

# Pascal's Triangle



## Pascal Triangle

1		1	1			
2		1	1	1		
3		1	2	1		
4		1	3	3	1	
5		1	4	6	4	1
6	1	5	10	10	5	1

(1) Given  $R \& C$ ,  
tell the element at that  
place

$$R = 5 \quad C = 3$$

$$ans = 6$$

(3) Given  $N$ , print the entire  
triangle

$$N = 6$$

(2) Print any  $N^{th}$  row  
of pascal triangle

$$N = 5$$

$$1 \ 4 \ 6 \ 4 \ 1$$



Fibonacci triangle

(1) Given R = 8 C = 3  
tell me element at next place

R = 5 C = 3  
ans = 6

(2) Print any Nth row of Pascal triangle

N = 5  
1 4 6 4 1

(3) Given N, print entire triangle N = 6

R - 1      C  
C - 1

${}^n C_r = \frac{n!}{r! \times (n-r)!}$

${}^4 C_2 = \frac{4! \times 3! \times 2!}{(2! \times 1) (2! \times 1)}$   
= 6



Calculator icon bar:

7C<sub>2</sub> =  $\frac{7!}{2! \times (5)!} = \frac{7 \times 6 \times (5 \times 4 \times 3 \times 2 \times 1)}{2 \times 1 \times (5 \times 4 \times 3 \times 2 \times 1)}$

$= \frac{7 \times 6}{2 \times 1}$

10C<sub>3</sub> =  $\frac{10 \times 9 \times 8}{3 \times 2 \times 1} = \frac{10}{1} \times \frac{9}{2} \times \frac{8}{3}$

$$\frac{100!}{3 \times 2 \times 1} = \frac{10!}{1} \times \frac{9!}{2} \times \frac{8!}{3}$$

fun fact(n, r)

{

res = 1

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

{

res = res \* (n - i)

TC  $\rightarrow O(n)$   
SC  $\rightarrow O(1)$

res = res / (i + 1);

long long

return res;

}



4 1 3 3 1 0 ~~1~~

5 1 4 6 4 1

6 1 5 10 10 5 1

— — — — — — —

$N^{th}$  row  $\rightarrow N$  elements

$n-1 \quad c \quad c-1$

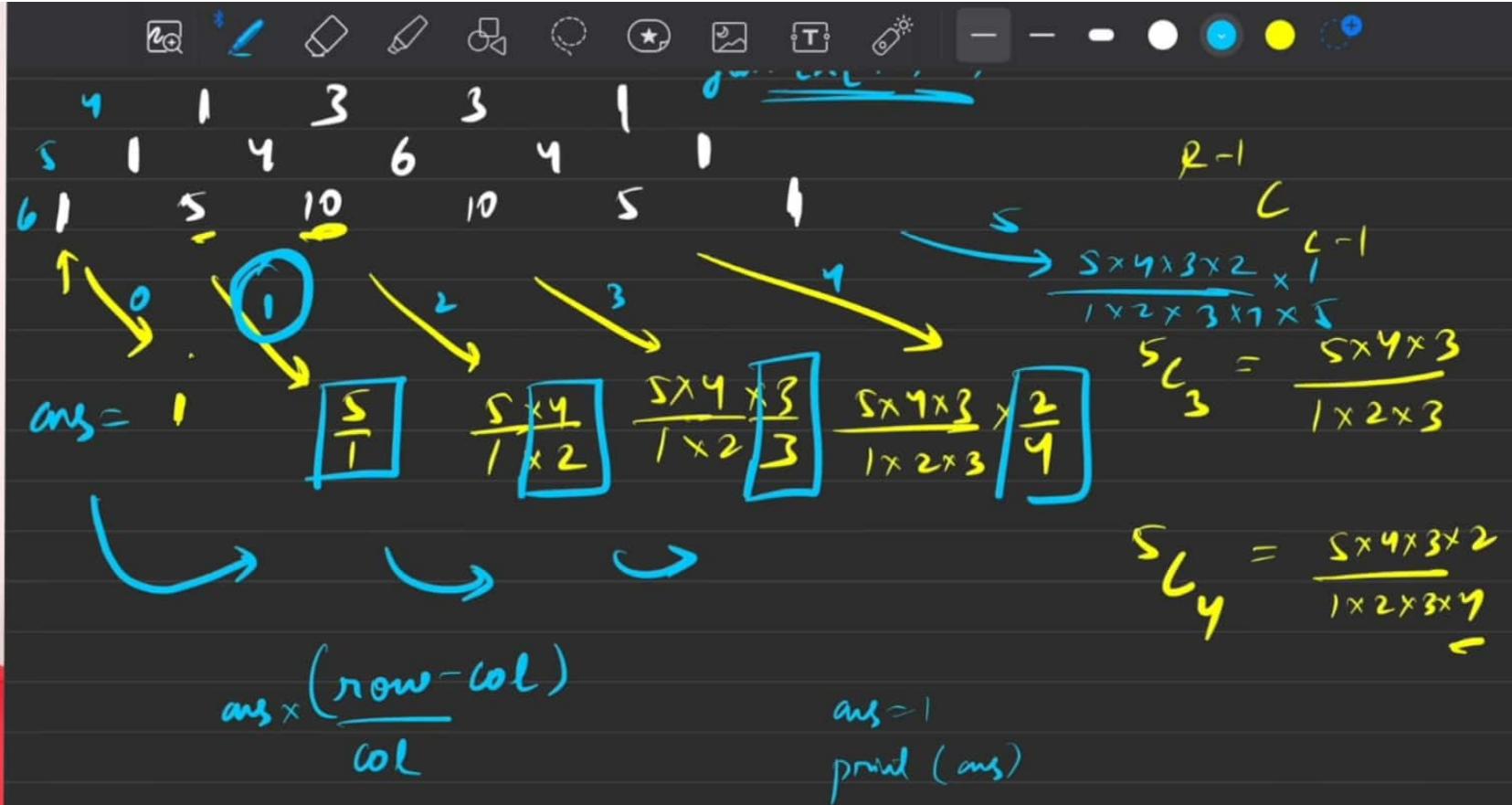
$\text{fun}(\underline{c=1}; \underline{cc=n}; c++)$

print  $(\underline{\text{fun}N(n-1, c-1)})$ )

{

$T.C \rightarrow O(N \times n)$

A digital note-taking interface showing handwritten code and calculations. At the top, there's a toolbar with various icons. Below it, a grid of numbers is written in blue and black ink. To the right of the grid, the text "N<sup>th</sup> row → N elements" is written in yellow. Further down, there's a recursive function definition in yellow ink: "fun(c=1; cc=n; c++)". Below that, the "print" statement is also in yellow, with the entire "print" line underlined. A large yellow curly brace is positioned to the left of the closing parenthesis of the print statement. At the bottom, the time complexity is given as "T.C → O(N × n)".





6 1 2 3 4 5 7 8 9

ans = 1

$\frac{5 \times 4 \times 3 \times 2}{1 \times 2 \times 3 \times 1} \times \frac{6}{5} \times \frac{7}{6} \times \frac{8}{7} \times \frac{9}{8}$

$S_{C_3} = \frac{5 \times 4 \times 3}{1 \times 2 \times 3}$

$S_{C_4} = \frac{5 \times 4 \times 3 \times 2}{1 \times 2 \times 3 \times 4}$

ans  $\times \frac{(row - col)}{col}$

ans = 1  
print (ans)

```
if (i = 1; i < n; i++)  
    ans = ans  $\times \frac{(n - i)}{i}$   
    ans = ans / (i)  
print (ans);
```



1 1 2 1  
1 3 3 1       $n-1$   $c_{l-1}$   
1 4 6 4 1  
1 5 10 10 5 1  
 $ans = 53$

fun(  $row = 1 \rightarrow n$  )  
{     $+tempList = 53$   
    fun(  $col = 1 \rightarrow row$  )  
    {  
         $+tempList . add( n(n(row-1, col-1)) )$   
    }  
     $ans . add(+tempList)$   
}  
 $ref(ans)$



Diagram illustrating a recursive function call:

```
graph TD; A[1 1 2 1] --> B[1 3 3 1]; B --> C[n-1 c_{l-1}];
```

Handwritten code analysis:

```
ans = 53
n = O(n × n × n)
n = O(n³)
```

Function definitions:

```
fun (row = 1 → n)
    tempAns = 53
    fun (col = 1 → row)
        tempAns · add (n(n(row-1, col-1)))
    end
    ans · add (tempAns);
end (ans)
```



Pascal Triangle

(1) Given R & C  
tell me element at that place

R=5    C=3

ans = 10

$n-1 \binom{C}{L-1}$

$\Theta(n)$

(2) Print any Nth row of pascal triangle

N=5

1 4 6 4 1

$\Theta(n)$

(3) Given N, print the entire triangle

N=6

$\Theta(n^2)$

$\Theta(n^2) \approx \underline{\underline{\Theta(n^2)}}$



codestudio  
POWERED BY CODING NINJAS

Guided Paths Contests Interview Prep Practice Resources

Topics Problem Submissions Solution New

Current Submission

Correct Answer Test Cases EXP: 12/12 Penalty C++  
few secs ago 40/40 0%

Did you find these test cases useful?

Previous Submissions All languages

Status Test cases EXP Penalty Language

```
vector<int> generateRow(int row) {
    long long ans = 1;
    vector<int> ansRow;
    ansRow.push_back(1);
    for(int col = 1; col < row; col++) {
        ans = ans * (row - col);
        ans = ans / (col);
        ansRow.push_back(ans);
    }
    return ansRow;
}
vector<vector<int>> pascalTriangle(int N) {
    vector<vector<int>> ans;
    for(int i = 1; i <= N; i++) {
        ans.push_back(generateRow(i));
    }
    return ans;
}
```

< Prev Next >

View hints

Last saved on 4:48:05 AM

Run

A screenshot of a programming environment on codestudio. The interface shows a navigation bar with 'Guided Paths', 'Contests', 'Interview Prep', 'Practice', and 'Resources'. Below this is a tabs bar with 'Topics', 'Problem', 'Submissions' (which is highlighted in orange), and 'Solution'. A 'New' button is also visible. On the left, there's a sidebar for 'Current Submission' showing it's a 'Correct Answer' with 12/12 test cases passed, 0% penalty, and C++ language. It was submitted 'few secs ago'. Below this is a section for 'Previous Submissions' with a dropdown for 'All languages'. The main area contains two C++ functions: 'generateRow' which generates a single row of Pascal's triangle, and 'pascalTriangle' which generates the entire triangle up to row N. The code uses vectors and basic arithmetic operations. At the bottom, there are navigation buttons for 'Prev' and 'Next', a 'View hints' button, and a status message indicating the code was last saved at 4:48:05 AM. A 'Run' button is also present.

# Next Permutation



Next Permutation

arr:  $\{1, 2, 3\}$

Brute

Better

Optimal

6 ways

The diagram illustrates the concept of generating the next permutation of a given array. It shows a 3x3 grid of permutations of the array [1, 2, 3]. A red bracket highlights the first two columns of the grid. Above the grid, the array is shown as  $\{1, 2, 3\}$ . An arrow points from the array to the first column of the grid. Another arrow points from the first column to the second column. To the right of the grid, the array is shown as  $\underline{1} \rightarrow \underline{2} \rightarrow \underline{3}$ , with arrows indicating a sequence. Below this, the array is shown as  $\underline{3} \rightarrow \underline{1} \rightarrow \underline{2}$ . A large red oval encloses the entire sequence of arrays and arrows. The text "6 ways" is written inside this oval. To the right of the oval, the words "Brute", "Better", and "Optimal" are listed vertically, with arrows pointing downwards between them.

1	2	3
1	3	2
2	1	3
2	3	1
3	1	2
3	2	1



Next Permutation

arr [ ] = [ 3 1 2 ]

Brute

123 < 132 < 213

↓

optimal

1 2 3  
1 3 2  
2 1 3  
2 3 1  
3 1 2  
3 2 1

A digital whiteboard interface showing handwritten notes. At the top is a toolbar with various icons. Below it, the title "Next Permutation" is written in blue ink. A line of code "arr [ ] = [ 3 1 2 ]" is shown below the title. To the right, the text "Brute" is written above a red underlined comparison "123 < 132 < 213", with a blue arrow pointing down from "Brute" to the comparison. Further down, the word "optimal" is written in blue ink. On the left side, there is a list of six permutations of the numbers 1, 2, and 3, each row consisting of three numbers separated by spaces. The first two rows have the numbers colored yellow, while the last four rows have them colored blue.



Next Permutation

arr [ ] = [ 3 1 2 ]

ans → [ 3 , 2 , 1 ]

Brute  
↓  
Better  
↓  
Optimal

1	2	3
1	3	2
2	1	3
2	3	1
1	3	2
3	2	1

A digital note-taking interface showing handwritten text and diagrams. The title 'Next Permutation' is underlined in blue. Below it, 'arr [ ] = [ 3 1 2 ]' is written. To its right, 'ans → [ 3 , 2 , 1 ]' is shown, with 'Brute' and a downward arrow above it, 'Better' and a downward arrow below it, and 'Optimal' at the bottom right. A table of six rows is displayed, with the fifth row highlighted by a blue underline. The first four rows show permutations of the numbers 1, 2, and 3. The fifth row shows the next permutation, 3, 1, 2, which is the result of the 'Optimal' algorithm.



Next Permutation

arr [ ] = [ 3 1 2 ]

Brute

↓

ans → [ 3, 2, 1 ]

Better

↓

optimal

arr [ ] = [ 3 2 1 ]

ans → [ 1, 2, 3 ]

A diagram illustrating the next permutation of the array [3, 1, 2]. It shows two rows of numbers. The top row has a yellow bracket under the first three digits (1, 2, 3) and a blue bracket under the last digit (2). The bottom row has a blue bracket under the first two digits (3, 1) and a yellow bracket under the last digit (2). A blue arrow points from the top row to the bottom row. A yellow arrow points from the bottom row back up to the top row. To the right of the top row, the text "Brute" and "Better" are written vertically, with a downward arrow between them. To the right of the bottom row, the word "optimal" is written vertically below it. The text "arr [ ] = [ 3 2 1 ]" is written above the top row, and "ans → [ 1, 2, 3 ]" is written below the bottom row.



Next Permutation

arr [ ] = [ 3 1 2 ]

$\{ N! \times N^3 \}$

Brute XX

Better

Permutation

1. Generate all sorted +  
2. Linear Search  
3. Next Smaller

$\frac{120}{15} \approx 10^{12}$

The image shows a digital whiteboard with handwritten notes about generating permutations. At the top, there's a toolbar with various icons. Below it, the title "Next Permutation" is underlined. An array "arr [ ] = [ 3 1 2 ]" is shown with a blue arrow pointing to a 3x3 grid of permutations. The grid contains the following rows:  

1	2	3
1	3	2
2	1	3
2	3	1
3	1	2
3	2	1

Handwritten annotations include a large curly brace above the grid labeled " $\{ N! \times N^3 \}$ ", a yellow circle around "Permutation", and a yellow bracket on the right side of the grid. To the right, there's a box labeled "Brute XX" with a downward arrow, followed by "Better". Below these, three steps are listed: "1. Generate all sorted", "2. Linear Search", and "3. Next Smaller". At the bottom right, there's a calculation: " $\frac{120}{15} \approx 10^{12}$ ".



Next Permutation

arr [ ] = [ 3 1 2 ]

STL

→ Brute XX

Better C++

Optimal

STL

The image shows a digital whiteboard interface with a toolbar at the top. The main area contains handwritten text and diagrams. At the top, there's a title "Next Permutation" underlined in blue. Below it, "arr [ ] = [ 3 1 2 ]" is written, with "STL" written below the array. To the right, three methods are listed in boxes: "Brute" (marked with a crossed-out double X), "Better" (with a C++ arrow), and "Optimal". A large yellow circle at the bottom right contains the acronym "STL".



Next Permutation

$\text{arr}[7] = \{ 2 \ 1 \ 5 \ 4 \ 3 \ 0 \ 0 \}$

Observation

Algorithm

Brute Force

Loco

The image shows a digital whiteboard interface with various tools at the top. The main content is handwritten in white ink on a black background. At the top, the text "Next Permutation" is written with a yellow underline. Below it, the array "arr[7] = { 2 1 5 4 3 0 0 }" is shown, with the number "2" circled in red. To the right, a large red bracket groups several concepts: "Observation" and "Algorithm", which then point to "Brute Force" and "Loco". The entire whiteboard has a grid pattern.



Next Permutation

ray

-

ram

rba

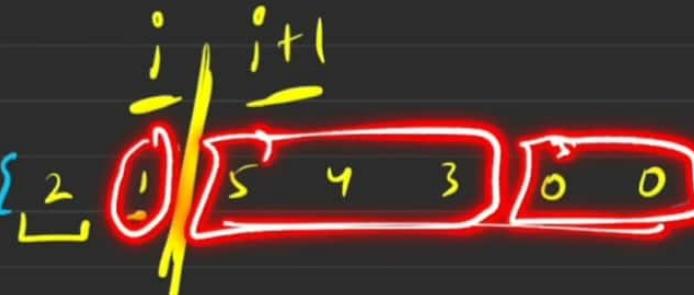
$\{ \text{ram} \} = \{ 2 \ 1 \ 5 \ 4 \ 3 \ 0 \ 0 \}$

A digital note-taking interface showing handwritten text and a list of permutations. The title "Next Permutation" is underlined. Below it, three permutations are listed: "ray", "ram", and "rba". At the bottom left, a set is defined as  $\{ \text{ram} \} = \{ 2 \ 1 \ 5 \ 4 \ 3 \ 0 \ 0 \}$ . The background of the note-taking app features horizontal ruling lines. A toolbar with various icons is visible at the top.



## Next Permutation

$\text{arr}[2] = \{2, 0, 5, 4, 3, 0, 0\}$



1. longer prefix match  
 $a[i:j] < a[i:j+1]$

2.

$< 1$   
 $> 1$



Next Permutation

1. longer prefix match  
 $a[i] < a[i+1]$

2. find  $> 1$ , but the  
smallest one  
so that you stay  
close

3. Try to place  $a[i]$   
in sorted array

array = {2, 1, 5, 4, 3, 0, 0}

2 3 0 0 1 4 5

5 4 1 0 0

Diagram showing the array [2, 1, 5, 4, 3, 0, 0]. A yellow bracket under the first two elements indicates a "longer prefix match". A blue bracket under the last four elements indicates the "smallest one" to swap. A blue arrow points from the value 1 to the position before the last four elements. A yellow bracket groups the last four elements [5, 4, 1, 0, 0], indicating the target for the swap.



Basic  $a[i] < a[i+1]$

arr[] = {2, 1, 5, 4, 3, 0, 0}

ind = -1

for(i=n-2; i>=0; i--)

{

    if(a[i] < a[i+1])

    {

        ind = i;

        break;

    }

}

2. Find  $\lceil \frac{n}{2} \rceil$ , but the smallest one so that you stay close

3. Try to place max in sorted array



1 2 3 4 5

y (ind == -1)  
rev(arr)

5 4 3 2 1

Next Permutation

arr[1] = {2 1 | 5 4 3 0 0}

→ longer prefix match  
a[i] < a[i+1]

2. find(> 1), but the



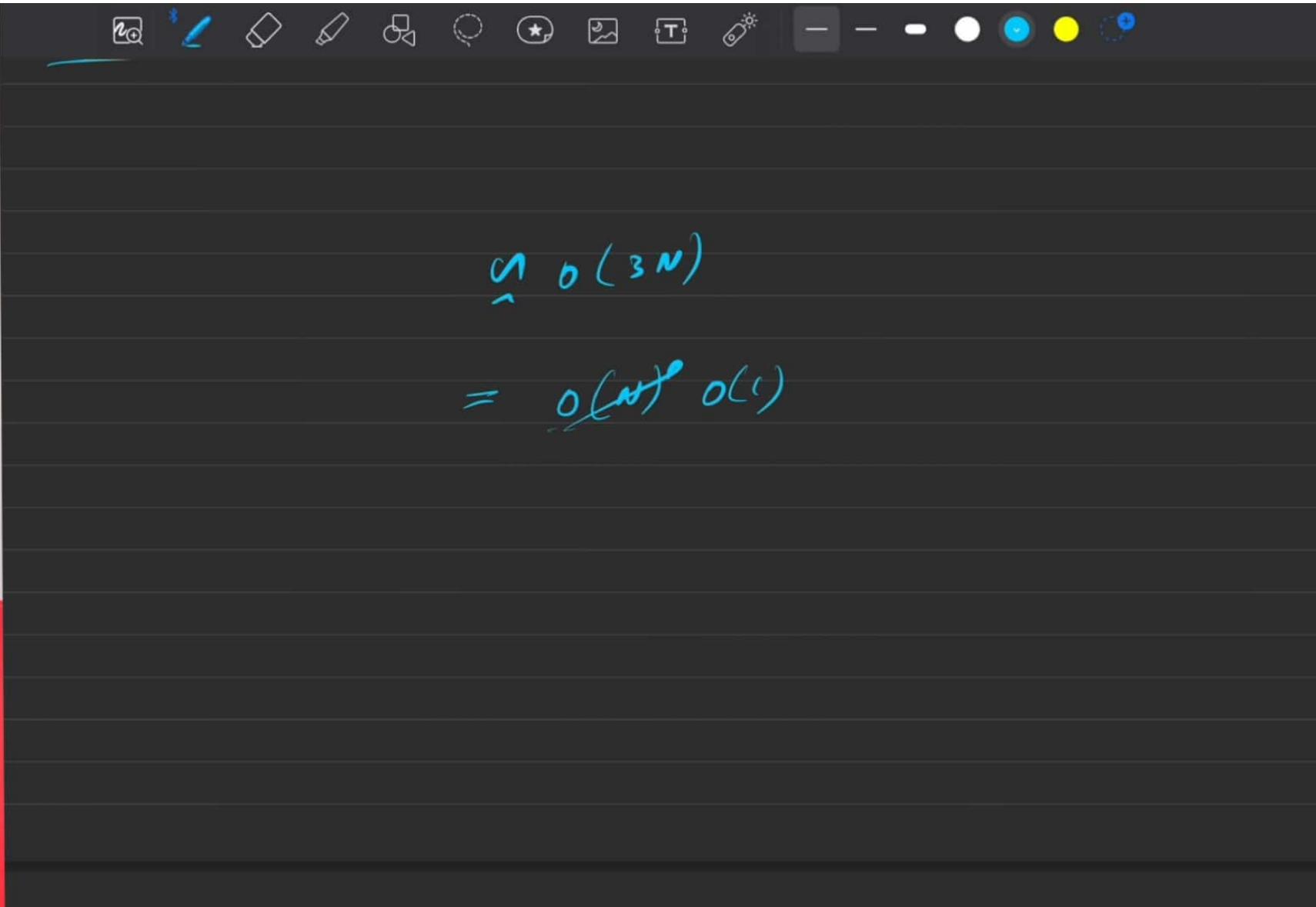
A handwritten pseudocode diagram on lined paper. At the top, there's a toolbar with various icons. Below it, a function definition is shown:

```
fun(i = n-1 ; i >= ind; i ->)
```

The body of the function contains a while loop with the condition `i >= ind`. Inside the loop, there is a call to `swap(arr[i], arr[ind])`, followed by a `break` statement. The loop is enclosed in curly braces. After the loop, there is another call to `arr(ind+1, n-1)`.

Below the pseudocode, there are two horizontal blue arrows pointing downwards, indicating the flow of execution from the end of the loop to the next step.





# Kadane's Algorithm | Maximum Subarray Sum



Maximum Subarray Sum → contiguous part of the array

arr = [-2, -3, 4, -1, -2, 1, 5, -3]

ans = 7

↓

{4, -1, 5} ← Subarr

A handwritten note on a digital interface. At the top, the text "Maximum Subarray Sum" is written with an arrow pointing to the right, followed by the explanatory text "contiguous part of the array". Below this, the array "arr" is defined as "[-2, -3, 4, -1, -2, 1, 5, -3]". The element "4" is highlighted with a red rectangle. The element "5" is circled in red. The result "ans = 7" is shown below the array. To the right of the array, the word "Subarr" is written in red with an arrow pointing to the subarray "4, -1, 5", which is also enclosed in red brackets. A red arrow points down from the array to this subarray.



Maximum Subarray Sum → contiguous part of the array

arr = [-2, -3, 4, -1, -2, 1, 5, -3]

ans = 7

The image shows a digital note-taking application interface. At the top, there is a toolbar with various icons for editing. The main area contains handwritten text and code. The text 'Maximum Subarray Sum' is written in large letters, with an arrow pointing from it to the word 'contiguous' in a smaller sentence. Below this, the code 'arr = [-2, -3, 4, -1, -2, 1, 5, -3]' is written. The subarray '[4, -1, -2, 1, 5]' is highlighted with a thick red rectangular box. To the left of the array, 'ans = 7' is written, with an arrow pointing from it to the circled value '7'.



Maximum Subarray Sum contiguous part of the array

arr = [-2, -3, 4, -1, -2, 1, 5, -3]

ans =   
fun( $i=0; i < n; i++$ )  
fun( $j=i; j < n; j++$ )

**Brunali**

**Better**

**Optimal**

}

}}



Maximum Subarray Sum

longgest part of the array

arr = [-2, -3, 9, -1, -2, 1, 5, -3]

ans = 7

Brute

fun(i=0; i<n; i++)

Better

Optimal

{i...j}

{i...r}

{i...s}

Brute force approach: A nested loop from i=0 to n-1 and j=i to n-1. The time complexity is O(n^2).

Better approach: A single pass through the array with a running sum. If the sum becomes negative, start a new subarray from the current element. The time complexity is O(n).

Optimal approach: Kadane's algorithm. It uses a single pass with a running sum and a maximum sum so far. The time complexity is O(n).



Maximum Subarray Sum

arr[T] = [-2, -3, 4, -1, -2, 1, 5, -3]

ans = 7

maxi = INT-MIN;

fun(i=0; i<n; i++)

{

fun(j=i; j<n; j++)

{ sum = 0

fun(k=i → j)

sum += arr[k]

maxi = max(sum, maxi)

TL  
 $O(N^3)$  Brute Force

SL  
 $O(1)$

Better

Optimal

every subarr

## Maximum Subarray Sum

arr[] = [-2, -3, 4, -1, -2, 1, 5, -3]

ans = 7

maxi = INT-MIN;

fun(i=0; i<n; i++)

{ sum = 0

fun(j=i; j<n; j++)

{ sum += arr[j];

} maxi = max(sum, maxi)

TL  
O(n<sup>3</sup>) Brute Force

SL  
O(n<sup>2</sup>) Better

Optimal  
Dynamic Programming



Maximum Subarray Sum

longest part of one array

arr = [-2, -3, 4, -1, -2, 1, 5, -3]

ans = 7

max = INT-MIN -2

Sum = 0  
-2

$\leftarrow -2, -3 \right] \quad \{ -3 \}$

$\overbrace{-5}^{\text{Optimal}} \quad \underbrace{-3}_{\text{Optimal}}$

TL      SL

- $O(n^3)$  Brute       $O(1)$
- $O(n^2)$  Better       $O(1)$

Optimal

[Kadane's Algorithm]

This diagram illustrates the Kadane's algorithm for finding the maximum subarray sum. It shows an array [ -2, -3, 4, -1, -2, 1, 5, -3 ] with a red bracket underlining the subarray [-2, -3] as the current maximum. A red circle highlights the value -3 in the array. Red arrows point from the subarray [-2, -3] to the value -5 and from the value -3 to the circle around -3, both labeled 'Optimal'. To the right, a comparison is made between two approaches: a brute-force method with time complexity  $O(n^3)$  and space complexity  $O(1)$ , and a better method with time complexity  $O(n^2)$  and space complexity  $O(1)$ . The better method is identified as Kadane's algorithm.



## Maximum Subarray Sum

longer part of one array

$$\text{arr}[T] = [-2, -3, 4, -1, -2, 1, 5, -3]$$

**0**

ans = 7

$$m_{ani} = \text{INT-INT} - 2$$

$$\text{Sum} = \cancel{x}$$

Sum < 0

$T_C$        $SL$   
 •  $(n^3)$  Brute       $O(1)$   
 ↓  
 $O(n^2)$  Better       $O(1)$

Optimal

[Kadane's Algorithm]



Maximum Subarray Sum

longgest part of one array

$\text{arr}[] = [-2, -3, 4, -1, -2, 1, 5, -3]$

$\text{ans} = 7$

$\text{maxi} = 4$

$\text{Sum} = \cancel{0} \quad \cancel{-3} \quad 4$

$\cancel{-2} \quad 0$

$0$

$\text{sum} < 0$

TL      SL  
 $\bullet (n^3)$  Brute       $O(1)$

$\downarrow$

$O(n^2)$  Better       $O(1)$

Optimal

[Kadane's Algorithm]



## Maximum Subarray Sum

continuous part of one array

arr = [-2, -3, 9, -1, -2, 1, 5, -3]  
ans = 7



maxi = 9

Sum =  $\begin{matrix} 0 & -2 & -3 & 9 \\ -2 & 0 & 3 & \end{matrix}$

Sum < 0

TL  
•  $O(N^3)$  Brute  $O(1)$

$\downarrow$   
 $O(N^2)$  Better  $O(1)$

Optimal

[Kadane's Algorithm]



Maximum Subarray Sum

longgest part of one array

arr = [-2, -3, 9, -1, -2, 1, 5, -3]

ans = 7

mani = 47

Sum = 0 -3 9  
-2 0 3  
0 X  
2  
7  
9

sum < 0

TL      SL  
 $\bullet (N^3)$  Brute     $O(1)$   
 $\downarrow$   
 $\underline{O(N^2)}$  Better     $\underline{O(1)}$

Optimal

[Kadane's Algorithm]

95 of 136



codestudio  
POWERED BY CODING NINJAS

Guided Paths Contests Interview Prep Practice Resources

Problem of the day

Topics Problem Submissions Solution New Discuss

## Maximum Subarray Sum

Contributed by Achint Narang

Medium 80/80 Avg time to solve 35 mins Success Rate 81 %

Share 702 upvotes

### Problem Statement

You are given an array (ARR) of length N, consisting of integers. You have to find the sum of the subarray (including empty subarray) having maximum sum among all subarrays.

A subarray is a contiguous segment of an array. In other words, a subarray can be formed by removing 0 or more integers from the beginning, and 0 or more integers from the end of an array.

Note :

The sum of an empty subarray is 0.

### Detailed explanation

(Input/output format, Notes, Images)

Constraints :

$1 \leq N \leq 10^6$   
 $-10^6 \leq A[i] \leq 10^6$

```
#include <bits/stdc++.h>
long long maxSubarraySum(int arr[], int n)
{
    long long sum = 0, maxi = LONG_MIN;
    for(int i = 0;i<n;i++) {
        sum += arr[i];
        if(sum > maxi) {
            maxi = sum;
        }
        if(sum < 0) {
            sum = 0;
        }
    }
    return maxi;
}
```

< Prev Next >

View Last saved on 9:39:18 PM



A digital whiteboard interface with a toolbar at the top featuring various drawing tools like a magnifying glass, pencil, eraser, etc. The main area contains handwritten mathematical content:

- A set of integers:  $\{-4, -2, -3, \boxed{-1}\}$  where  $-1$  is highlighted with a blue box.
- A yellow box contains the value  $-1$ .
- A yellow box contains the equation  $m = -1$ .
- A red oval encloses the number  $1$ , with a red arrow pointing from it to a red circle labeled  $0$ .



TL      SL

Time =  $O(n)$

Sum =  $\sigma -31 4$

$\sigma$     $-31$     $4$   
 $-2$     $0$     $3$   
 $0$     $+$     $-$   
 $x$     $y$

Sum < 0

$O(n^2)$  Brute    $O(1)$   
 $\downarrow$

$O(n^2)$  Better    $O(1)$

Optimal

$O(n)$  [Kadane's Algorithm]    $O(1)$

{ -4 -2 -3  $\sqrt{-1}$  }

A handwritten note on a digital whiteboard. At the top, there are various icons for file operations like zoom, pen, eraser, crop, etc. Below that, there are two columns labeled 'TL' and 'SL'. In the 'TL' column, it says 'Time =  $O(n)$ ' with a blue underline. In the 'SL' column, it says ' $O(1)$ '. Then, there is a large yellow equation 'Sum =  $\sigma -31 4$ ' with a blue underline. Below the equation, there are several rows of numbers: '-2', '0', '3', '0', '+', and '-' followed by 'x' and 'y'. To the right of the equation, there is a yellow arrow pointing down from 'Sum < 0' to ' $O(n^2)$  Brute' and ' $O(1)$ ', with a blue underline under both. Further down, another yellow arrow points down from ' $O(n^2)$  Better' and ' $O(1)$ ' to 'Optimal'. At the bottom, there is a blue underlined section containing ' $O(n)$  [Kadane's Algorithm]' and ' $O(1)$ '. At the very bottom, there is a blue bracketed expression: '{ -4 -2 -3  $\sqrt{-1}$  }'.



→ Contiguous part of the array

Printing Maximum Subarray Sum

arr[] = [-2, -3, 4, -1, -2, 1, 5, -3]

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

{

    sum = sum + arr[i];

    if (sum > max)

        max = sum;

}

if (sum < 0)

    sum = 0;

max = INT-MIN

sum = 0



Printing Maximum Subarray Sum → contiguous part of the array

arr[] = [-2, -3, 9, -1, -2, 1, 5, -3]  
           $\underbrace{\quad}_{0} \underbrace{\quad}_{0} \underbrace{\quad}_{0}$

for (i=0; i<n; i++)  
    {  
        sum = sum + arr[i];  
  
        if (sum > max)  
            max = sum;  
  
        if (sum < 0)  
            sum = 0;  
    }

max = INT-MIN  
sum = 0



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

{ if ( $sum == 0$ ) start =  $i$ ;

    sum = sum + arr[i];

    if ( $sum > main$ )

        main = sum;

        [ansStart = start, ansEnd = i];

    if ( $sum < 0$ )

        sum = 0;

}

}

main = INT-MIN

sum = 0

ansStart = -1

ansEnd = -1

$\boxed{[TC \rightarrow O(n) \\ SC \rightarrow O(1)]}$

A handwritten code snippet on a digital notepad. The code uses curly braces for both function blocks and condition blocks. It includes several annotations in yellow: 'main = INT-MIN' and 'sum = 0' are placed to the right of their respective declarations; 'ansStart = -1' and 'ansEnd = -1' are placed to the right of the assignment inside the first if-block; and a large bracket at the bottom right groups the time complexity 'TC → O(n)' and space complexity 'SC → O(1)'.

Sort an array of 0's 1's & 2's



Sort an array of 0's, 1's and 2's.

arr[] = 

0	1	2	0	1	2	1	2	0	0	0	1
---	---	---	---	---	---	---	---	---	---	---	---

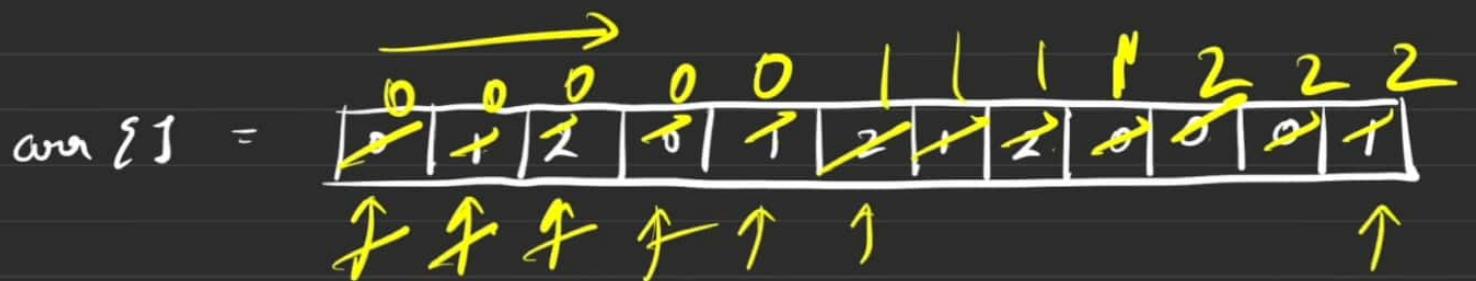
0 1 1 2

$T_C$   
 $(N \log N)$

$S_C$   
 $(N)$  Sort

$\rightarrow$  Brute  
Better ↗  
Optimal

Sort an array of 0's, 1's and 2's.



TC  $(N \log N)$  SC  $(N)$  sort

→ Brute  
Better  
Optimal

0 1 1 2

$$\begin{aligned} \text{cnt } 0 &= 1 + 2 + 5 \\ \text{cnt } 1 &= 1 + 4 \\ \text{cnt } 2 &= 1 + 3 \end{aligned}$$



$$\text{area } \mathcal{E} = \boxed{0 | 1 | 2 | 0 | 1 | 2 | 1 | 2 | 0 | 0 | 0 | 1}$$

$\text{cut } 0 = 0$     $\text{cut } 1 = 1$     $\text{cut } 2 = 2$       ↑

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

```
if (a[5] == 0) cout << ;  
else if (a[5] == 1) cout << ;  
else cout << ;
```

3

for(i=0 ; i< auto ; i+t) a[i] = 0 →

```
fun( i=cnt0; i< cnt0+cnt1; i++) a[i]=1 →
```

for ( i = cout < 0 + cout < 1 ; i < n ; i++) a[i] = 2 →

## Optimal

## Brule

## Butter

## Optimal

$O(n)$

1

$$\tau_c \rightarrow o(2N) \quad \zeta \zeta \rightarrow o(1)$$



arr [ ] = [ 0 | 1 | 2 | 0 | 1 | 2 | 1 | 2 | 0 | 0 | 0 | 1 ]

$O(2^n)$

→ Brute  
Better  
Optimal

will come in and this is going to blow  
your mind

A digital whiteboard interface showing handwritten code and annotations. The code 'arr [ ] = [ 0 | 1 | 2 | 0 | 1 | 2 | 1 | 2 | 0 | 0 | 0 | 1 ]' is written in black ink. To the right, there are handwritten notes in yellow and orange: 'Brute' with an arrow pointing to it, 'Better' with an arrow pointing to it, and 'Optimal' inside a yellow-outlined box. Below these, the time complexity 'O(2^n)' is written. A large black rectangular box at the bottom contains the text 'will come in and this is going to blow your mind'.



Dutch National Flag Algorithm  $\rightarrow$  Optimal

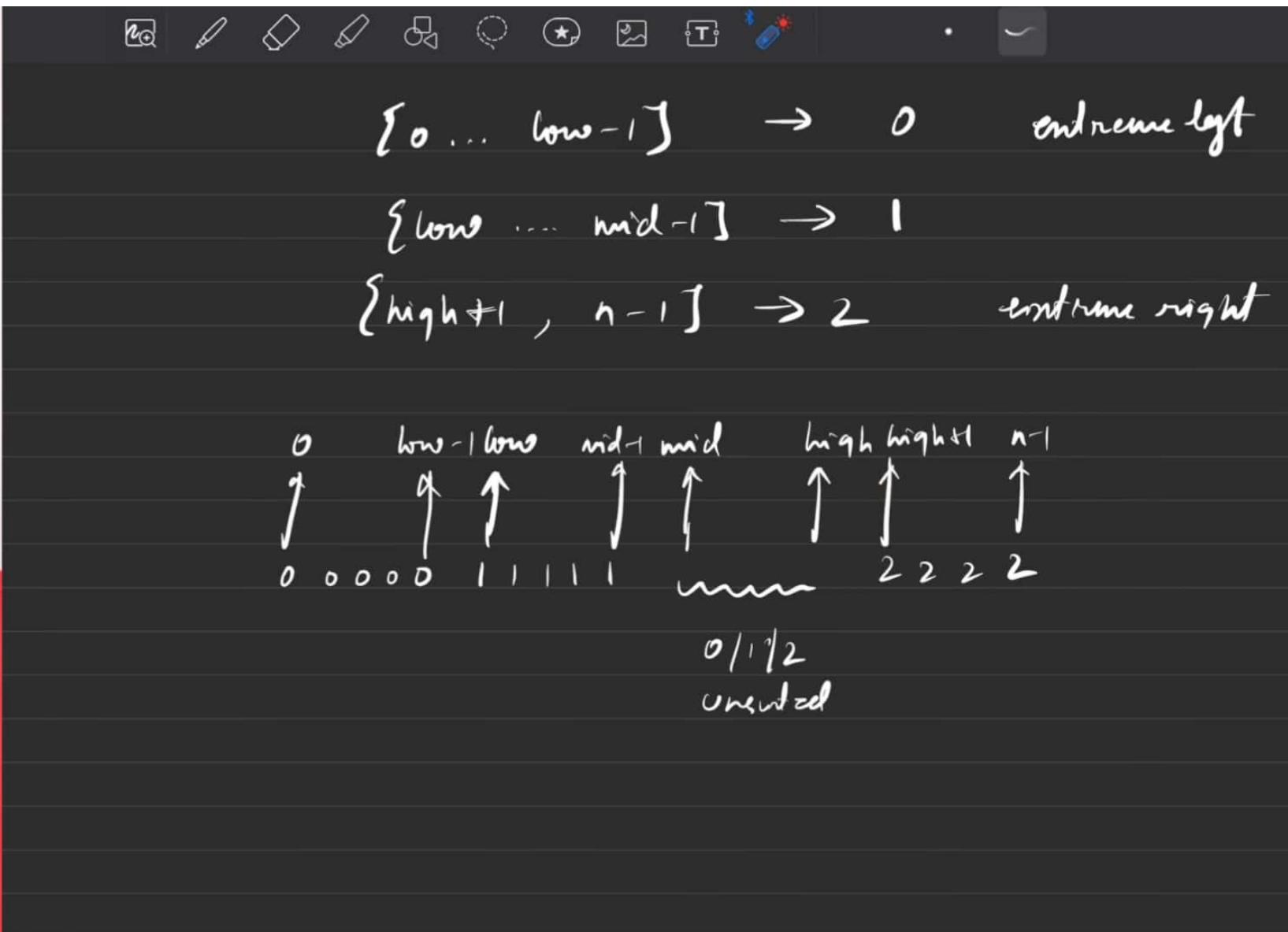
arr [ ] = 0 | 1 | 1 | 0 | 1 | 2 | 1 | 2 | 0 | 0 | 0

low

mid

high

A screenshot of a digital whiteboard or note-taking app. At the top, there is a toolbar with various icons for drawing and editing. Below the toolbar, the text "Dutch National Flag Algorithm" is written in black ink, with a red arrow pointing from it to the word "Optimal" which is underlined. Underneath this, there is a sequence of numbers in brackets: "arr [ ] = [0 | 1 | 1 | 0 | 1 | 2 | 1 | 2 | 0 | 0 | 0]". Below the sequence, the words "low", "mid", and "high" are written vertically, likely indicating indices or pointers used in the algorithm. The background of the whiteboard is dark grey.

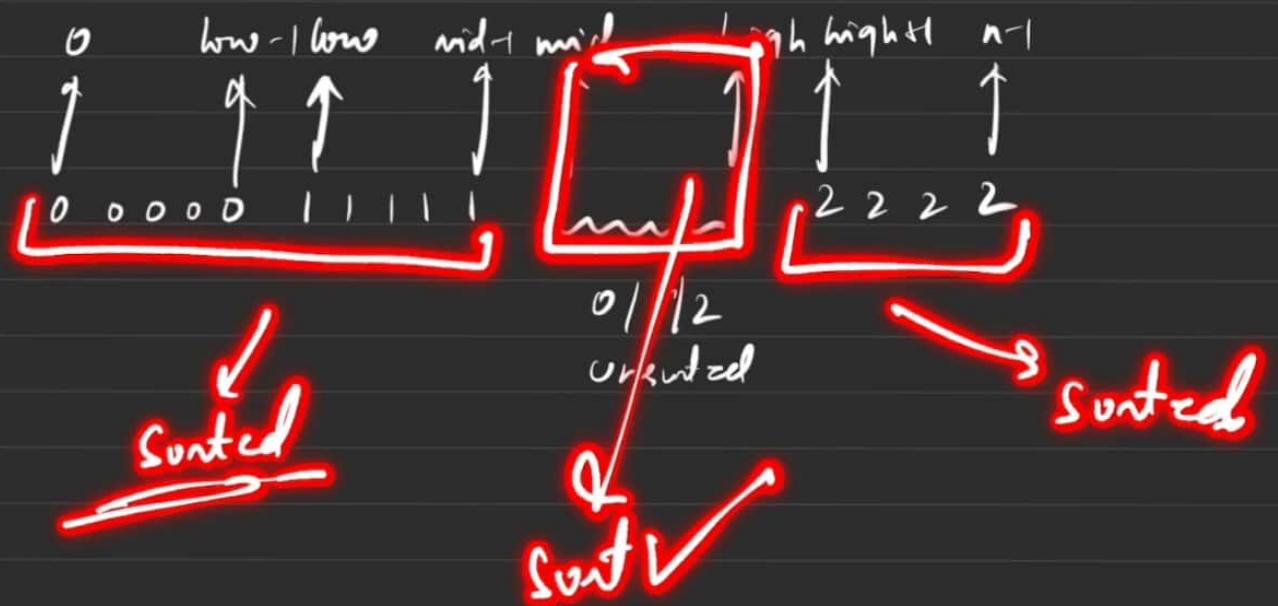




$\{0 \dots \text{low}-1\} \rightarrow 0$  extreme left

$\{\text{low} \dots \text{mid}-1\} \rightarrow 1$

$\{\text{high}+1, n-1\} \rightarrow 2$  extreme right





Dutch National Flag Algorithm  $\rightarrow$  Optimal

arr[ ] = [0 | 1 | 1 | 0 | 1 | 2 | 1 | 2 | 0 | 0 | 0]

low                          mid                          high

The image shows a digital whiteboard or notes application interface with various icons at the top. The main content is handwritten text and a diagram. The text "Dutch National Flag Algorithm" is underlined, followed by an orange arrow pointing to the word "optimal" which is also underlined. Below this, the text "arr[ ] = [0 | 1 | 1 | 0 | 1 | 2 | 1 | 2 | 0 | 0 | 0]" is written, representing an array of integers. Three yellow arrows point to specific indices: one from the label "low" to the first element (0), one from the label "mid" to the third element (1), and one from the label "high" to the last element (0). The array elements are separated by vertical lines and some horizontal lines, creating a segmented look.



low

mid

mid

high

T  
high

$$a[\text{mid}] == 0$$

$$a[\text{mid}] == 1$$

$$a[\text{mid}] == 2$$

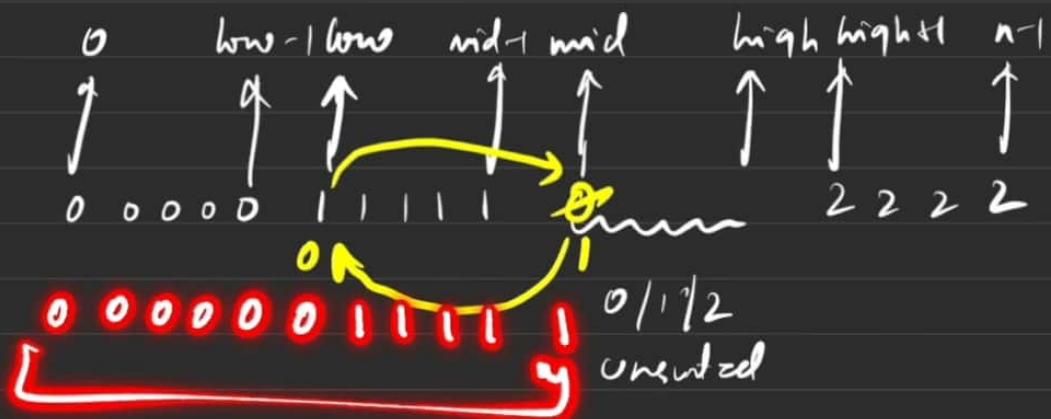
$[0 \dots \text{low}-1]$   $\rightarrow$  0 extreme left



[0 ... low-1] → 0   extreme left

[low ... mid-1] → 1

[high+1, n-1] → 2   extreme right

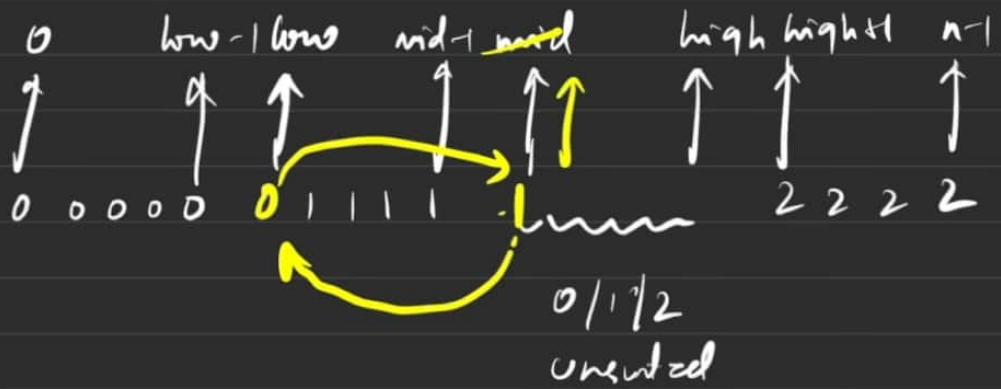




{ $o \dots low-1$ } → 0 extreme left

{low ... mid-1] → |

$\{high+1, n-1\} \rightarrow 2$  extreme right

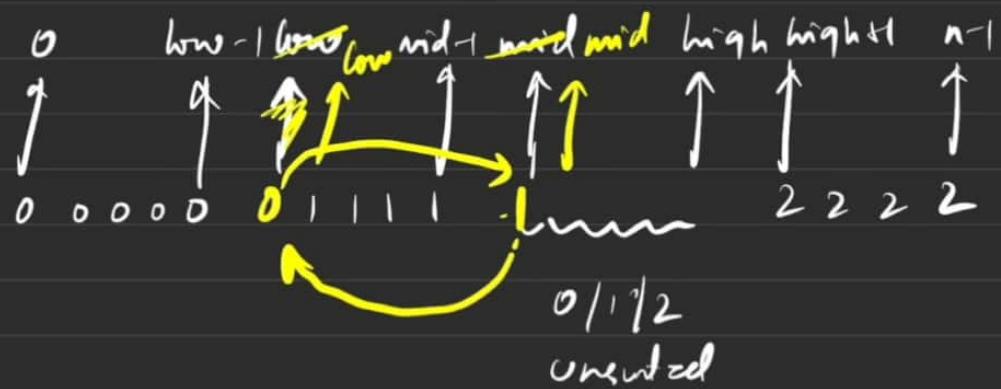




$\{0 \dots low-1\} \rightarrow 0$  extreme left

$\{low \dots mid-1\} \rightarrow 1$

$\{high+1, n-1\} \rightarrow 2$  extreme right





arr [ ] =   
low ↑ mid ↑ high ↑

low

mid

high

✓  $a[\text{mid}] == 0$  swap( $a[\text{low}]$ ,  $a[\text{mid}]$ )  
 $\text{low}++$ ,  $\text{mid}++$

$a[\text{mid}] == 1$

$a[\text{mid}] == 2$



l o ... low - 1 ) - . . . unsorted

{ low ... mid - 1 }  $\rightarrow$  1

{ high + 1 , n - 1 }  $\rightarrow$  2      extreme right

0      low - 1      low      mid - 1      mid      high      high + 1      n - 1  
↑      |      ↑      |      ↑      |      ↑  
0 0 0 0 0      1 1 1 1 1      2 2 2 2 2

0 / 1 / 2  
unsorted



0 ... low-1 low mid-1 mid high high n-1

{low ... mid-1} → 1

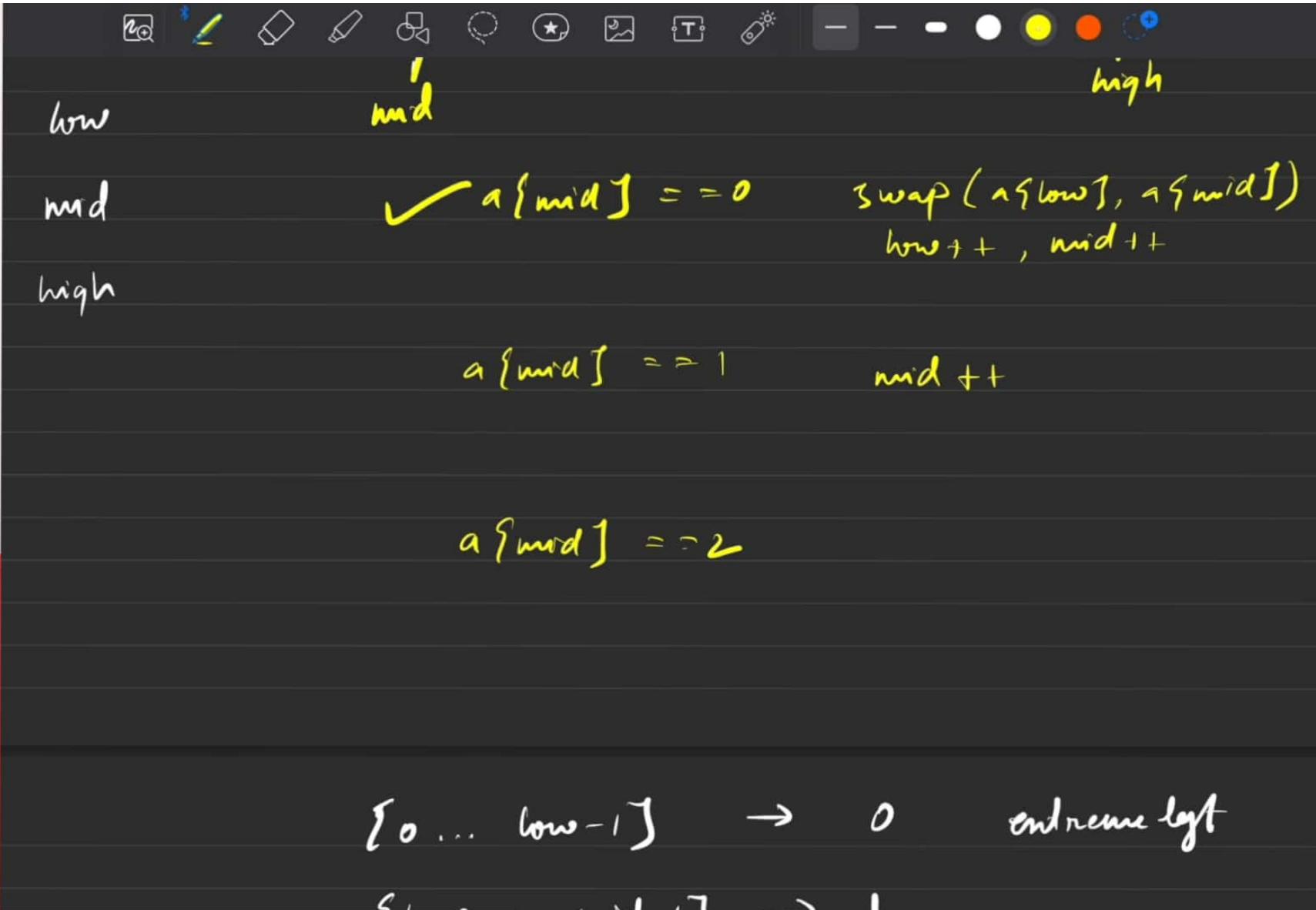
{high+1, n-1} → 2      extreme right

0 0 0 0 0    1 1 1 1    2 2 2 2

↑ ↑ ↑ ↑ ↑  
0 0 0 0    1 1 1 1    2 2 2 2

**T**  
**gad**  
**begin**  
1/2  
unsorted

A handwritten diagram illustrating a search or partitioning process on an array of integers. The array is shown with four groups of numbers: 0s, 1s, 2s, and 3s. Above the array, indices are labeled: 0, ..., low-1, low, mid-1, mid, high, high+1, ..., n-1. Below the array, arrows point from each group of numbers to its corresponding index label. A large red 'T' is written vertically through the middle of the array, with 'gad' written across it. Below the array, the word 'begin' is written under the first 0s, and 'unsorted' is written below the entire array. The number 1/2 is written below the 'begin' label.





low ... mid ...

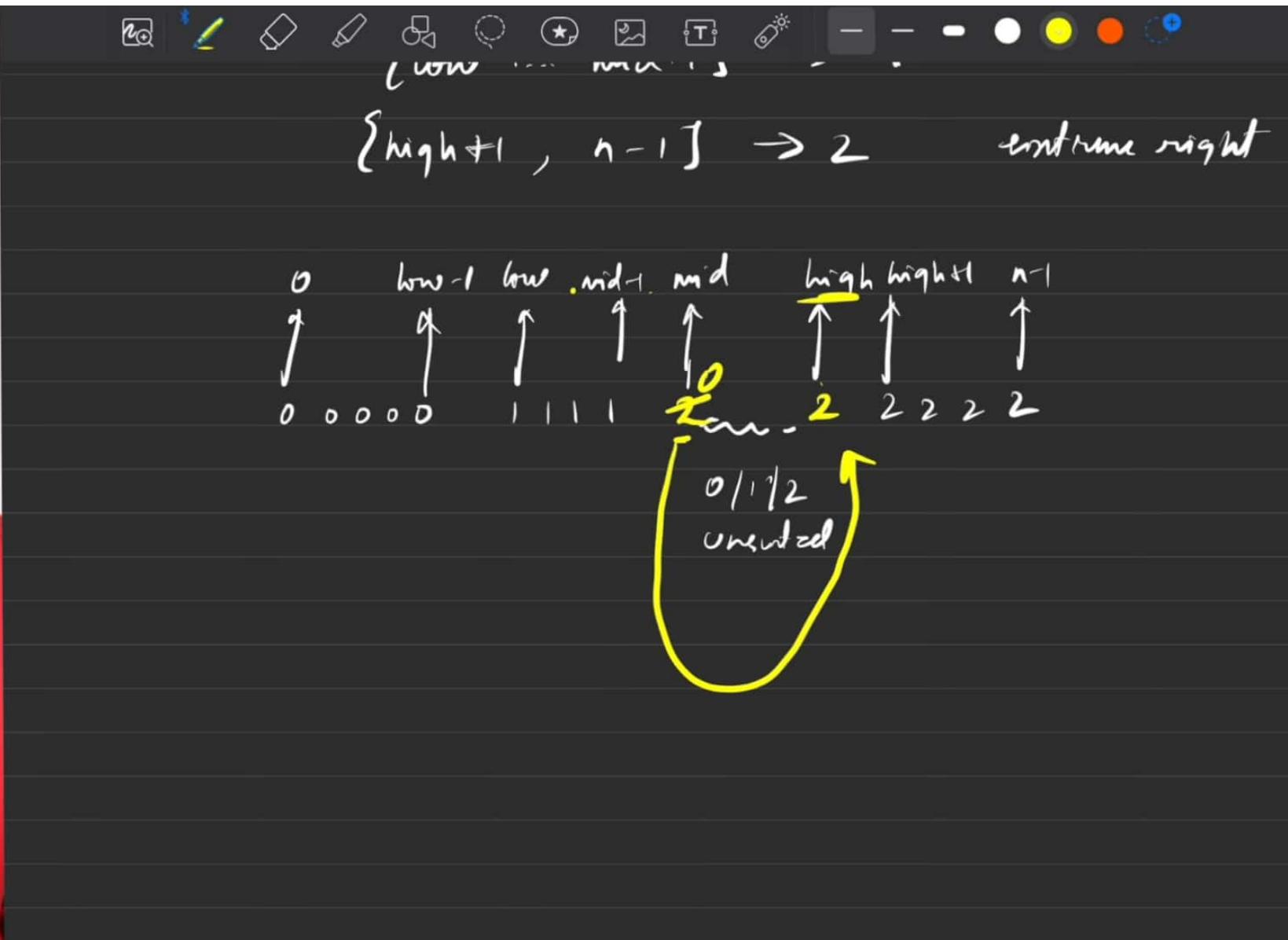
$\{high+1, n-1\} \rightarrow 2$  extreme right

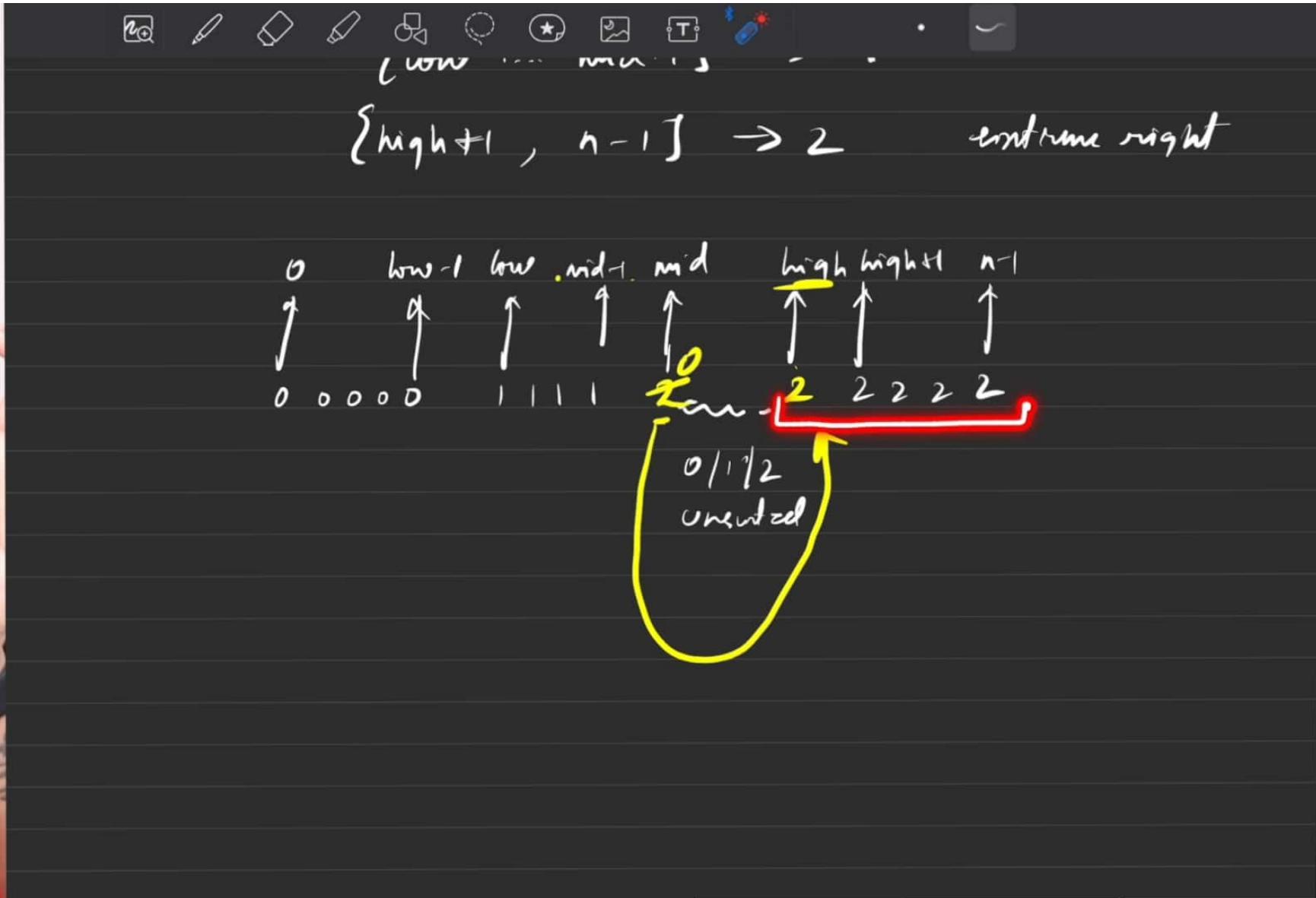
0 low-1 low mid-1 mid high high+1 n-1  
↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑  
0 0 0 0 1 1 1 2 2 2 2 2

2 and 2

0/1/2  
unsorted

The diagram illustrates a binary search process on an array of length 13. The array elements are labeled from 0 to n-1. The search range is indicated by brackets from index 0 to high+1 and index n-1 to high. The search is narrowed down to the element at index 2, which is highlighted in yellow. A red bracket groups the elements at indices 2 and 2, and a red arrow points to the value 2. The text "unsorted" is written below the array.







low                          mid                          high

mid                          ✓  $a[\text{mid}] == 0$       swap( $a[\text{low}]$ ,  $a[\text{mid}]$ )  
                                low++, mid++

high                          ✓  $a[\text{mid}] == 1$       mid++

✓  $a[\text{mid}] == 2$       swap( $a[\text{mid}]$ ,  $a[\text{high}]$ )  
                                high--;

$\{0 \dots \text{low}-1\} \rightarrow 0$       end recursive left



.



## Dutch National Flag Algorithm $\rightarrow$ Optimal

low



$a[0 \dots 9] = [0 | 1 | 1 | 0 | 1 | 2 | 1 | 2 | 0 | 0 | 0]$

low

mid

mid

high

high

$\checkmark a[mid] == 0$

swap( $a[low], a[mid]$ )

$low++$ ,  $mid++$

$\checkmark a[mid] == 1$

mid++



Dutch National Flag Algorithm  $\rightarrow$  Optimal

arr [ ] = 

low                          high

mid

high

$\checkmark a[\text{mid}] == 0$       swap( $a[\text{low}], a[\text{mid}]$ )  
low++, mid++

$\checkmark a[\text{mid}] == 1$       mid++



Dutch National Flag Algorithm  $\rightarrow$  Optimal

↓  
*low*

arr [ ] = 0 | 1 | 1 | 0 | 1 | 2 | 1 | 2 | 0 | 0 | 0

↑  
*mid*

↑  
*high*

$\checkmark a[\text{mid}] == 0$       swap( $a[\text{low}], a[\text{mid}]$ )  
 $\text{low}++$ ,  $\text{mid}++$

$\checkmark a[\text{mid}] == 1$        $\text{mid}++$

A handwritten note on a digital notepad. At the top left, it says "Dutch National Flag Algorithm" with a blue arrow pointing to the right and the word "Optimal" underlined. Below this, there is a diagram of an array with eleven cells. The first four cells contain "0", the next three contain "1", the next two contain "2", and the last two are empty. A blue arrow labeled "low" points to the first cell. A blue arrow labeled "mid" points to the fifth cell. A yellow arrow labeled "high" points to the ninth cell. Below the array, there are two yellow checkmarks followed by text: "swap(a[low], a[mid])" and "low++, mid++". At the bottom, another yellow checkmark is followed by "mid++" underlined.



Dutch National Flag Algorithm  $\rightarrow$  Optimal  
↓  
low



low

mid

high

$$\checkmark a[\text{mid}] == 0$$

$\text{swap}(a[\text{low}], a[\text{mid}])$   
 $\text{low}++$ ,  $\text{mid}++$

$$\checkmark a[\text{mid}] == 1$$

mid ++



Dutch National Flag Algorithm  $\rightarrow$  Optimal

↓  
*low*

arr [ ] = 0 | 1 | 1 | 0 | 1 | 2 | 1 | 2 | 0 | 0 | 0

↑  
*mid*

↑  
*high*

low  
mid  
high

$\checkmark a[\text{mid}] == 0$  swap( $a[\text{low}], a[\text{mid}]$ )  
 $\text{low}++, \text{mid}++$

$\checkmark a[\text{mid}] == 1$  mid++





Dutch National Flag Algorithm → Optimal

→ Optimal

low

arr[8] = [0, 0, 1, 1, 1, 2, 1, 2, 0, 0, 0]

www

md

high

$$\checkmark a\{mid\} == 0$$

swap(a[low], a[mid])  
low++, mid++

$$\checkmark a \{ \text{and} \} = = 1$$

mid++



Dutch National Flag Algorithm  $\rightarrow$  Optimal

low

arr [ ] = 

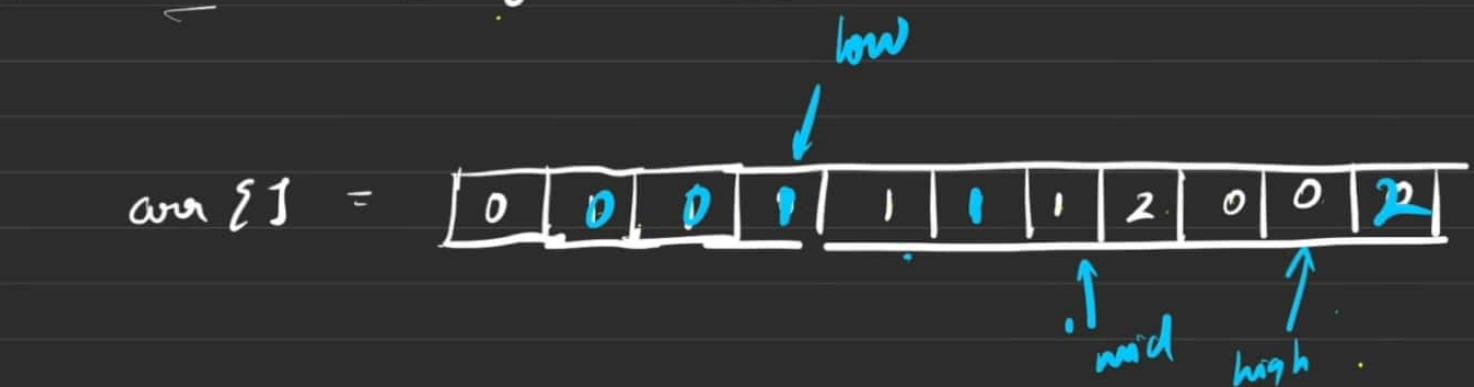
low      mid      high

$\checkmark a[\text{mid}] == 0$       swap( $a[\text{low}], a[\text{mid}]$ )  
 $\text{low}++$ ,  $\text{mid}++$

$\checkmark a[\text{mid}] == 1$       mid++



Dutch National Flag Algorithm  $\rightarrow$  optimal



low

mid

high

✓  $a[\text{mid}] == 0$

swap( $a[\text{low}], a[\text{mid}]$ )  
 $\text{low}++$ ,  $\text{mid}++$

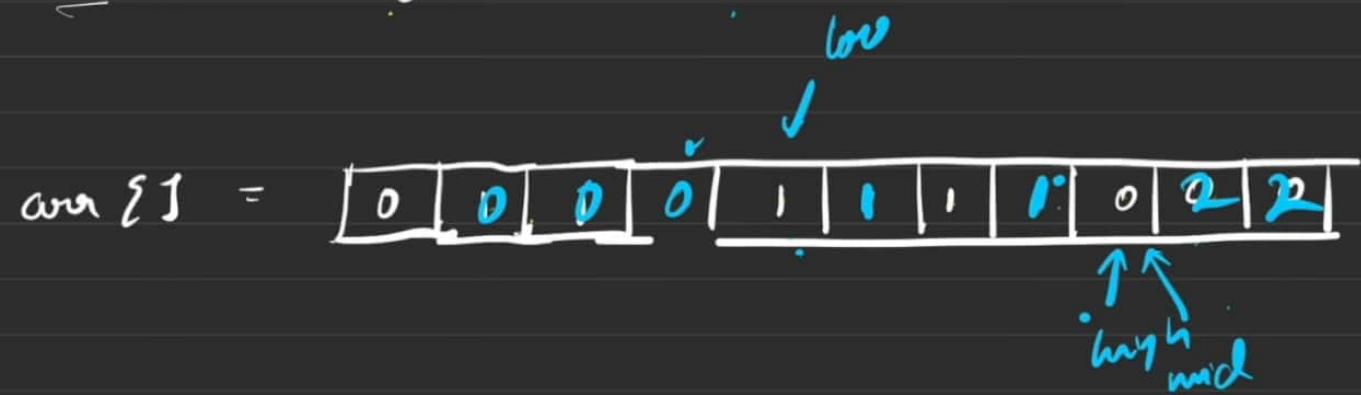
✓  $a[\text{mid}] == 1$

mid ++





## Dutch National Flag Algorithm → Optimal



low

mid

high

✓  $a[\text{mid}] == 0$

swap( $a[\text{low}]$ ,  $a[\text{mid}]$ )  
 $\text{low}++$ ,  $\text{mid}++$

✓  $a[\text{mid}] == 1$

mid ++



Dutch National Flag Algorithm → Optimal

low

arr [ ] =   
high mid

low ++ , mid ++

✓  $a[\text{mid}] == 0$  swap( $a[\text{low}], a[\text{mid}]$ )  
 $\text{low}++$ ,  $\text{mid}++$

✓  $a[\text{mid}] == 1$  mid ++



arr [ ] = 

low

mid

high

$\checkmark a[\text{mid}] == 0$  swap( $a[\text{low}]$ ,  $a[\text{mid}]$ )  
 $\text{low}++$ ,  $\text{mid}++$

$\checkmark a[\text{mid}] == 1$  mid++

$\checkmark a[\text{mid}] == -2$  swap( $a[\text{mid}]$ ,  $a[\text{high}]$ )  
 $\text{high}--;$



codestudio  
POWERED BY CODING NINJAS

Guided Paths Contests Interview Prep Practice Resources

Problem of the day

Topics Problem Submissions Solution New

### Sort An Array of 0s, 1s and 2s

Contributed by Anup Kumar Singh

Easy 0/40 Avg time to solve 10 mins Success Rate 90 %

Share 3 upvotes

#### Problem Statement

You have been given an array/list ARR consisting of 'N' elements. Each element in the array is either 0, 1 or 2. Now, your task is to sort this array/list in increasing order. For example, if ARR = [0, 2, 1, 1], then after sorting ARR must look like [0, 1, 1, 2].

#### Detailed explanation

(Input/output format, Notes, Images)

#### Constraints:

1 <= N <= 5000  
0 <= ARR[i] <= 2

Time limit: 1 sec

```
#include <bits/stdc++.h>
void sortArray(vector<int>& arr, int n) {
    int low = 0, mid = 0, high = n-1;
    while(mid <= high) {
        if(arr[mid] == 0) {
            swap(arr[low], arr[mid]);
            low++;
            mid++;
        }
        else if(arr[mid] == 1) {
            mid++;
        }
        else {
            swap(arr[mid], arr[high]);
            high--;
        }
    }
}
```

Last saved on 6:34:04

Run Submit

Console

Sample Test Case

5/5 test cases passed

- Test Case 1: Correct Answer
- Test Case 2: Correct Answer
- Test Case 3: Correct Answer
- Test Case 4: Correct Answer
- Test Case 5: Correct Answer



Diagram illustrating the Dutch National Flag Algorithm:

Time Complexity ( $T_C$ ):  $O(N)$

Space Complexity ( $S_C$ ):  $O(1)$

The algorithm uses three pointers to partition an array of length  $N$  into three regions:

- Low**: The region containing elements less than the pivot. It is bounded by the **low** pointer (marked with a checkmark) and the **mid** pointer.
- Mid**: The region containing elements equal to the pivot. It is bounded by the **mid** pointer (marked with a checkmark) and the **high** pointer.
- High**: The region containing elements greater than the pivot. It is bounded by the **mid** pointer (marked with a checkmark) and the **high** pointer.

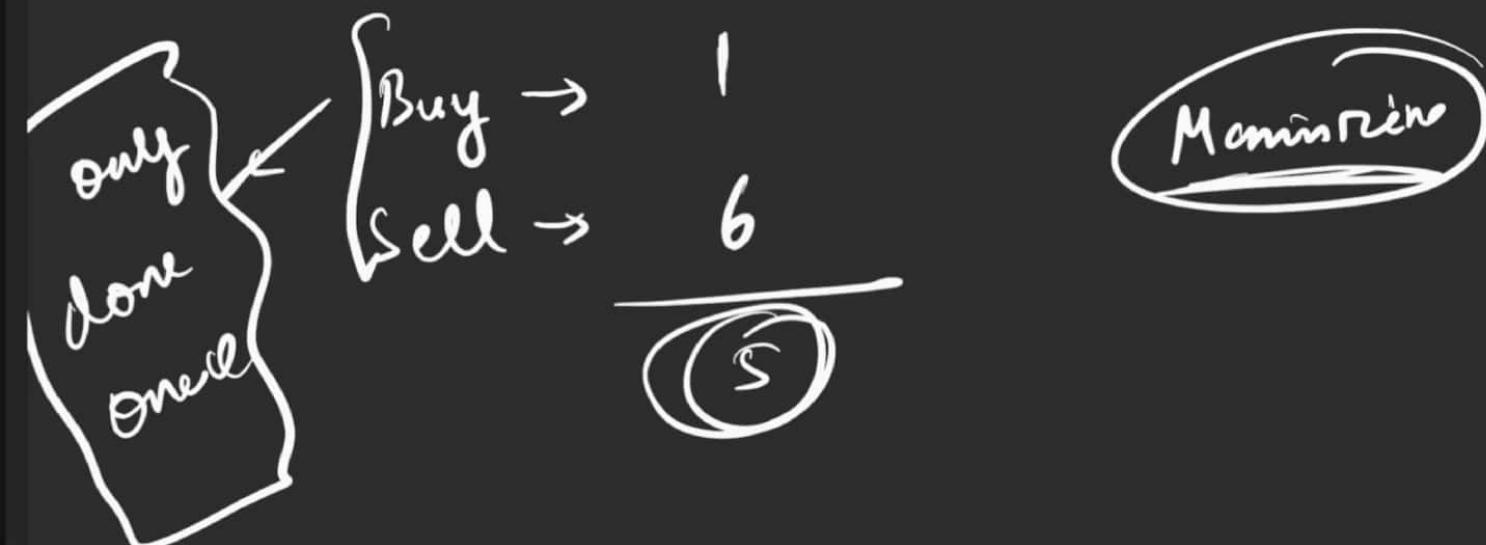
The algorithm is labeled as Optimal.

90 of 94

# **Best Time to Buy and Sell Stock**

Best Time to Buy & Sell stock

arr[] → { 7, 15, 5, 3, 6, 4 ]      n=6



7 1 5 3 6 4

If you are selling on  $i^{th}$  day  
you buy on the minin price from  $\boxed{1st \rightarrow (i-1)}$

$mini = a[0]$ , profit = 0

$\{$  fun( $i=1$ ;  $i < n$ ;  $i+1$ )



$mini = a[0]$ ,  $profit = 0$

$\{$   
     $for(i=1; i < n; i++)$

$cost = a[i] - mini;$

$profit = \max(profit, cost);$

$mini = \min(mini, a[i]);$

$\}$





Problem Submissions

## Correct Answer

 Report

Submitted on Mar 11, 2022, 4:46:52 AM

Penalty

Score ?

0% 100%

Runtime

196ms

Language

C++ (g++ 5.4)

```
1 int maximumProfit(vector<int> &prices){  
2     int mini = prices[0];  
3     int maxProfit = 0;  
4     int n = prices.size();  
5     for(int i = 0;i<n;i++) {  
6         int cost = prices[i]- mini;  
7         maxProfit = max(maxProfit, cost);  
8         mini = min(mini, prices[i]);  
9     }  
10    return maxProfit;  
11    // Write your code here.  
12 }
```

 Previous  Next  Show Hint Last saved on 4:46:53 AM