Space Complexity:

Input array } Not counted in Extra Space

top 

print

O(n) extra space

Generally, Time & Space Tradeoff

ARRAYS:

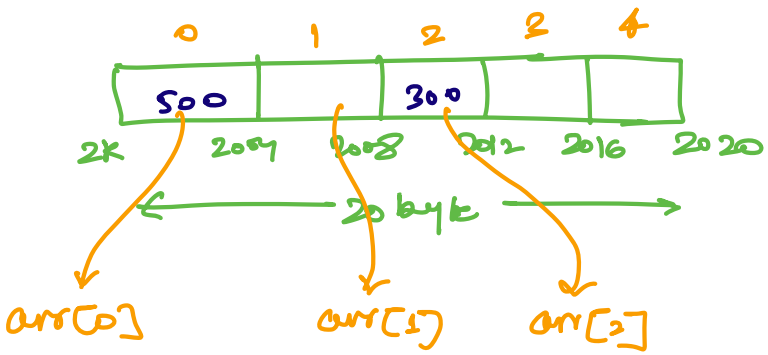
1D:

int arr[5];

↳ homogeneous collection

1 int → 4 byte

5 int → 20 byte



arr[2] = 300

base address + (idx * int size)

2000 + (2 * 4) = 2008

constant time

arr[0] = 500

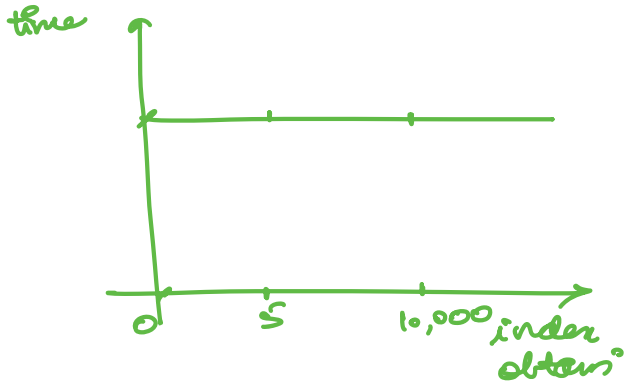
2000 + (0 * 4) = 2000

10,001 size

2K + (10000 * 4) = ____

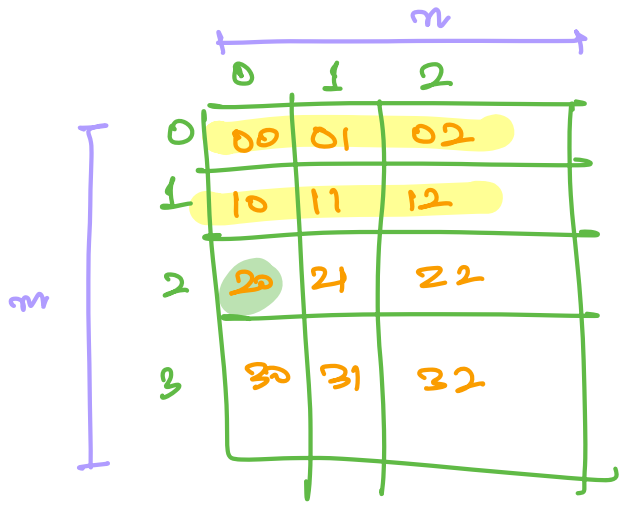
Array index update, obtain : O(1)
get/set

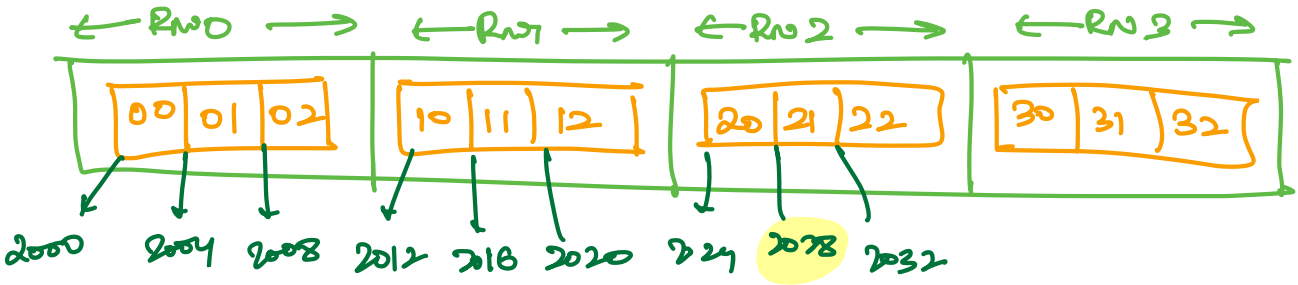
To reach 10,000 index you don't do linear traversal ✗
instead calculation, jump to memory location.



2D Arrays

- Row Major Order
- Column Major order





cout << arr[2][1]
i j

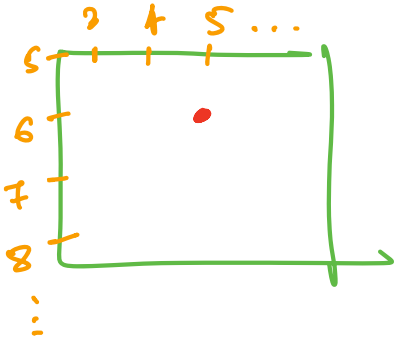
2000 + ((2*3) + 1) * int size

2000 + (7*4) = 2028

m rows
n colums

m x n

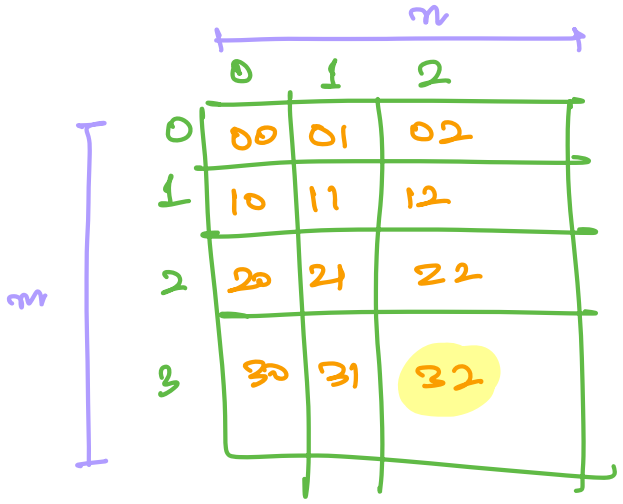
(i, j cell) → base address + (i * n + j) * data type size

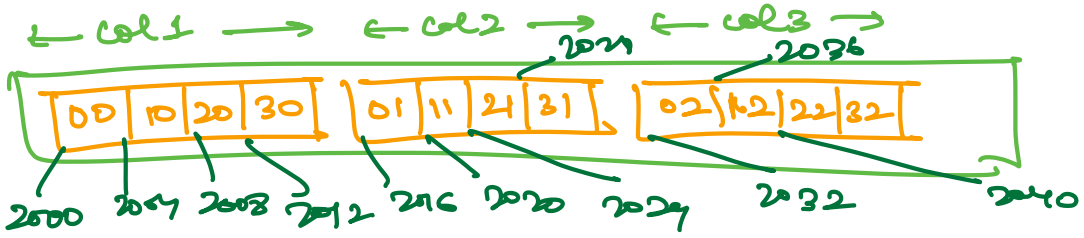


lr = 5 lc = 3 (i, j) cell

base address + ((i - lr) * n + (j - lc)) * size

Column major Order





3, 2 :

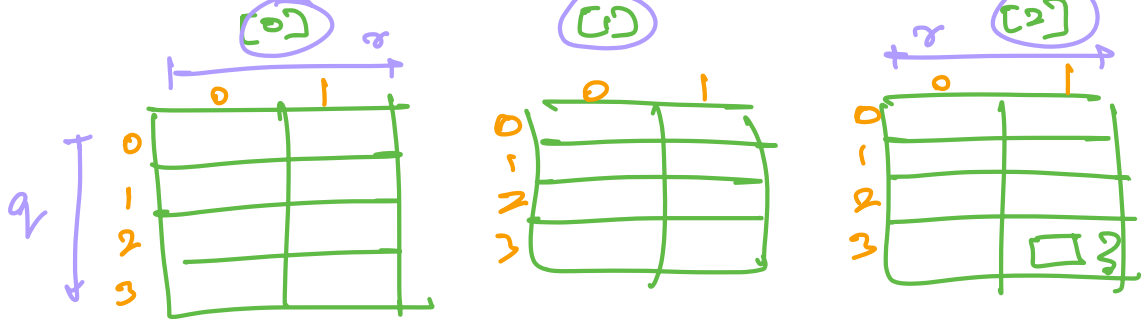
(i, j)

base address + (j * m + i) * size

3D Arrays

3 x 4 x 2 : 3, 2D Arrays of size 4 x 2

int arr[3][4][2]
p q r

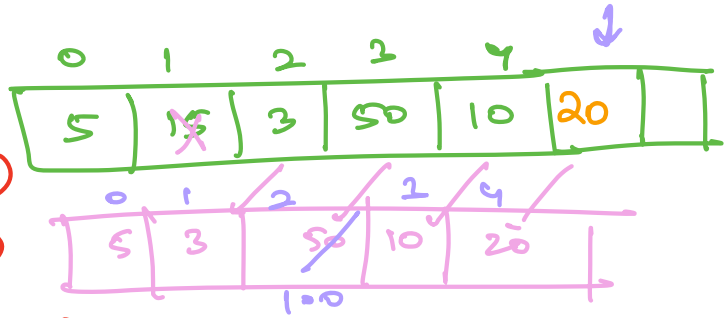


arr[2][3][1] ?

base address + (i * q * r + j * r + k) * int size

Array operations

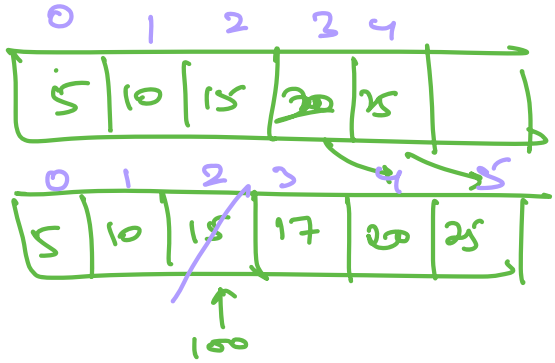
Unsorted



- Insert — arr[i] = item : O(1)
- Delete → item is ? : O(n)
- update → 2nd exp → 100
  arr[2] = 100 O(1)

- Insert → loop O(n)
- Max → loop O(n)
→ Find → loop O(n)

Sorted



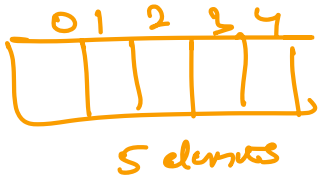
- Insert — 17 insert O(n)
- Delete O(n)
- update arr[2] = 100 O(1)

- Insert : O(1) 0 index
- Max : O(1) last index
→ Find : O(log n)

Dynamic Arrays

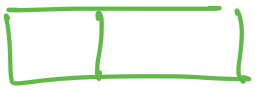
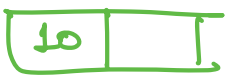
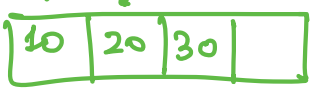
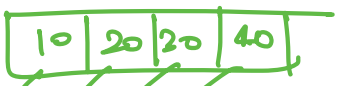
C++: vectors
Java: ArrayList

int arr[5];



DA: Size don't have to worry.

vector <int> v; ⟶ internally array

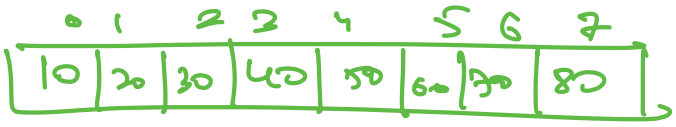
	Size	Capacity	
	0	2	
push_back(10)	1	2	
(20)	2	2	
(30)	3	4	2 copy+1 
(40)	4	4	1 
(50)	5	8	4 copy +1 
4 push_back	6	8	1
	7	8	1
	8	8	1
	9	16	

2 push_back = 4 ops
4 push_back = 8 ops

1 push_back → 2 ops
1 push_back → 2 ops
} O(1)

Insert: Size == Capacity Double size Array copy.

Delete: Size = Cap/2 Array half



S=8 C=8

delete 10 20 30 40 50 60 70 _ S=7 C=8

delete 10 20 30 40 50 60 _ _ S=6 C=8

delete 10 20 30 40 50 _ _ _ S=5 C=8

delete 10 20 30 40 _ _ _ _ S=4 C=8

(4)

S=4 C=4

add 4+1 10 20 30 40 50 _ _ _ _ S=5 C=8

delete 4 10 20 30 40 S=4 C=4

add 4+1 10 20 30 40 50 _ _ _ S=5 C=8

→ $\frac{4+4+4+4+2^x}{4 \cdot 4}$

4 push pop → 16 ≠ 2

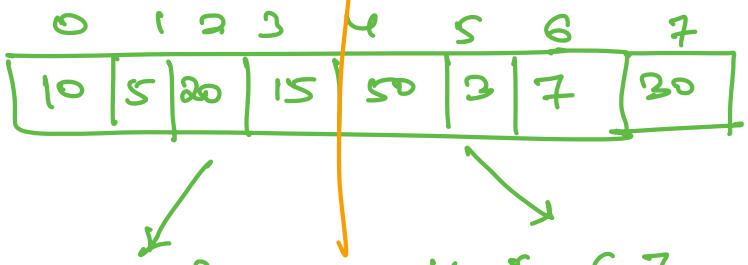
1 → 4 ops ≠ n

Size = Cap/4 Array half

Linear Search | B.S, S.S, I.S
Binary Search |

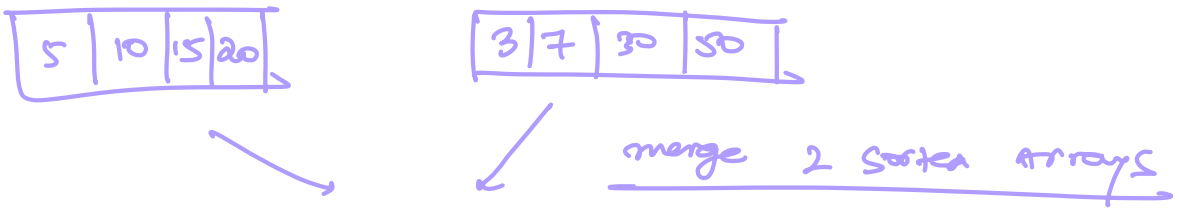
Merge Sort

Divide & Conquer Algo

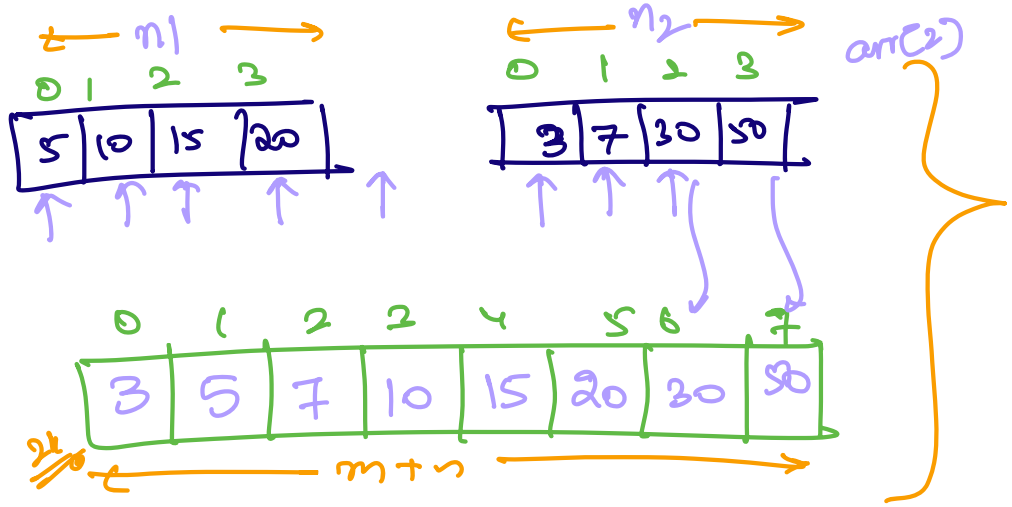

0 1 2 3 4 5 6 7
10 5 20 15 50 3 7 30

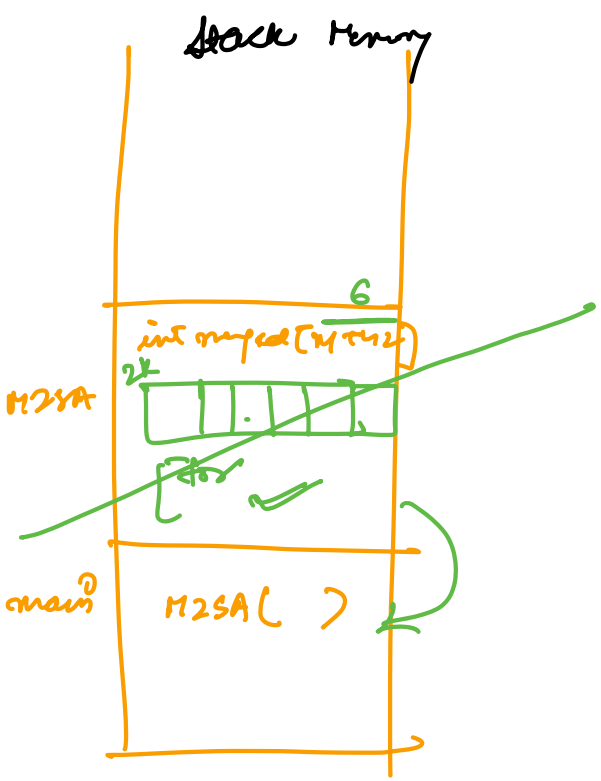
Divide

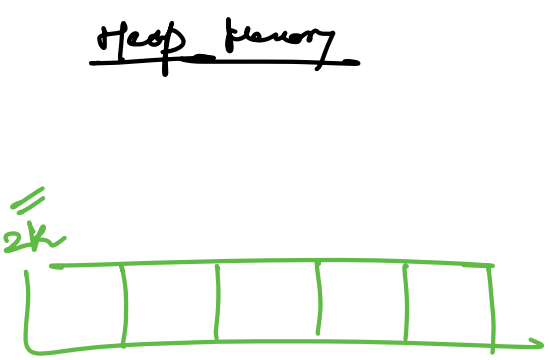

0 1 2 3
10 5 20 15
4 5 6 7
50 3 7 30

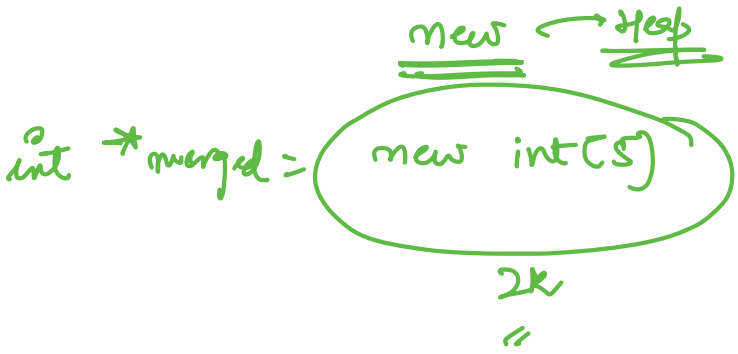

5 10 15 20
3 7 30 50
merge 2 sorted arrays

Q:


n1
n2
arr1
arr2
arr[2]
0 1 2 3
5 10 15 20
3 7 30 50
0 1 2 3 4 5 6 7
3 5 7 10 15 20 30 50
m+n


Stack Memory
6
int merged[m+n2]
2k
M2SA
main
M2SA( )


Heap Memory
2k



vector<int> v;

vector<int> v = new vector();