

Optimal/Optimised Approach

Constraints

Constraints define the range of values for input variables in a problem. They help you understand:

- What size of input (n) is allowed
- What time and space efficiency is expected
- Whether brute force will work or not

Example:

$$1 \leq n \leq 10^5$$

$$1 \leq \text{arr}[i] \leq 10^9$$

Means: Input array can be of size up to 100,000 and elements up to 1 billion.

Why Are Constraints Important?

Understanding constraints allows you to:

- Predict the correct **algorithm complexity**
- Decide whether **brute force**, **DP**, or **optimized techniques** are needed
- Avoid **TLE (Time Limit Exceeded)** or **MLE (Memory Limit Exceeded)**
- Estimate how many **operations per second** are allowed

Modern Computer $\rightarrow 10^8 \text{ ops/sec}$

problem \rightarrow 1 sec \rightarrow 1.46 TLE / MLE

✓ 1. Time Limit: $\leq 10^8$ operations ($\approx 1s$)

Acceptable Complexity: $O(n)$, $O(n \log n)$

Case 1: $O(n)$

- Let's assume $n = 10^8$
- Then number of operations = $n = 10^8$ ✓

Case 2: $O(n \log n)$

Let's assume $n = 10^6$

$$\log_2(10^6) = \frac{\log_{10}(10^6)}{\log_{10}(2)} = \frac{6}{0.3010} \approx 19.93 \approx 20$$

$$n \cdot \log_2 n = 10^6 \cdot 20 = 2 \cdot 10^7 \text{ operations}$$

✓ 2×10^7 is well below 10^8 → acceptable.

Time Limit vs Acceptable Complexity (With Proofs)

Time Limit	Acceptable Complexity	Proof & Reasoning
$\leq 10^8$ ops (1s)	$O(n)$, $O(n \log n)$	CPU handles $\sim 10^8$ ops/sec. For $n = 10^6$, $\log n \approx 20 \rightarrow 10^6 \times 20 = 2 \times 10^7$ ✓
$\leq 10^7$ ops	$O(n \log n)$, $O(n \sqrt{n})$	For $n = 10^5$, $\log n \approx 17 \rightarrow 1.7 \times 10^6$ ops ✓ For $n = 10^5$, $\sqrt{n} \approx 316 \rightarrow 3.1 \times 10^7$ ops ✓
$\leq 10^6$ ops	$O(n^2)$	For $n = 1000 \rightarrow 10^6$ operations = acceptable
$\leq 10^5$ ops	$O(n^2)$, $O(n \sqrt{n})$ (small constants)	For $n = 300 \Rightarrow 90,000$ ops ✓ For $n = 1000$, $\sqrt{n} = 31 \rightarrow 31,000$ ops ✓
$\leq 10^4$ ops	$O(n^3)$, $O(n^2 \log n)$	For $n = 100 \Rightarrow 10^6$ ops ✓ For $n = 300$: $90,000 \times \log n \approx 9 \times 10^5$ ✓

Use this to estimate complexity just from time limit and n.

Handwritten notes in red:

- $10^2 \rightarrow n^3$
- $10^6 \rightarrow 10^8$
- $10^2 \rightarrow 10^6$

Arr = [], $n = 10^6$ →
✓ $1 \leq n \leq 10^6$

$O(n)$ → $\log n$ → $n \log n$
 10^8 $7LC$
↓

C_1 → $n \log n$ → \log_2
↪ $(n \log_2 n)$

→ $10^6 \times \log_2 (10^6)$

$$n = \underline{\underline{10^6}}$$

$$C_1 \rightarrow n \log_2 n \rightarrow$$

$$\underline{\underline{10^6}}$$

$$\log_2(10^6) \Rightarrow \underline{\underline{20}}$$

$$\log_2(10^6) \rightarrow$$



$$\frac{\log_{10}(10^6)}{\log_{10}(2)} \Rightarrow$$

$$\frac{6 \times \log_{10} 10}{\log_{10} 2} \Rightarrow 0.3010$$

$$\frac{6}{0.3010} \Rightarrow$$

$$19.98 \Rightarrow$$

$$\underline{\underline{20}}$$

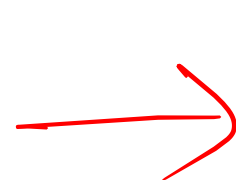
$$n = 10^6$$

$$n \log_2 n \rightarrow 10^6 \times 20 \Rightarrow 2 \times 10^1 \times 10^6$$

$$\underline{\underline{2 \times 10^7}} < \underline{\underline{10^8}}$$

$$\geq 10^5 \text{ to } 10^8 \rightarrow$$

$(n \log n, n, \log n)$
constant



Acceptable

Tn

$$n = \underline{\underline{10^6}} \rightarrow$$

$$C_1 \rightarrow (n^2)$$

$$(10^6)^2 \rightarrow \underline{\underline{10^{12} > 10^8}}$$

$$n^2 \underline{\underline{10^8}}$$

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

$$(\underline{\underline{10^8}})^2$$

TLE

$$10^{16} = \underline{\underline{TLE}}$$

Optimal Approaches

Category	Common Techniques / Examples
1. Searching	Linear Search, Binary Search, Two Pointer, Sliding Window
2. Sorting	Bubble, Selection, Insertion, Merge, Quick, (Counting, Radix)
3. Greedy	Activity Selection, Huffman Coding
4. Dynamic Programming	Knapsack, Longest Common Subsequence (LCS), Fibonacci
5. Graph Algorithms	BFS, DFS, Dijkstra, Kruskal, Prim, Topological Sort
6. Divide & Conquer	Merge Sort, Binary Search, Quick Sort
7. Backtracking	N-Queens, Sudoku Solver, Subset Sum
8. Bit Manipulation	XOR Tricks, Count Set Bits, Check Power of Two
9. Mathematical	GCD (Euclidean), Sieve of Eratosthenes, Fast Exponentiation
10. Advanced DS	Trie, Union-Find (Disjoint Set), Segment Tree, Fenwick Tree

Introduction to Two Pointer

Two Sum Problem:-

Two Sum Problem:-

n=8, k=18

if
arr

0	1	2	3	4	5	6	7
7	4	9	6	21	8	11	17

0
1

~~7 4 9 6 21 8 11 17~~

$$(7, 4) \neq k$$

$$(7, 9) \neq k$$

$$(7, 6) \neq k$$

$$(7, 21) \neq k$$

$$(7, 8) \neq k$$

$$(7, 11) = k \longrightarrow \text{return true}$$

```

1 function twoSum(arr, target) {
2   let n = arr.length;
3   for(let i = 0; i < n; i++)
4   {
5     for(let j = i + 1; j < n; j++)
6     {
7       if(arr[i] + arr[j] === target)
8       {
9         return true;
10      }
11    }
12  }
13  return false;
14 }

```

	0	1	2	3	4	5	6	7
arr ↓	7	4	9	6	2	8	11	17

$$n=8 \rightarrow 2$$

$$n_{(2)} \Rightarrow$$

$$\frac{n!}{(n-2)! \times 2!}$$

$$n_{(2)} \Rightarrow$$

$$\frac{n!}{(n-2)! \times 2!} \Rightarrow$$

$$\frac{n(n-1)}{2} \Rightarrow$$

$$\frac{n!}{(n-2)! \times 2}$$

$$\underline{\underline{O(n^2)}}$$

$$\Rightarrow \frac{n \times (n-1) \times (n-2)!}{(n-2)! \times 2}$$

$$O(1) < O(\log n) < O(\sqrt{n}) < O(n) < O(n \log n) < O(n^2) < O(n^3) < O(2^n) < O(n!) < O(n^n)$$

Note:-

MAX value of N	Time complexity
10^8	$O(N)$ Border case
10^7	$O(N)$ Might be accepted
10^6	$O(N)$ Perfect
10^5	$O(N * \log N)$
10^4	$O(N^2)$
10^2	$O(N^3)$
10^9 /	$O(\log N)$ or $\text{Sqrt}(N)$

USE TWO POINTER IN **LAST PROBLEM**

$$n = 8$$

$$K = 18$$

Step 1 :- sort the array (\uparrow order)

	0	1	2	3	4	5	6	7
array	7	4	9	6	21	8	11	17

	0	1	2	3	4	5	6	7
array	4	6	7	8	9	11	17	21

L H

Two pointer

Step 1 \rightarrow Array must be in sorted order

$$7 + 11 \rightarrow \textcircled{18}$$

$$A[L] + A[H] \text{ vs } K$$

X

$$A[L] + A[H] = K$$

true

X

$$A[L] + A[H] < K$$

L++

✓

$$A[L] + A[H] > K$$

H--

arr \hookrightarrow

0	1	2	3	4	5	6	7
4	6	7	8	9	11	17	21

$$n = 8$$

$$k = 18$$

```

1 function twoSum(arr, target)
2 {
3   arr.sort((a, b) => a - b);
4   let left = 0, right = arr.length - 1;
5   while (left < right)
6   {
7     let sum = arr[left] + arr[right];
8     if (sum === target)
9       return true;
10    else if (sum < target)
11      left++;
12    else
13      right--;
14  }
15  return false;
16 }

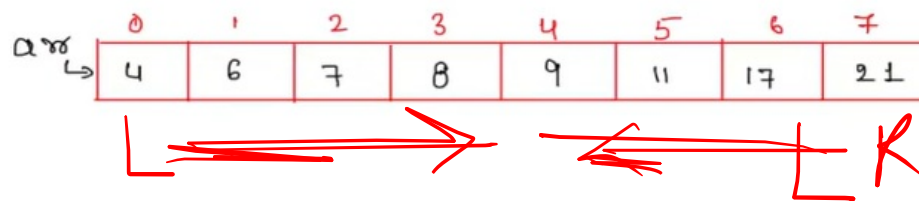
```

$n \log n$

$L < R$

$L \neq R$

$R > L$



$\text{sort}(\text{arr.sort}(), \text{arr}(\text{end}()))$

$n \log n + n$

$n \log n < n^2$

Remove Duplicates from Sorted Array

Input: `arr[] = [2, 2, 2, 2, 2]`

Output: `[2]`

Explanation: All the elements are 2, So only keep one instance of 2.

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

Output: `[1, 2, 3, 4, 5]`

Input: `arr[] = [1, 2, 3]`

Output: `[1, 2, 3]`

Explanation : No change as all elements are distinct.

Specific

Linked List

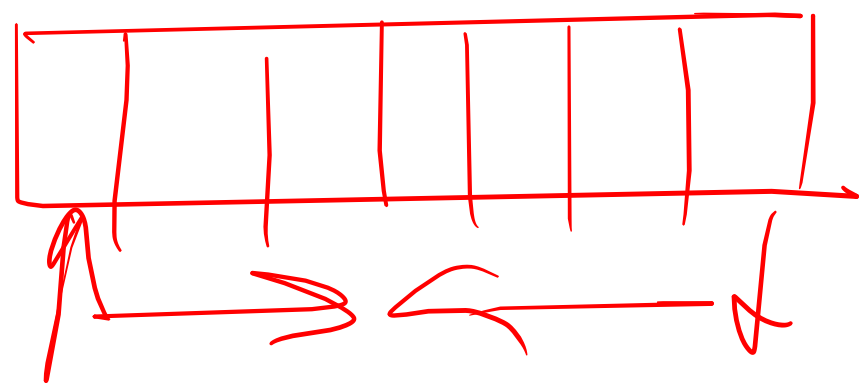
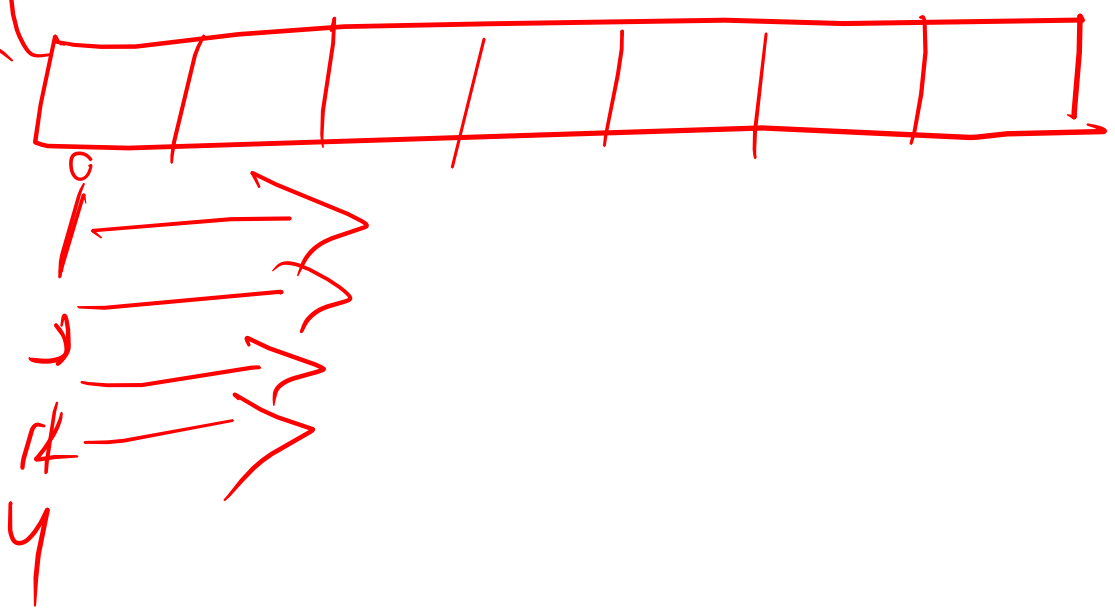
Two Pointer

1

Same Direction

2

Opposite Direction



Opposite Direction

1) Find a pair whose sum is equal to k [$a+b=k$]

array

0	1	2	3	4	5	6	7
7	4	9	6	21	8	11	17

$$n=8$$

$$k=16$$

```
function chkPair(arr,n,k)
{
    for (i = 0; i < n-1; i++)
    {
        for (j = i + 1; j < n; j++)
        {
            if (arr[i] + arr[j] == k)
            {
                return true;
            }
        }
    }

    return false;
}
```

	0	1	2	3	4	5	6	7
array ↳	7	4	9	6	21	8	11	17

$$n = 8$$

$$K = \underline{16}$$

0	1	2	3	4	5	6	7

```
function chkPair(arr,n,k)
{
    arr.sort();
    l=0, r=n-1;
    while(l<r)
    {
        if(arr[l]+arr[r]==k)
            return true
        else if(arr[l]+arr[r]<k)
            l++;
        else
            r--;
    }
    return false;
}
```

2) Find a triplet whose sum is equal to k [$a+b+c=k$]

	0	1	2	3	4	5	6	7
arr	7	4	9	6	2	8	11	17

Handwritten annotations: Red circles around indices 0, 1, 2, 3, 4, 5 and values 7, 4, 9, 6, 2, 8. A red line connects indices 0, 1, 2. A green line connects indices 1, 2, 3. A blue line connects indices 2, 3, 4.

$$(7, 4, 9) == k$$

$n = 8$
 $k = 33$

$O(n^3)$

$L1$ for ($i \rightarrow n-2$)
 $L2$ for ($j \rightarrow n-1$)
 $L3$ for ($k \rightarrow n$)

	0	1	2	3	4	5	6	7
array	7	4	9	6	21	8	11	17

	0	1	2	3	4	5	6	7

```

function chkPair(arr,n,k)
{
    arr.sort();
    for(i=0; i<n-2; i++)
    {
        l=i+1;
        r=n-1;
        while(l<r)
        {
            if(arr[i]+arr[l]+arr[r]==k)
                return true;
            else if(arr[i]+arr[l]+arr[r]<k)
                l++;
            else
                r--;
        }
    }
    return false;
}

```

Handwritten annotations on the code:

- Arrows pointing to `arr.sort()` and the `for` loop with the label (n) .
- An arrow pointing to the `while` loop with the label (n) .
- Circles around `l++` and `r--`.
- A circle around the `if` condition `arr[i]+arr[l]+arr[r]==k`.
- A bracket on the left side of the `while` loop.

	0	1	2	3	4	5	6	7
arr	7	4	9	6	21	8	11	17

	0	1	2	3	4	5	6	7
	4	6	7	8	9	11	12	21

Handwritten annotations on the table:

- An arrow pointing from index 3 to index 5.
- A circle around index 1 with an arrow pointing to index 5.
- A circle around index 6 with an arrow pointing to index 1.

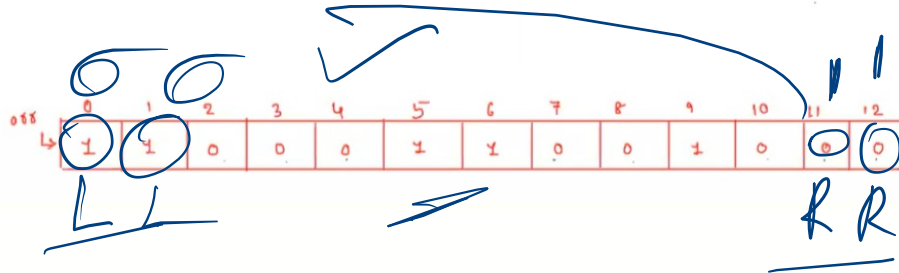
$10^3 \rightarrow 1000$

$\underline{\underline{O(n^2) \rightarrow 100}}$

$\underline{\underline{n \log n}} + \underline{\underline{nL}}$

3) Separate 0's and 1's

$M \pm \rightarrow 2ptr$ (opposite - dir)



o/p

0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	1	1	1	1	1

$n = 13$

App1

$\rightarrow \text{sort}$
 $(n \log n)$

App2

$\rightarrow \text{Extra } O(1)$
 $\rightarrow O(n)$ ~~$O(n)$~~

App3 $\rightarrow O(n)$ ✓

if array is already sorted?

$O(n)$

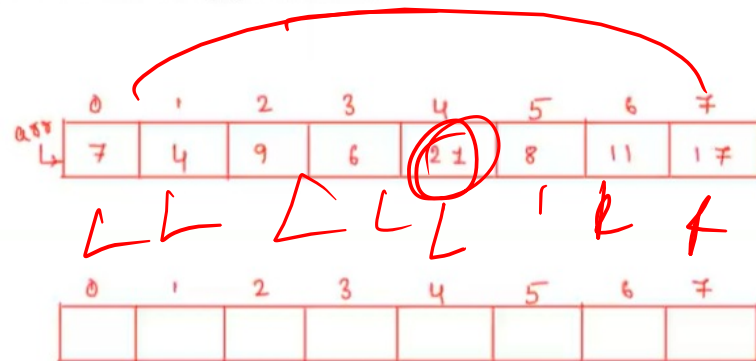
```
function segregate0and1(arr, n)
{
    let left = 0, right = n-1;

    while (left < right)
    {
        /* Increment left index while we see 0 at left */
        while (arr[left] == 0 && left < right)
            left++;

        /* Decrement right index while we see 1 at right */
        while (arr[right] == 1 && left < right)
            right--;

        /* If left is smaller than right then there is a 1 at left
        and a 0 at right. Exchange arr[left] and arr[right]*/
        if (left < right)
        {
            arr[left] = 0;
            arr[right] = 1;
            left++;
            right--;
        }
    }
}
```

4. Reverse the array [in-place]



L L R

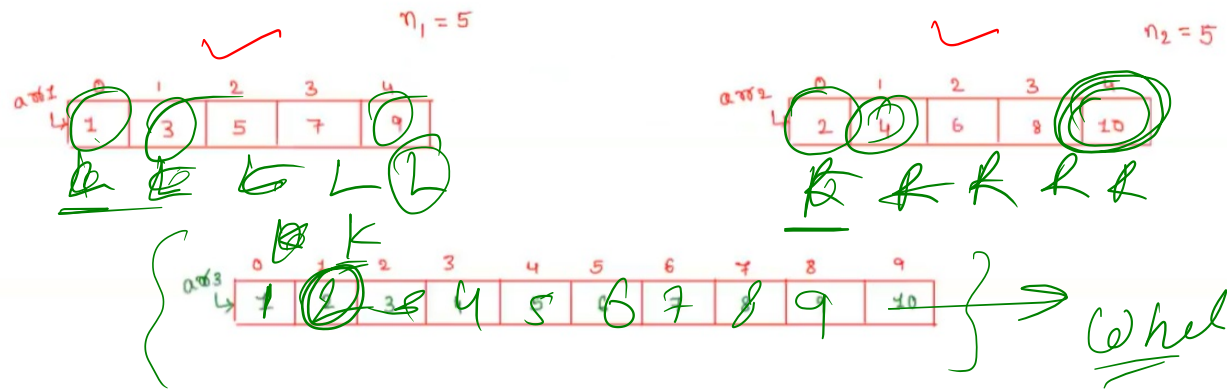
Interview

$O(n)$

Same Direction Pointer

5) Merge Two Sorted Arrays

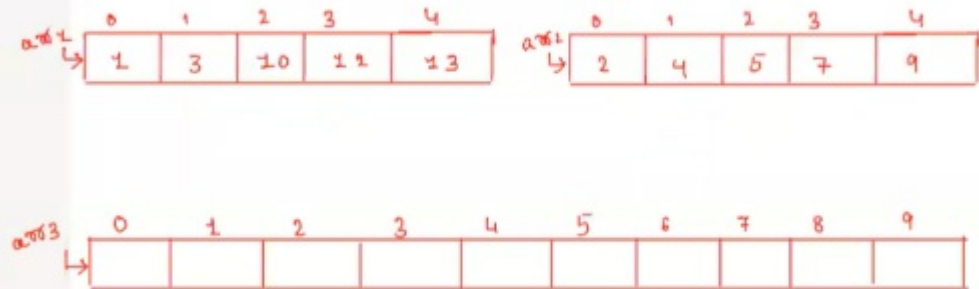
$O(n)$



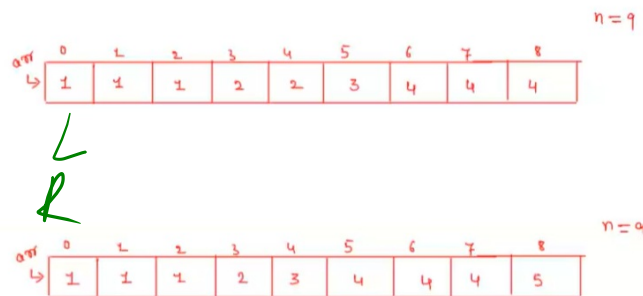
```

function mergeTwoSortedArrays(arr1,n1,arr2,n2,arr3,n)
{
    i=0, j=0, k=0
    while(i<n1 && j<n2)
    {
        if(arr1[i]<arr2[j])
        {
            arr3[k]=arr1[i]
            i++
            k++
        }
        else
        {
            arr3[k]=arr2[j]
            j++
            k++
        }
    }
    while(i<n1)
    {
        arr3[k]=arr1[i]
        i++
        k++
    }
    while(j<n2)
    {
        arr3[k++]=arr2[j++]
    }
}

```



6) Remove Duplicates from Sorted array



Assignment

Two Pointers

```

function removeDupSortedArray(arr, n)
{
    j=0
    for(i=0; i<=n-2; i++)
    {
        if(arr[i] != arr[i+1])
        {
            arr[j] = arr[i]
            j++
        }
    }
    arr[j] = arr[n-1]

    for(i=0; i<=j; i++)
    {
        print(arr[i])
    }
}

```

arr

0	1	2	3	4	5	6	7	8
1	1	1	2	2	3	4	4	4