* Given an array of only 0's, 1's & 2's (1,2,2,2,1)
Sort the given array without using any sorting algorithm. (No Built-in functions allowed)

Sample I/p → int[] arr = {2,1,2,1,0,2}
Expected O/p → int[] arr = {0,1,1,2,2,2}

[Time Complexity → O(n) → Linear
Space Complexity → O(1)]

Extra DS not allowed.
O(n) ✗

c0=1 c1=2 c2=3
0,1,2,2,2

index = 0
c0 = 1-1=0 | 2|1|2|0|1|2|
c1 = 2-1-1=0
c2 = 3-1-1-1=0

while (c0>0){ → O(n)
arr[index++]=0;
c0--;
}
+
while (c1>0){
Same for c1; → O(n)
}
+
while (c2>0){
Same for c2; → O(n)
}

[(DP) → R+M+T+SO] Dry Run O(n)

(O/p)
arr | 0 | 1 | 1 | 2 | 2 | 2 |
0 1 2 3 4 5
index index

≈ O(n)
O(1)

* Arrays → Imp for DSA → Jagged Arrays
* Kadane Algorithm? → Maximum Subarray Sum O(n)

int[] arr = {5,-8,1,2,-1,4} arr[i]=4
0 1 2 3 4 5
int cmax = arr[0] = 5 -3 1 3 2 6 cm = arr[i], cm+arr[i]
int gmax = arr[0] = 5 6
for(int i=1; i<n; i++){ = 4, 2+4
cmax = max(arr[i], cmax+arr[i]); 4,6
} gmax = max(cmax, gmax); = 6
return gmax; → 6 [DRY RUN]

Derivation: →
(Search Space) (Key)
N
mid = s+e/2 → (OOB) index
mid → optimal formula → = [s + (e-s)/2]
= 2s+e-s/2
= [s+e/2]
M+M
= S+e (-ve)/2

N = 2^0-k
2^1-k
2^2-k
2^3-k
Max Integer → -2^31 to 2^31 - 1
1000 +1
(Max + Max) = (-ve)
N/2 N/2
N/4 N/4 N/4 N/4
N/2^k
k is constant → 0,1,2,3,4,...
k = log2 N = log N
m =

Percentage Allocation of Topics for Coding Interviews

40 %	10 %	20 %	20 %	10 %
Binary Search	Arrays, Strings (Linked Lists)	Trees, Graphs, Heaps	Greedy, DP, Bit Masking	Recursion + Backtracking
Euclid's Algo	Special Algos: ✓ Kadane ✓ Robin Karp ✓ Sieve of Eratosthenes (Prime nos)			

{ Built-in Data Structures
↳ C++ → (STL)
↳ Java → Collections

* Finding the square root of a number using Binary Search (log N)
log N

int sqroot(int n){
ans = mid = 3,5
s = m+1
s = m+1
}

0 — 36
s e
mid = 36/2 = 18
0 — 7
4 — 7
5x5 = 25 < 36
√n = 36 = 6x6 = (36)
11/2 = 5
13/2 = 6
7/2 = 3
3x3 = 9
18x18 = 324
e = m-1
0 — 17
17/2 = 8 (Search Space)
e = m-1 = 7
8x8 = 64 > n
6x6 = 36
37 = 6

Sorted Array
bs(int[] arr, int key)