★ Merge two sorted array without using extra space.

arr1[] = {1,3,5,7}   arr2[] = {0,2,6,8,9}

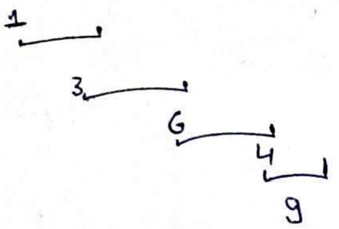
Vector Ans ✓   push arr1 element, push arr2 element   then sort the array

. 1,3,5,7, 0,2,6,8,9   then

(for i=0 i<n: i++) arr1[i] = s[i];   for (i=0 i<m i++) arr2[i] = s[i+n]

★ KADANE'S ALGO

1 2 3 -2 5



$\boxed{Ans = 9}$

~~Max = arr[0]~~ Maxi = INT-MIN , currsum = 0.

curr sum all elements one by one.

. maxi = max(maxi, curr sum);

if currSum < 0 , currSum = 0

★ MERGE INTERVALS

$\boxed{\text{Vector ans}}$ -   size == 0 return ans.

[[1,3], [2,6], [8,10], [15,18]]

[[1,6], [8,10], [15,18]]

sort the intervals

push first like push (intervals[0]) now take a pointer j = 0;

```
for (i=1; i< size; i++) {
    if ( ans[j][1] >= intervals[i][0])
        ans[j][1] = max (ans[j][1], intervals[i][1])
    else
        j++
        push intervals[i]
return ans.
```

1,3	2,6	8,10	8,9	9,11	15,18	2,4	16,17
1,3	2,4	2,6	8,9	8,10	9,11	15,18	16,17

T.C. (NlogN) + (N)

★ NEXT PERMUTATION

Find largest index K such that nums[K] < nums[K+1]

Find index l > K such that nums[K] < nums[l]

Swap nums[K] , nums[l]

Reverse the subarray

(begin + K+1, end)

```
int n = size, k, l
for (k = n-2 . . . ) if
    (nums[k] < nums[k+1]) break;
if (k<0){
    reverse (nums.begin(), nums.end())
}
else {
    l = n-1 .....){ if
    nums[l] > nums[k]
        break;
swap( nums[k], nums[l] )
reverse ( nums + k+1 , nums.end() ) ; ✓
```

★ BEST TIME TO BUY AND SELL STOCK

OPTIMAL

Prices = [7, 1, 5, 3, 6, 4] Profit = 0;

↑ minprice

```
for (i=1; i<n; i++){
minprice = min( minprice, prices[i]);
profit = max (profit, prices[i] - minPrice)
return Profit
```

example = [3,5,1,7,4,9,3]

check for 3 , for everyone ,
2 , -2 . . . etc.

[3,1,4,8,7,2,5]

aux = [8, 8, 8, 8, 7, 5, 5]

maxprofit = 8 7

check minSoFar , maxprofit

Brute
```
for(buy) {
    for(sell) {

    }
}
```

②
```
for (i = 0 i < size - 1; i++)
for (j = i+1; j<size ; j++)
    profit = price[j] - price[i]
    if (profit > maxprofit)
        maxprofit = profit

return maxprofit
```

COUNT PAIRS WITH GIVEN SUM

use map

{1,5,7,1}

$\nearrow$ 2Ans

[1+5 = 6
  5+1 = 6]

```
unordered map < int, int> m

for(i=0 ; i<n; i++) {
    int b = k - [arr[i]]        }  Check if it can be combined with other element
    if (m[b]) {
        ans += m[b]
    }else
        m[arr[i]]++;
                                return ans
```
store frequency

COMMON ELEMENT IN 3 ARRAYS

CREATE THREE maps and store their element now create ans vector ,

if (present) then pushback that element

WAYS OF WRITING FUNCTION IN JS

- Function Declaration
- Function Expression
- Arrow Function

* declaring with a keyword "function".
* Define using a variable and store returned value in a variable

Ex const add = funtion(a,b) {Console.log(a+b); }
add (2,3);

* Introduced in ES6 version use the => symbol

```
let add = (a,b) => (a+b);
      console.log(add (3,2)).
```

FIRST CLASS CITIZEN

Ability to treat function as values, to pass them as arguments and to return a function from another function then it is said that it has first class functions and functions are called first class citizen

1. Ability to treat function as values

2. ~~Pass function as arguments~~

eg

```
function teacher(){              function greet (user) {
   return teacher;                  console.log ("Welcome", user());
}

         Var message = greet (teacher); ✓
```

3. Ability to return a function from another function

```
Var greet = function() {
            return function(){
               console.log ("Hello"); } }
greet ()();
```

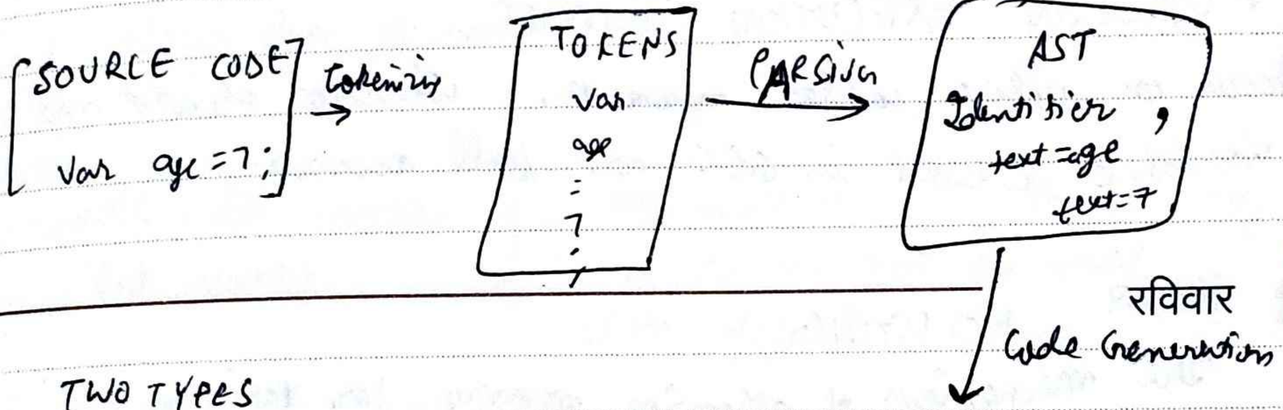
रवि.	सोम.	मंगल.	बुध.	गुरु.	शुक्र.	शनि.
30						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29

A LEXICAL ENVIORNMENT determins how and where we write our code physically

function doSomethig(){
    Var age=7;
}

Variable age is lexically inside fuction dosomethig.

* There will be more than one lexical enviornmt However not all executed at once.
* enviornment that helps the code get execued is called EXECUTON CONTEXT.



TWO TYPES
- Global Execution Context.
- Function Execution Contest

CREATION AND EXECUTION PHASE
- A global object called window
- A global varible call this
- Variable get initialized with a unique value called undefined.

WHAT HAPPENS IN EXECUTION PHASE

* G.E.C gets created when we load file,
* Creates two special things window object and this
* In G.E.C both are equal

रवि	सोम	मंगल	बुध	गुरु	शुक्र	शनि
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30				

Example

```
Var blog = 'freeCodeCamp';
function logblog() {
    Console.log(this.blog);
}
```

CREATON

* Global object window and variable this gets created.
* Memory gets allocated for blog and logblog
* Blog initialized by undefined

EXECUTION

→ Value freeCodecamp is assigned to variable blog.

FUNCTION EXECUTION CONTEXT

* Access to value called arguments. Window object and this variable created in GEC are still accessible.

HOISTING

The mechanism of allocating memory for variables and initializing with the value defined at the execution context's creation phase is called Variable Hoisting.

example

```
console.log(name);
    Var (name);
```

CREATION

Memory allocate for variable name

Special value undefined assigned

EXECUTION

console.log executes.

* HOISTING is only for function declaration not initialization.
* Define variables and functions before using them to reduce chances of errors

HOISTING IN FUNCTION

* execution context creates memory for fuction and puts the entire fuction of declaration of

```
Ex    logMe();
      Var logMe = function() {
          console.log ('logging');
      }
```

Give errors, because with function initialization the Variable logMe will be hoisted as a variable, not as a fuc^n. so with ~~me~~ variable hoisting memory allocation will happen with initialization of undefined

Also

```
console.log (name);
  let name;
```

Reference Error: In this case they will be hoisted but not assigned with default undefined.