMENT NOT FOUND";
         }   }
```

> **Time Complexity** : -
>
> **O(n) - as worst Case if required element is at last index**
>
> **O(1) - as best case if required element is at first position**

▸ 2. Insertion

1.Insert elements at the end of the array.

- Suppose there is an array 'arr' of size 'N' and length of an array is 'len'.
- To insert an element k at the end of the arr[] The first step is to check if there is any space left in the array for new element.

```
if(len < N)
     // space left
else
     // array is full
```

- If there is space left for the new element, insert it directly at the end at position len + 1 and index len:

```
arr[len] = k;
```

> **Time Complexity :-**
>
> **O(1)** as we are directly inserting the element in a single operation.

2. Insert element at any given index in the array.

- Suppose we have to insert element ‘k’ at index ‘idx’.
- For this opearation we have to check if there is any space left in the array for new element.
- To do this check following ,

```
if(len < N)
     // space left
else
     // array is full
```

- Now, if there is space left, the element can be inserted. The index of the new element will be idx = pos - 1.
- Before inserting the elements at idx shift all elements from the index idx till end of the array to the right by 1 place.

> **Implementation**

```
for(i = len-1; i >= idx; i--)
{
    arr[i+1] = arr[i];
}
```

- After Shifting the elements place k at index idx. arr[idx] = K;

> **Time Complexity : -**
>
> **O(N)** as in worst case we might have to shift all of the elements by one place to the right.

▸ 3.Deletion

- To delete a given element from an array, we will have to first search the element in the array.
- If the element is present in the array then delete operation is performed for the element otherwise the user is notified that the array does not contains the given element.
- For Deleting an element K from the array arr[] , Search the element K in the array arr[] to find the index at which it is present.

```
for(i = 0; i < N; i++){
    if(arr[i] == K)
        idx = i; return;
    else
        Element not Found;
}
```

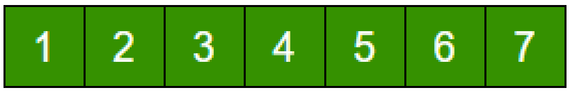
- Now, to delete the element present at index idx, left shift all of the elements present after idx by one place and finally reduce the length of the array by 1.

```
for(i = idx+1; i < len; i++)
{
    arr[i-1] = arr[i];
}
len = len-1;
```

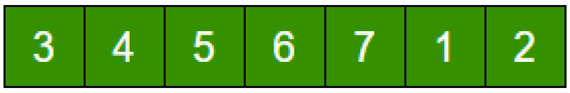
> **Time Complexity : -**
>
> **O(N)** in worst case as we might have to shift all of the elements by one place to the left.

▸ 4. Array Rotation

- Consider the following array :-



- The above array is rotated `counter-clockwise`(towards left) by 2 elements.



* After rotation, the array will be:

The process looks like following :-

1. Shift all elements after K-th element to the left by K positions.
2. Fill the K blank spaces at the end of the array by first K elements from the original array.

`The similar approach can also be applied for clockwise array rotation.`

> **Implementations**

a) Simple Method

1. Store the first K elements in a temporary array say temp[].
2. Shift all elements after K-th element to the left by K positions in the original array.
3. Fill the K blank spaces at the end of the original array by the K elements from the temp array.

> **Algorithm**
>
> Say, `arr[]` = [1, 2, 3, 4, 5, 6, 7] , `K` = 2
>
> 1. Store first K elements in a temp array temp[] = [1, 2]
> 2. Shift rest of the arr[] arr[] = [3, 4, 5, 6, 7, 6, 7]
> 3. Store back the K elements from temp arr[] = [3, 4, 5, 6, 7, 1, 2]

> **Time Complexity : - O(N)** as we have to shift all N elements.

> **Auxiliary Space : - O(K)** where k is the number of places by which elements will be rotated.

b) Another Method (Without Extra Space)

- We can also rotate an array by avoiding the use of temporary array.
- The idea is to rotate the array one by one `K times.`

> **Algorithm**
>
> 1. Store the first element in a temporary variable say temp.
> 2. Left shift all elements after the first element by 1 position. That is, move arr[1] to arr[0], arr[2] to arr[1] and so on.
> 3. Initialize arr[N-1] with temp.

- To rotate an array by K position to the left, repeat the above process K times.

> Eg :-
>
> arr[] = [1, 2, 3, 4, 5, 6, 7], K = 2
>
> Rotate arr[] one by one 2 times.
>
> After 1st rotation: [2, 3, 4, 5, 6, 7, 1] After 2nd rotation: [3, 4, 5, 6, 7, 1, 2]

> **Time Complexity :- O(N*K)** where `N` is the number of elements in the array and `K` is the number of places by which elements will be rotated.

> **Auxiliary Space : - O(1)** as one storage space is used.

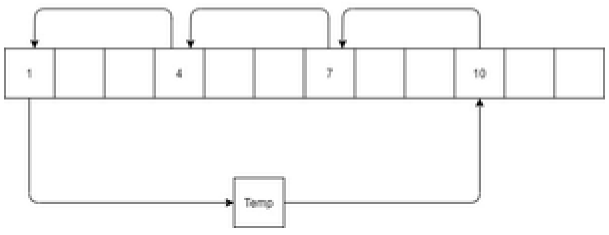
c) Juggling Algorithm : -

- Here , Instead of moving one by one , `divide the array in different sets where number of sets is equal to GCD of N and K and move the elements within sets.`

- `If GCD is 1` as is for the above example array (N = 7 and K = 2), then elements will be moved within one set only, we just start with temp = arr[0] and keep moving arr[I+d] to arr[I] and finally store temp at the right place.

> **Example :-**

```
Here is an example for N = 12 and K = 3. GCD of N and K is 3.

Let arr[] be {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}

a) Elements are first moved in first set – (See below  diagram for this movement)

 arr[] after this step --> {4 2 3 7 5 6 10 8 9 1 11 12}

b) Then in second set.

       arr[] after this step --> {4 5 3 7 8 6 10 11 9 1 2 12}

c) Finally in third set.

       arr[] after this step --> {4 5 6 7 8 9 10 11 12 1 2 3}
```



> **Time Complexity:- O(N)**, as worst case because we have shift all elements of the array.

> **Auxiliary Space:- O(1)** , here as one temporary variable is needed.

▸ 5. Reversing An Array

- Reversing an array means reversing the order of the elements in the given array.
- Suppose you have to reverse the following array : -

Iterative Solution

1. Using Temporary Array : -

- The idea is to first copy all of the elements of the given array in a temporary array.
- Then traverse the temporary array from end and replace elements in original array by elements of temp array.

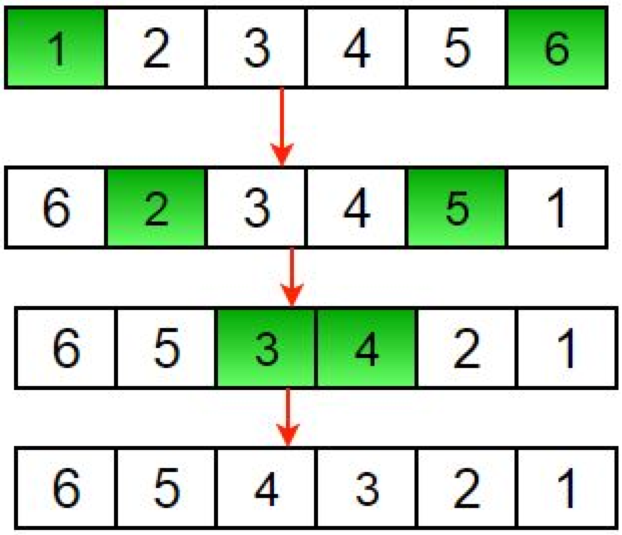
> **Auxiliary Space : - O(N)**

2. Efficient One :-

- The idea is to `traverse the array from both ends and keep swapping elements from both ends until middle of the array is reached`.

> **Algorithm**

```
1) Initialize start and end indexes as
   start = 0, end = N-1
2) In a loop, swap arr[start] with arr[end]
   and change start and end as follows :
       start = start + 1,
       end = end – 1
```



For Reference : -

> **Time Complexity : - O(N)** , as we have to change all the elements.

Recursive Solution

> **Algorithm**

```
1) Initialize start and end indexes as
   start = 0, end = n-1
2) Swap arr[start] with arr[end]
3) Recursively call reverse for rest of the array.
```

> **Recursive Function : -**

```
void reverseArray(arr[], start, end)
{
    if (start >= end)
        return;

    // Swap elements at start and end
    temp = arr[start];
    arr[start] = arr[end];
    arr[end] = temp;

    // Recursive Function calling
```

```
        reverseArray(arr, start + 1, end - 1);
    }
```

> **Time Complexity : - O(N)** , as we have to shift all elements .

> ## Important Concepts

▾ 1.Sliding Window Technique

- This technique `shows how a nested ‘for loop’ in few problems can be converted to a single ‘for loop; and hence reducing time complexity.`

> **Example**

```
Q. Given an array of integers of size 'n'. Our aim is to calculate the maximum sum
of 'k' consecutive elements in the array.

Input  : arr[] = {100, 200, 300, 400}
         k = 2
Output : 700
```

Naïve Approach : -

- We have to find the sum of each k consecutive elements and compare which block has the maximum sum possible .

> **Time Complexity : - O(n*k)**

Sliding Window Technique

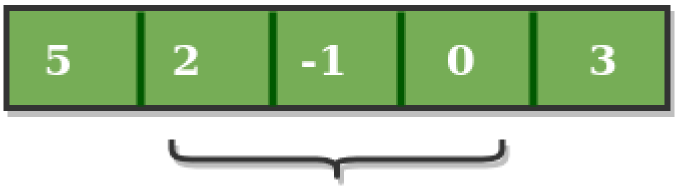
- To understand this topic lets take an example of window pane in bus .
- Consider of window of length n and the pane which is fixed in it of length k;
- Initially the pane is at extreme left i.e., at 0 units from the left.
- Now, co-relate the window with array arr[] of size n and plane with current_sum of size k elements.
- Now, if we apply force on the window such that it moves a unit distance ahead.
- The pane will cover next k consecutive elements.
- Consider an array `arr[] = {5 , 2 , -1 , 0 , 3} and value of k = 3 and n = 5.`

> **Applying sliding window technique : -**

```
1.  We compute the sum of first k elements out of  n terms using a linear loop and
store the sum in variable window_sum.

2.  Then we will graze linearly over the array till it reaches the end and
simultaneously keep track of maximum sum.
```

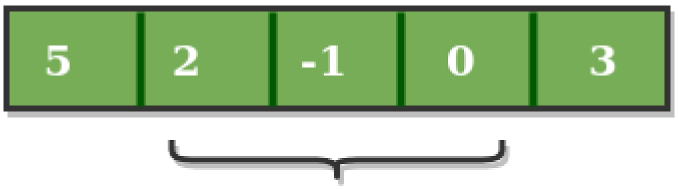
```
3.  To get the current sum of block of k elements just subtract the first element
from the previous block and add the last element of the current block .
```

- This is the initial phase where we have calculated the initial window sum starting from index 0 . At this stage the window sum is 6. Now, we set the maximum_sum as current_window i.e 6.



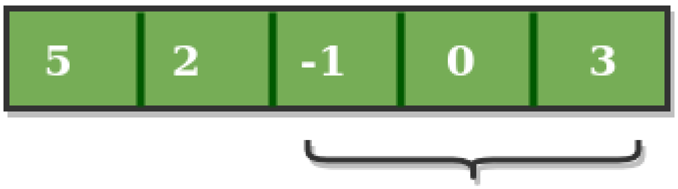
$$current\_sum = window\_sum + (-5) + (0)$$

- Now, we slide our window by a unit index. Therefore, now it discards 5 from the window and adds 0 to the window. Hence, we will get our new window sum by subtracting 5 and then adding 0 to it. So, our window sum now becomes 1.
- Now, we will compare this window sum with the maximum_sum. As it is smaller we wont the change the maximum_sum.



$$current\_sum = window\_sum + (-5) + (0)$$

- Similarly, now once again we slide our window by a unit index and obtain the new window sum to be 2. Again we check if this current window sum is greater than the maximum_sum till now. Once, again it is smaller so we don't change the maximum_sum.



$$current\_sum = window\_sum + (-2) + (3)$$

`Therefore, for the above array our maximum_sum is 6.`

> **Example Code**

```
#include <iostream>
using namespace std;



// Returns maximum sum in a subarray of size k.
int maxSum(int arr[], int n, int k)
```

```
{
    // n must be greater
    if (n < k) {
        cout << "Invalid";
        return -1;
    }

    // Compute sum of first window of size k
    int max_sum = 0;
    for (int i = 0; i < k; i++)
        max_sum += arr[i];

    // Compute sums of remaining windows by
    // removing first element of previous
    // window and adding last element of
    // current window.
    int window_sum = max_sum;
    for (int i = k; i < n; i++) {
        window_sum += arr[i] - arr[i - k];
        max_sum = max(max_sum, window_sum);
    }

    return max_sum;
}

// Driver code
int main()
{
    int arr[] = { 1, 4, 2, 10, 2, 3, 1, 0, 20 };
    int k = 4;
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << maxSum(arr, n, k);
    return 0;
}
```

> **Time Complexity : - O(n)**

▸ 2. Prefix Sum Array

- The prefix sum array of any array, arr[] is defined as an array of same size say, `prefixSum[] such that the value at any index i in prefixSum[] is sum of all elements from indexes 0 to i in arr[].`

> **Example**

```
Input  : arr[] = {10, 20, 10, 5, 15}
Output : prefixSum[] = {10, 30, 40, 45, 60}
```

> **Explanation :**

```
While traversing the array, update
the element by adding it with its previous element.
prefixSum[0] = 10,
prefixSum[1] = prefixSum[0] + arr[1] = 30,
prefixSum[2] = prefixSum[1] + arr[2] = 40 and so on.
```

> **Function To generate prefix sum array :-**

```
void fillPrefixSum(int arr[], int N, int prefixSum[])
{
    prefixSum[0] = arr[0];

    // Adding present element
    // with previous element
    for (int i = 1; i < N; i++)
        prefixSum[i] = prefixSum[i-1] + arr[i];
}
```

Finding Sum in Range : -

- Since the array prefixSum[i] stores the sum of all elements upto i.
- Therefore, prefixSum[j] - prefixSum[i] will give:

```
sum of elements upto j-th index - sum of elements upto i-th element
```

```
sumInRange = prefixSum[j]   , if i = 0
otherwise,
sumInRange = prefixSum[j] - prefixSum[i-1]  ,  if (i != 0)
```

> ## Interview Coding Problems

▾ 1.Beginner Level

▸ 1.Check if a key is present in every segment of size k in an array

- **Given an array arr[] and size of array is n and one another key x, and give you a segment size k. The task is to find that the key x present in every segment of size k in arr[].**

> Example

```
Input :
arr[] = { 3, 5, 2, 4, 9, 3, 1, 7, 3, 11, 12, 3}
x = 3
k = 3
Output : Yes
There are 4 non-overlapping segments of size k in the array, {3, 5, 2}, {4, 9, 3},
{1, 7, 3} and {11, 12, 3}. 3 is present all segments.
I
```

- Solution

▸ 2.Find the frequency of a number in an array.

- **Given an array, a[], and an element x, find a number of occurrences of x in a[]**.

> Example

```


Input  : a[] = {0, 5, 5, 5, 4}
           x = 5
Output : 3

Input  : a[] = {1, 2, 3}
          x = 4
Output : 0
```

- Solution

▸ 3.Range and Coefficient of range of Array

- **Given an array arr of integer elements, the task is to find the range and coefficient of range of the given array where: Range: Difference between the maximum value and the minimum value in the distribution. Coefficient of Range: (Max – Min) / (Max + Min)**

> Example

```
Input: arr[] = {15, 16, 10, 9, 6, 7, 17}
Output: Range : 11
Coefficient of Range : 0.478261
Max = 17, Min = 6
Range = Max - Min = 17 - 6 = 11
Coefficient of Range = (Max - Min) / (Max + Min) = 11 / 23 = 0.478261
Input: arr[] = {5, 10, 15}
Output: Range : 10
Coefficient of Range : 0.5
```

- Solution

▸ 4.Sort an array of 0s, 1s and 2s

- **Given an array A[] consisting 0s, 1s and 2s. The task is to write a function that sorts the given array. The functions should put all 0s first, then all 1s and all 2s in last.**

> Examples

```
Input: {0, 1, 2, 0, 1, 2}
Output: {0, 0, 1, 1, 2, 2}

Input: {0, 1, 1, 0, 1, 2, 1, 2, 0, 0, 0, 1}
Output: {0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 2, 2}
```

- Solution

▸ 5.Union and Intersection of two sorted arrays

- **Given two sorted arrays, find their union and intersection**

> Example

```
Input : arr1[] = {1, 3, 4, 5, 7}
        arr2[] = {2, 3, 5, 6}
Output : Union : {1, 2, 3, 4, 5, 6, 7}
         Intersection : {3, 5}

Input : arr1[] = {2, 5, 6}
        arr2[] = {4, 6, 8, 10}
Output : Union : {2, 4, 5, 6, 8, 10}
         Intersection : {6}
```

- Solution

▸ 2.Intermidiate Level

▸ 1.Program to cyclically rotate an array by one

- **Given an array, cyclically rotate the array clockwise by one.**

> Example

```
Input:  arr[] = {1, 2, 3, 4, 5}

Output: arr[] = {5, 1, 2, 3, 4}
```

- Solution

▶ 2.Minimum number of jumps

- **Given an array of N integers arr[] where each element represents the max number of steps that can be made forward from that element. Find the minimum number of jumps to reach the end of the array (starting from the first element). If an element is 0, then you cannot move through that element.**

> Example:

```
Input:
N = 11
arr[] = {1, 3, 5, 8, 9, 2, 6, 7, 6, 8, 9}
Output: 3
Explanation:
First jump from 1st element to 2nd
element with value 3. Now, from here
we jump to 5th element with value 9,
and from here we will jump to last.
```

Task : -

> You don't need to read input or print anything. Your task is to complete function minJumps() which takes the array arr and it's size N as input parameters and returns the minimum number of jumps.

Expected Time Complexity: *O(N)*

Expected Space Complexity: *O(1)*

Constraints:

```
1 ≤ N ≤ 107

0 ≤ ai ≤ 107
```

- Solution

▶ 3.Factorials of large numbers.

- **Given an integer N, find its factorial.**

> Example

```
Input: N = 10
Output: 3628800

Explanation :
10! = 1*2*3*4*5*6*7*8*9*10 = 3628800
```

- **Expected Time Complexity :** *O(N)*
- **Expected Auxilliary Space :** *O(1)*
- **Constraints:** 1 ≤ N ≤ 1000
- Solution

▶ 4.Longest consecutive subsequence.

- **Given an array of positive integers.**
- **Find the length of the longest sub-sequence such that elements in the subsequence are consecutive integers, the consecutive numbers can be in any order.**

> Example

```
Input:
N = 7
a[] = {1,9,3,10,4,20,2}
Output: 4

Explanation:
1, 2, 3, 4 is the longest
consecutive subsequence.
```

- **Expected Time Complexity:** *O(N).*
- **Expected Auxiliary Space:** *O(N).*
- **Constraints:**

```
1 <= N <= 105
```

```
0 <= a[i] <= 105
```

- Solution

▸ 5.Sort an array in wave form

- **Given an unsorted array of integers, sort the array into a wave like array.**
- **An array 'arr[0..n-1]' is sorted in wave form if arr[0] >= arr[1] <= arr[2] >= arr[3] <= arr[4] >= .....**

> Example

```
Input:  arr[] = {10, 5, 6, 3, 2, 20, 100, 80}
 Output: arr[] = {10, 5, 6, 2, 20, 3, 100, 80} OR
                 {20, 5, 10, 2, 80, 6, 100, 3} OR
                 any other array that is in wave form


Input:  arr[] = {20, 10, 8, 6, 4, 2}
 Output: arr[] = {20, 8, 10, 4, 6, 2} OR
                 {10, 8, 20, 2, 6, 4} OR
                 any other array that is in wave form
```

- Solution

▸ 3.Advanced Level

▸ 1.Trapping Rain Water

- **Given n non-negative integers representing an elevation map where the width of each bar is 1, compute how much water it is able to trap after raining.**

> Example

```
Input: arr[]   = {2, 0, 2}
Output: 2
```

- Solution

▸ 2.Chocolate Distribution Problem

- ☐ **of positive integers of size N, where each value represents the number of chocolates in a packet.**

- **Each packet can have a variable number of chocolates. There are M students, the task is to distribute chocolate packets among M students such that :**

1. Each student gets exactly one packet.
2. The difference between maximum number of chocolates given to a student and minimum number of chocolates given to a student is minimum.

> Example

```
Input:
N = 7, M = 3
A = {7, 3, 2, 4, 9, 12, 56}

Output: 2

Explanation:
The minimum difference between
maximum chocolates and minimum chocolates
is 4 - 2 = 2 by choosing following M packets :
{3, 2, 4}.
```

- **Expected Time Complexity:** *O(N*Log(N))*

- **Expected Auxiliary Space:** *O(1)*

- **Constraints:**

```
1 ≤ T ≤ 100
1 ≤ N ≤ 105
1 ≤ Ai ≤ 109
1 ≤ M ≤ N
```

- Solution

▶ 3.Find the median

- **Given an array arr[] of N integers, calculate the median.**

> Example

```
Input: N = 4
arr[] = 56 67 30 79â€‹

Output: 61
Explanation: In case of even number of
```

```
elemebts average of two middle elements
is the median.
```

- **Expected Time Complexity:** *O(n * log(n))*
- **Expected Space Complexity:** *O(1)*
- **Constraints:**

```
1 <= Length of Array <= 100
1 <= Elements of Array <= 100
```

- Solution

▶ 4.Longest Alternating subsequence

- A sequence {x1, x2, .. xn} is alternating sequence if its elements satisfy one of the following relations :

```
  x1 < x2 > x3 < x4 > x5 < …. xn or
  x1 > x2 < x3 > x4 < x5 > …. xn
```

> Example

```
Input: arr[] = {1, 5, 4}
Output: 3
The whole arrays is of the form  x1 < x2 > x3


Input: arr[] = {1, 4, 5}
Output: 2
All subsequences of length 2 are either of the form
x1 < x2; or x1 > x2


Input: arr[] = {10, 22, 9, 33, 49, 50, 31, 60}
Output: 6
The subsequences {10, 22, 9, 33, 31, 60} or
{10, 22, 9, 49, 31, 60} or {10, 22, 9, 50, 31, 60}
are longest subsequence of length 6.
```

- Solution

▶ 5.Stock Buy Sell to Maximize Profit

- **The cost of a stock on each day is given in an array, find the max profit that you can make by buying and selling in those days.**
- **For example, if the given array is {100, 180, 260, 310, 40, 535, 695}, the maximum profit can earned by buying on day 0, selling on day 3. Again buy on day 4 and sell on day 6.**
- **If the given array of prices is sorted in decreasing order, then profit cannot be earned at all.**
- Solution

Questions Asked In Interview

▸ Advantages and Disadvantages of Array?

Advantages : -

- We can put in place other data structures like stacks, queues, linked lists, trees, graphs, etc. in Array.
- Arrays can sort multiple elements at a time.
- We can access an element of Array by using an index.

Disadvantages : -

- We have to declare Size of an array in advance. However, we may not know what size we need at the time of array declaration.
- The array is static structure. It means array size is always fixed, so we cannot increase or decrease memory allocation.

- The key benefit of an array data structure is that it offers fast **O(1)** search if you know the index, but **adding and removing**an element from an array is slow because**you cannot change the size of the array once it's created.**

▸ Can we use Generics with the array?

- **No, we cannot use Generic with an array.**

▸ Where is the memory allocated for arrays in Java?

- In Java, memory for arrays is always allocated on the **heap** as arrays in Java are objects.

▸ What is the difference between array_name and &array_name?

- To understand this question let's take an example, suppose arr is an integer array of 5 elements.

> int arr[5];

- If you print arr and &arr then you found the same result but both have different types.

arr	& arr

arr	& arr
arr=> The name of the array is a pointer to its first element. So here arr split as the pointer to the integer.	&arr=> It split into the pointer to an array that means &arr will be similar to int(*)[5];

▶ What are the advantages of using an array of pointers to string instead of an array of strings?

- An array of pointers to string is useful when sorting the strings, we need to **swap only pointers instead of swapping the whole string** which helps in the efficient use of memory and time.

▶ What is a flexible array in C?

- A very good feature that is introduced by C99 is a flexible array member.
- This feature enables the user to create an empty array in a structure, the size of the empty array can be changed at runtime as per the user requirements.
- This empty array should be declared as the last member of the structure and the structure must contain at least one more named member.

Rules For Creating Flexible Array : -

- The flexible array member must be the last member of the structure.
- There must be at least one other member.
- The flexible array is declared like an ordinary array, except that the brackets are empty.

> Example

```
struct userData
{
    int iTrackNumber;

    float fAmount;

  //flexible array member
    char acAddress[];
};
```

▶ Can we create an array of a void type?

- **NO**