Array

- Array is a list of Elements of the same type placed in a contiguous memory location

Linear Search :

- Finding the index of any given index.

▼ Binary Search

- Binary Search is always applied on a Sorted Array
- It can be either in increasing order or decreasing order
- example dictionary
- **LOGIC** : Selecting the middle value of the array and comparing it to the key , if the value is greater than the key than we would consider the LEFT side of the array in case of Ascending ordered array or if the key is greater than the middle value we would then further consider the RIGHT half of the array.
- The logic would be again applied to the remaining array until the middle value is equal to the key itself.
- **CONDITION** :
  - *Start ≤ end :::::* Loop condition
  - *mid = (Start + End)/2*
  - *mid == key* :: Found key
    - In the case that array does not exist , return -1
- *Time Complexity of Binary Search< Linear Search*

▼ Reverse an Array :

- **Bruteforce Approach** : A loop Starting from the last index of the array and adding the values into a new Array .
  - Time Complexity : O(n) It is because the loop would run until ‘n’
  - Space Complexity : O(n) , It is because the new formed array would take up space equal to the length of ‘n’
- **Optimized Approach : A loop in which we exchange the values of the first and last index of an array , simultaneously moving towards the center of the array i.e. +1 from the start and -1 from the end.**
- The swapping is done with the help of a temp variable i.e. the last value is first stored inside the temp variable and then the first value is stored inside the last index after which the temp value is moved to the first index .
  - Time Complexity : O(n)

▼ Pairs in an Array :

Finding all possible pairs of any given array

- Nested loop would be used
- First loop to print the first digit of the pair
- The second loop is to print the second digit of the pairs
  - Time Complexity : O(n^2) because of the nested loops

▼ SubArrays :

Print SubArrays

- A continuous part of array
- For an array of size ‘n’ , the total number of sub-arrays present are $n(n+1)/2$
- Nested loops would be used
  - Loop one would be to find the first value

- Loop two would be used to find the end value
- Loop three would be used to Print the values in between First and Last

▼ Maximum SubArray Sum :

Print maximum sum of the SubArrays.

▼ Brute force Method :

- Creating a maximum sum and a current sum (i.e. Sum of all SubArrays one by one) variable and comparing those values again until maximum sum is greater than current sum.
- Same as the print subarray code but instead of printing we create two variable and initialize ***current_sum*** with 0 and ***maximum_sum*** with ***Interger.MIN_Value***
- Then adding the current sum with itself

```
current_sum += numbers[k];
```

- Then comparing and updating the value of the maximum_sum

```
if (maximum_sum < current_sum) {
                        maximum_sum = current_sum;
                    }
```

- Time Complexity : $O(n^3)$

▼ **Prefix Approach ||** Optimized Approach :

- In the Approach we store the sum of all sub arrays upto the index
- i.e. we create different variables and store the values in the respective indexes
- prefix of i-1 is the sum of all the indexes before i and then the sum of i is also added to create current sum.

```
Current index sum = prefix(i-1) + array[i]
```

- Prefix [end] - prefix [start -1]
- Creating a loop to store the values of all subarrays one by one in ascending order

- Time Complexity : $O(n^2)$ because one loop is replaced by

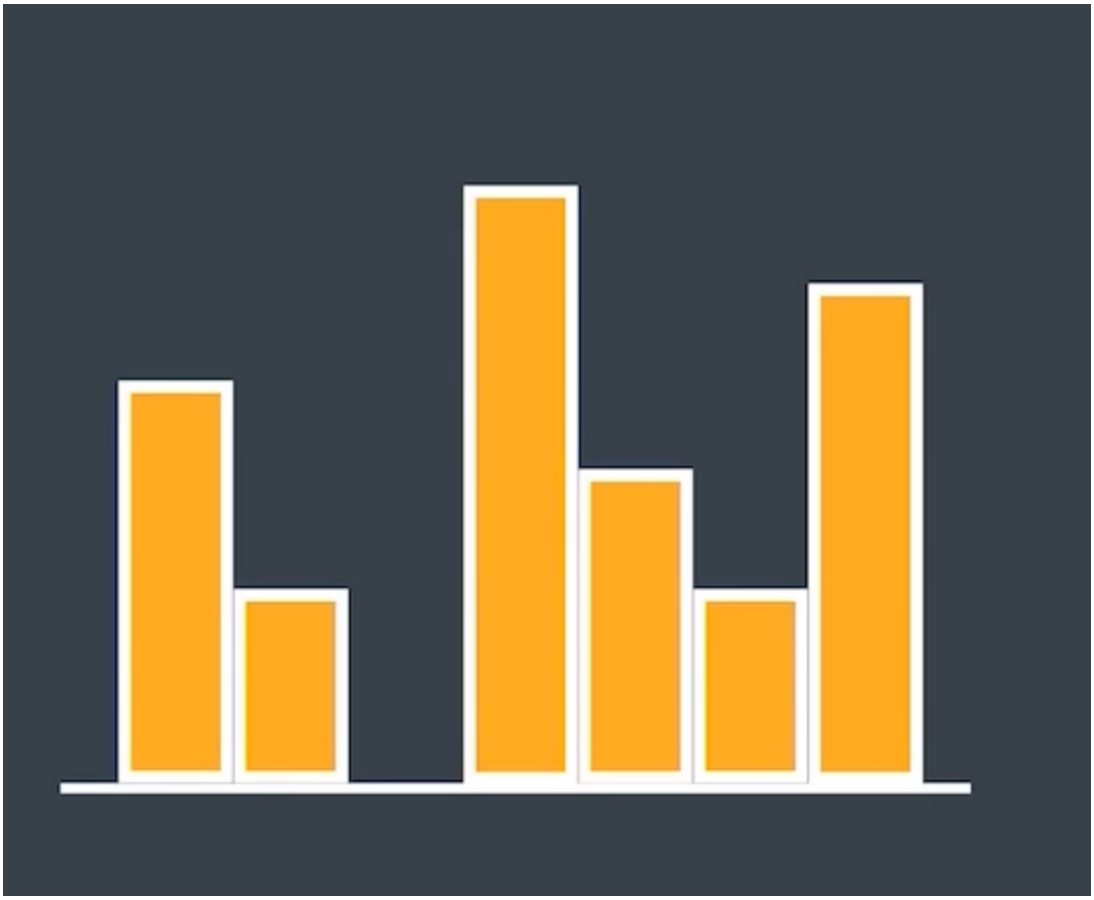
```
current_sum = first ==0 ? prefix[last] : prefix[last] - prefix [first -1];
```

▼ **Kadan's Algorithm || Most Optimized Algorithm**

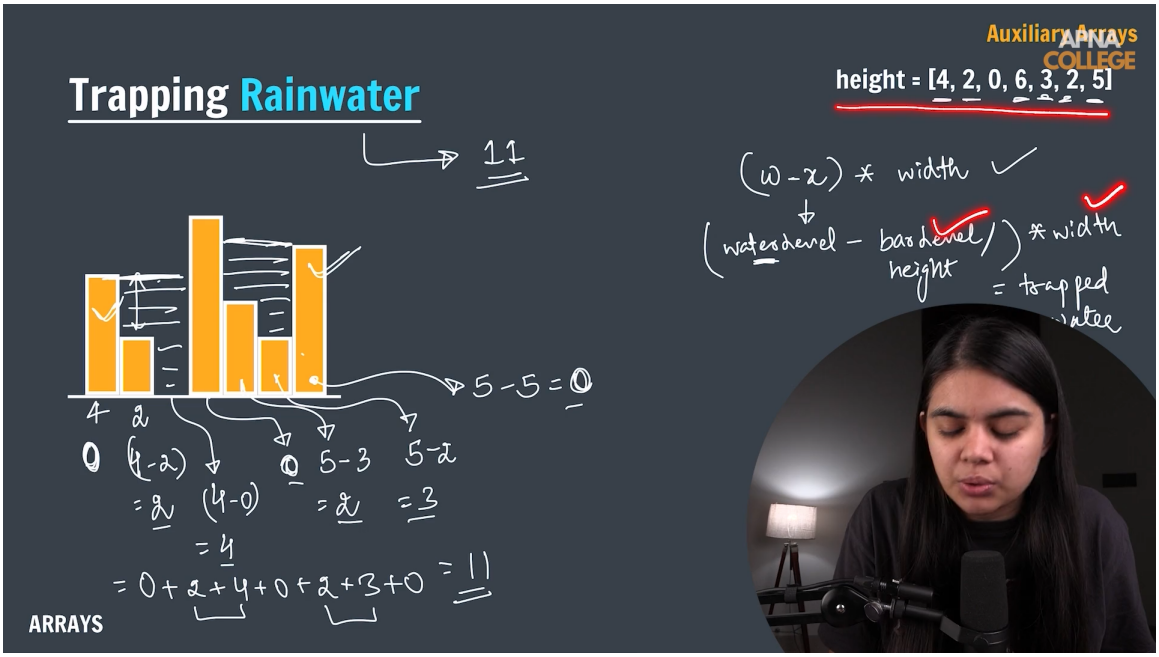
- Favorable Situation :
  - In case of two values ,
    - If we add two Positive number = Favorable Situation
    - If the add a very greater Positive number to a very small negative number = Favorable Situation
    - But, if the add a very small Positive number to a very greater negative number = Not a favorable Situation because the outcome would be a Negative number
      - Rather than taking the negative value as the current sum we mark current sum as zero
- **Time Complexity** : $O(n)$ because only one loop is used

▼ Trapping Rainwater Problem : Medium Level Problem

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

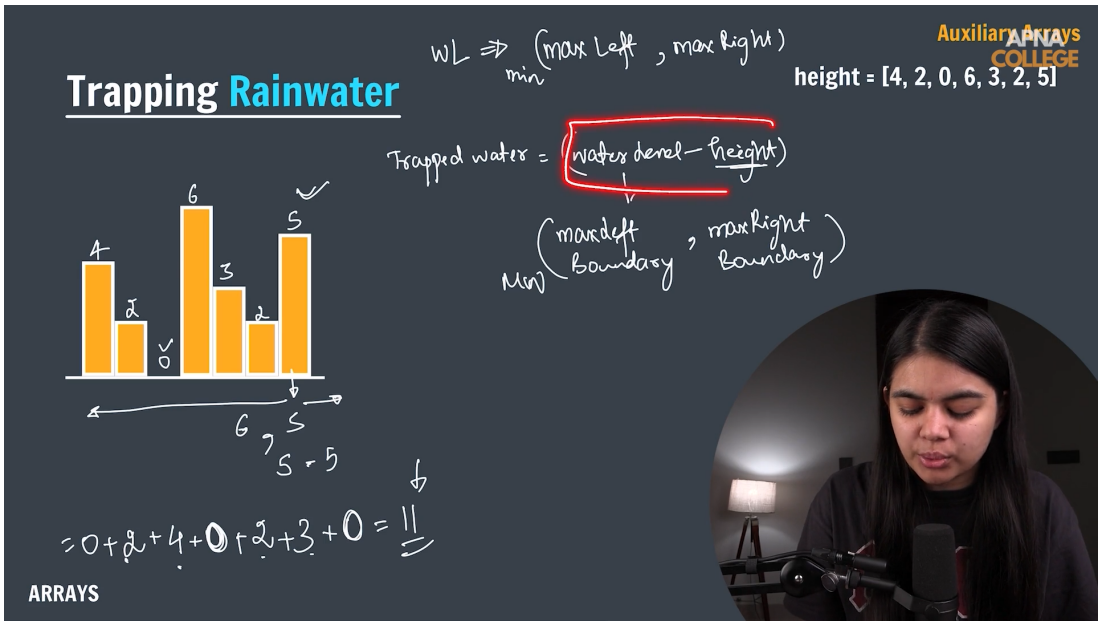


- The bars of the graph are resembled as buildings
  - Rainwater gets trapped inside the space between the bars
  - To find : Volume of the water trapped
  - Water level - Bar level (Height) * Width



Different cases to consider :

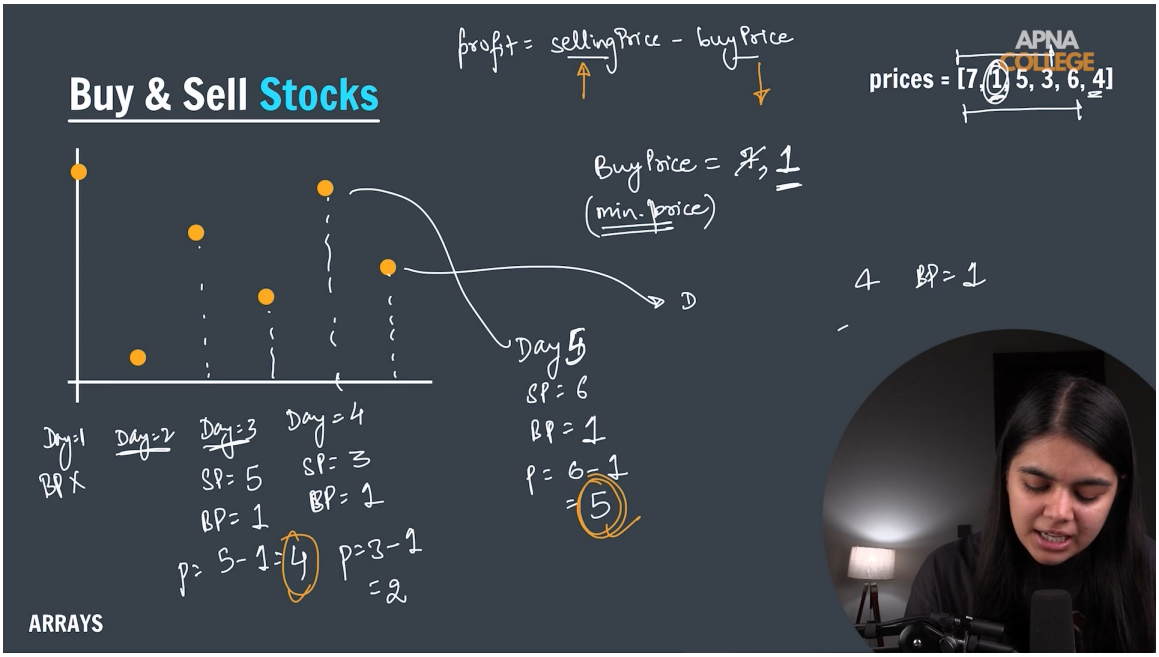
- Single Bar : No water to be stored
- Two Bars : No water would get trapped
- Three Bars : Water would get trapped
    - Bars arranged in Ascending or descending order : No water would be trapped
- Water would only get trapped if there is a pocket inside two bars its the bar at the center is less in height than the bars on the corners
    - In such cases the height of the water stored would be the height of the shortest/minimum surrounding bars height
    - The maximum amount of water stored would depend on the maximum height boundary on both sides and then choose the minimum of them
- Trapped water = (water level - height of current Bar) * width
    - Water level = minimum of (Max Left boundary , Max Right boundary)



- Auxiliary Arrays : Helper Arrays
  - To calculate left and right max boundary
  - Creating two arrays (Left max and Right max) and storing left and right max values in the arrays
  - Then compare the index values and chose the minimum value and then minus the height of the current bar.
- **Time Complexity** : $O(n)$

▼ Buy sell stocks Problem :

- To find maximum profit by selling at highest and buying at the lowest
- define Buy price variable which tracks the lowest buying price
- **Time Complexity** : $O(n)$


Buy & Sell Stocks
profit = sellingPrice - buyPrice
prices = [7, 1, 5, 3, 6, 4]
APNA COLLEGE
Buy Price = 7, 1
(min. price)
Day 1
BP X
Day 2
Day 3
SP = 5
BP = 1
p = 5 - 1 = 4
Day 4
SP = 3
BP = 1
p = 3 - 1 = 2
Day 5
SP = 6
BP = 1
p = 6 - 1 = 5
4 BP = 1
ARRAYS