Q Given an array A of N integers. The value of subarray is defined as a bitwise OR of all elements in it.

Return the sum of value of all the subarrays.

$N \leq 10^5$

$A[i] \leq 10^8$

ex: [1, 2, 3, 4] ← value

[1] → 1

[1,2] → 3

[1,2,3] → 3

[1,2,3,4] → 7

[2] → 2

[2,3] → 3

[2,3,4] → 7

[3] → 3

[3,4] → 7

[4] → 4

return result.

ans = 1+3+3+7+2+3+7+3+7+4.

= 40

1 | 2 | 3

3 | 3 = 3.

OR

0	0	0
0	1	1
1	0	1
1	1	1

4 2 1

7 → 1 1 1

4 → 1 0 0

BF

1. Find all subarrays. ($O(N^2)$)
2. Take the OR of each subarray (N)
3. Take & return sum. ($O(1)$)

TC → $O(N^2)$

10^5

$O(N)$

$N \log N$

Q 10^8 → 31 or 32 N (given)

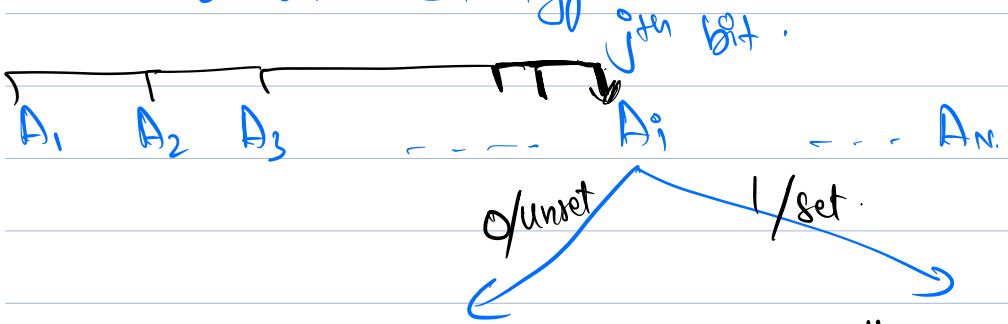
$2^{10} \approx 1024 \approx 10^3$

$2^{20} \approx 10^6$

$2^{30} \approx 10^9$

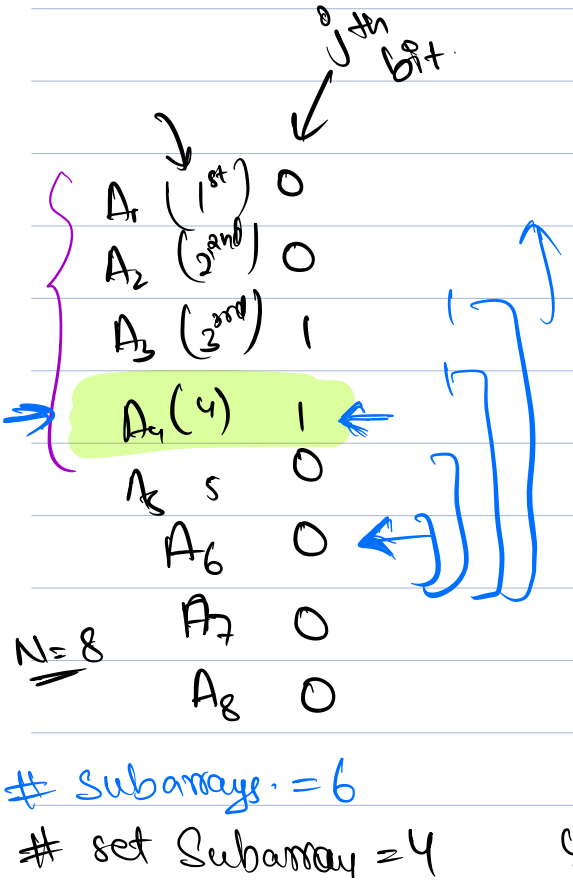
30 bits.

Contribution strategy



# of subarray = i

contribution = $i \times 2^j$



Subarrays. = 6

set Subarray = 4

jth →

A = [1, 2, 3, 4] 2^0

1 →	0	0	1
2 →	0	1	0
3 →	0	1	1 ← j
4 →	1	0	0

subarray → 3

$2^0 * 3 = 1 \times 3$.

count $2^j * i$ → if bit is set

Last Set Index (LSI)

contribution = $LSI * 2^{i}$ → If bit is unset.

i				
1	0	0	1	$2^0 \times i = 2^0 \times 1 = 1$
2	0	1	0	$2^1 \times LSI = 2^0 * 1 = 1$
3	0	1	1	$2^0 \times i = 2^0 * 3 = 3$
4	1	0	0	$2^2 \times LSI = 2^0 \times 3 = 3$
				8

0th bit → 8

2^1

1	0	⟷	$2^1 \times 0$
2	1	⟷	$2^1 \times 2 = 4$
3	1	←	$2^1 \times 3 = 6$
4	0	←	$2^1 \times 3 = 6$

1st bit = 16

1	0	→	0×2^2 → 2^0
2	0	→	0×2^2
3	0	→	0×2^2
4	1	→	$4 \times 2^2 = 4 \times 4 = 16$

2nd bit = 16

$16 + 16 + 8 = 40$

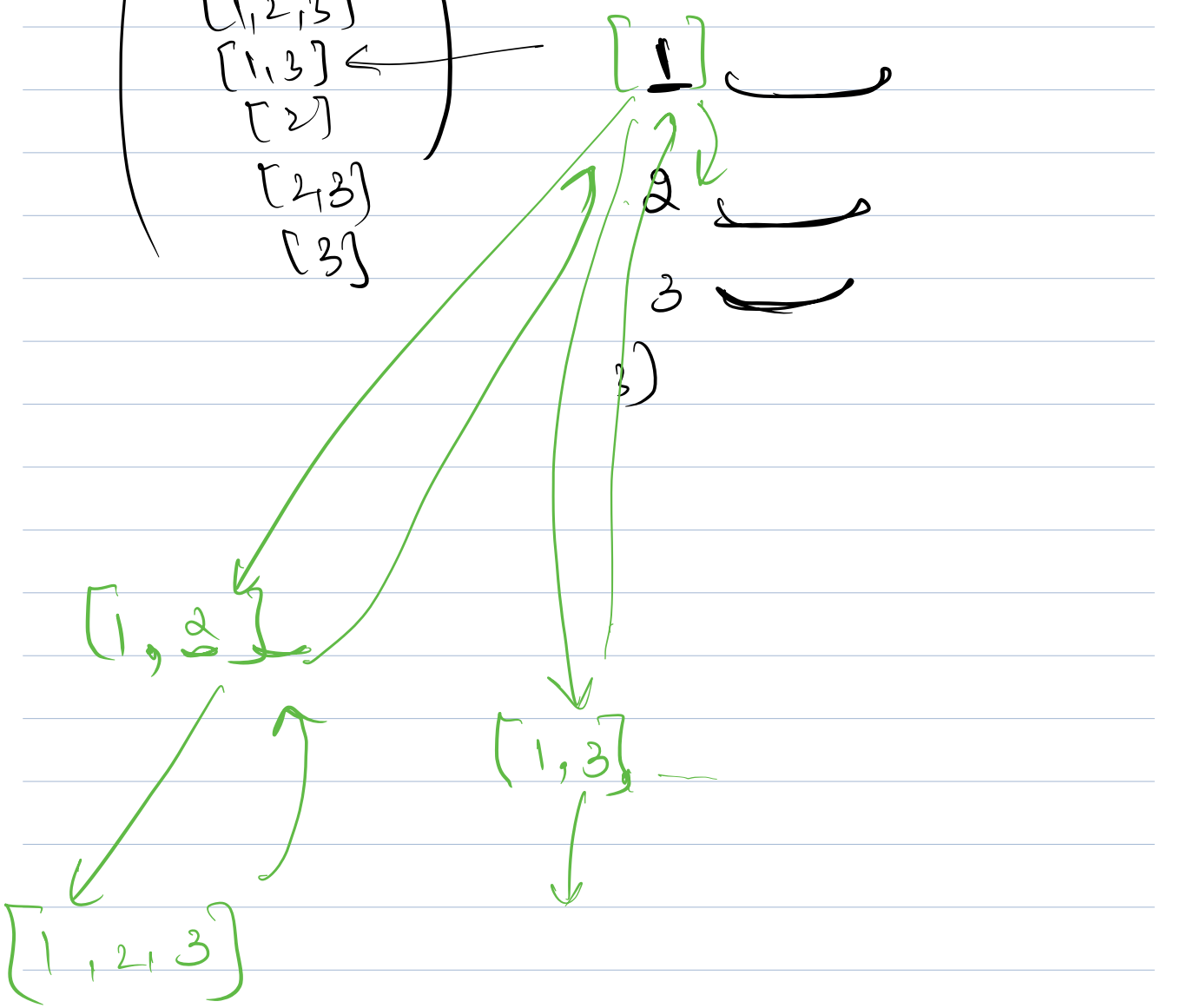
Given a sorted array A of distinct integers. Return all possible subsets of A

Notes:
- Subsets ↑ lexicographically
- No duplicates in set.

ex: A = [1,2,3]

[ ]
[1]
[1,2]
[1,2,3]
[1,3]
[2]
[2,3]
[3]

} Subsets.



Pseudo code.

```
ans = { }
temp = { }

findSubset(A, idx)
    ans.add(temp)                    //
    for(i = idx; i < A.length; i++) {
        temp.add(A[i])               // do
        findSubset(A, i+1)
        temp.removeLast()            // undo
    }
```

ans = { [ ] [1] [1,2] [1,2,3] } temp = (0, 1, 2) {1, 2, 3}.

findSubset(A,0)

findSubset(A,1)

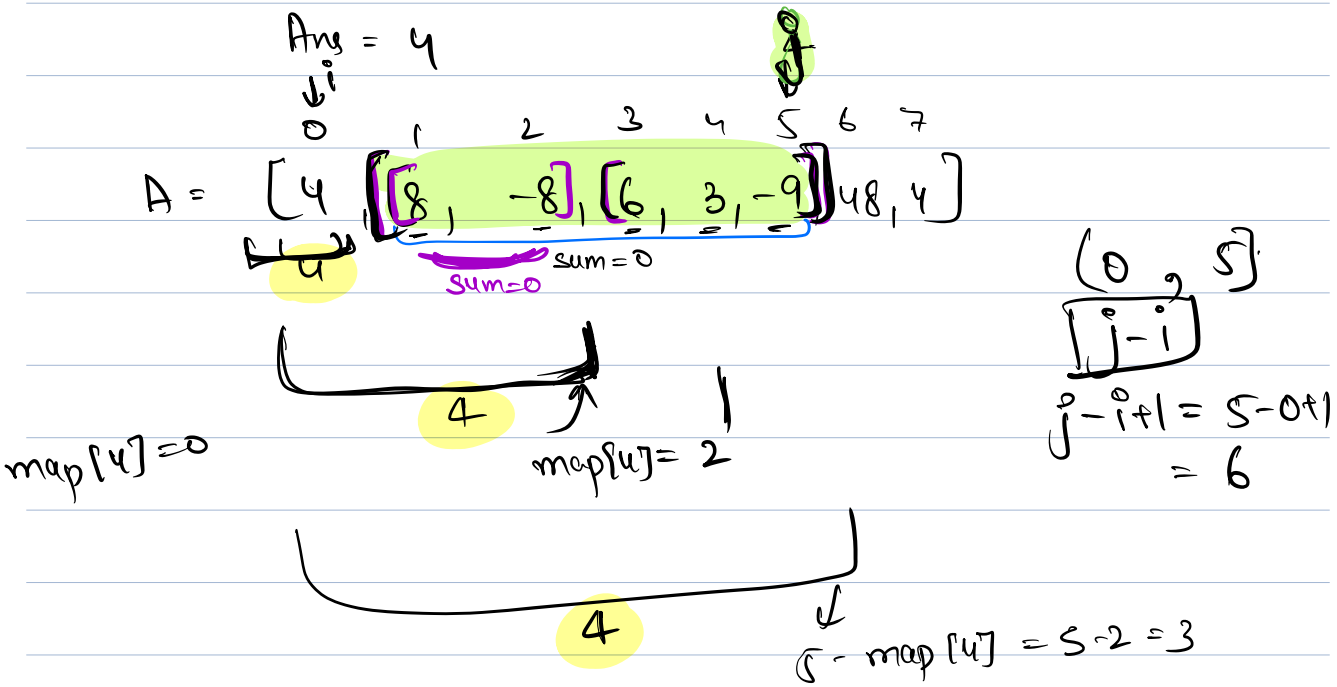
fS(A,2)

fS(A,3)

Q prefix Sum.

Q Given an array A of N integers find the largest continuous subsequence in the array which sums to zero & return length.

ex: A = [1, 2, -2, 4, -4]



1. Find prefix Sum
2. Hashmap to find if that sum had already occured, if yes at what index.

Code:

```
prefixSum[0] = A[0]
for (i=1; i<A.size(); i++)
    prefixSum[i] = A[i-1];
```

```
}.                    sum      Index.
HashMap<Int, Int> map = new HM<>
ans = 0;
for(i=0 ; i<A.size() ; i++) {.
    if (map.contains(prefixSum[i])) {
        ans = max(ans, i - map[ps[i]])
        continue;
    }
    map[ps[i]] = i
```

Code for Q1

```
setBit[N][31]                                   // 28
for(j=0 ; j<30 ; j++) {.
    int LSI = 0
    for(i=0 ; i<A.length ; i++) {.
        if((A[i] & (1<<j)) > 0)                 // Bit set
            setBit[i][j] = i+1
            LSI = i+1;
        } else                                  // Bit is unset
            setBit[i][j] = LSI
}.
ans += ((1<<j) % (10^9+7) * setBit[i][j]) % (10^9+7)
```

000
001
010
011

[1] [2] [1,2]
[1] [1,2] [2]

Q Given an array A comprising of 0's & 1's & an integer B. Find indices s.t. there are B alternating 0 & 1 on left of that idx.

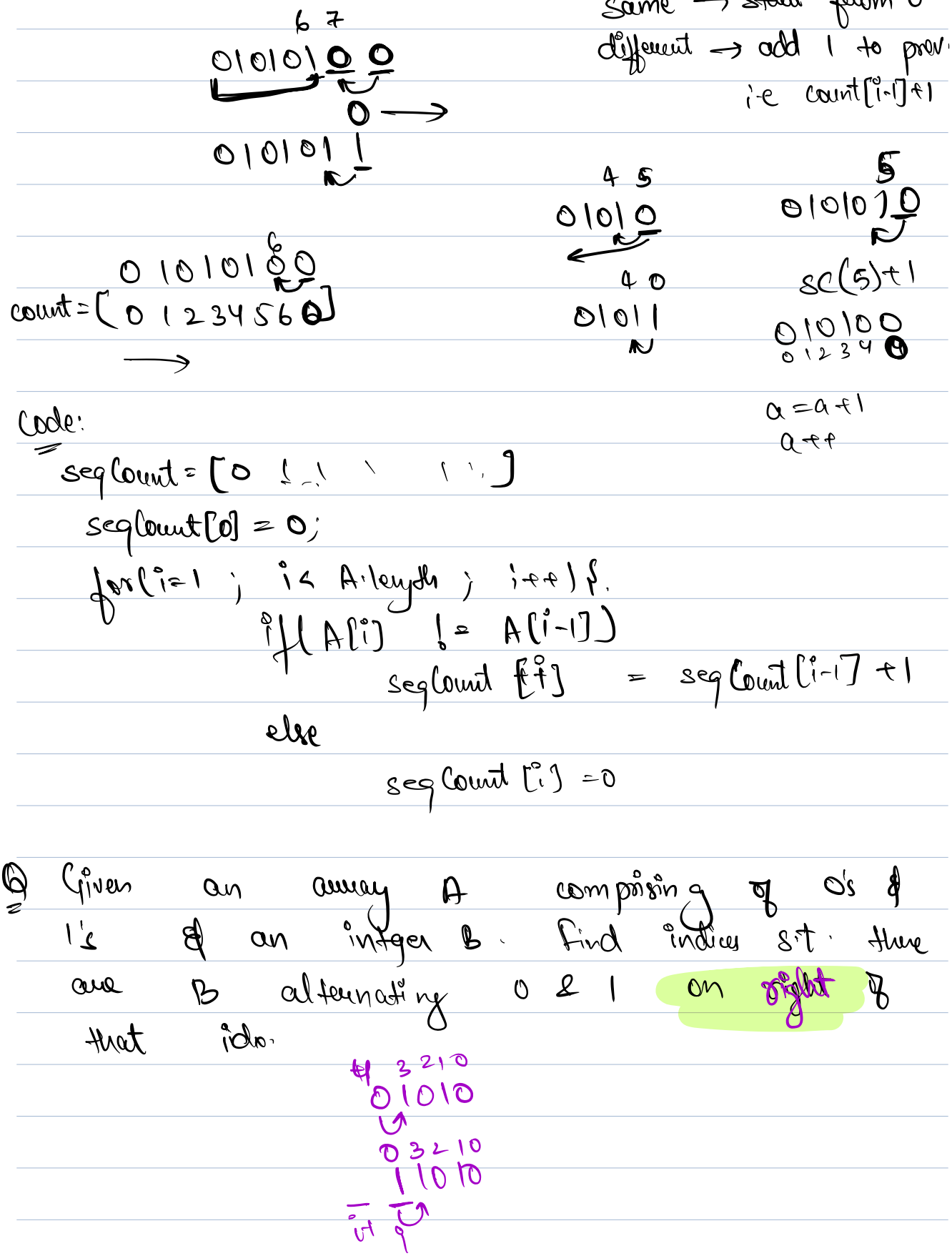
ex: [1, 0, 1, 0, 1, 1]
0 1 2 3 4 5

B=0 [0, 1, 2, 3, 4]
B=1 [1, 2, 3, 4]
B=2 [2, 3, 4]

ex: [0, 0, 0, 1, 1, 0, 1]
0 1 2 3 4 5 6

B=1 [5, 6]
1

B.F: for every idx, check sequence on left
O(N²)



Code:

```
seqCount = [0 ! . ! \ ! ! . ]
seqCount[0] = 0;
for(i=1 ; i< A.length ; i++){
        if( A[i] != A[i-1])
                seqCount [i] = seqCount[i-1] +1
        else
                seqCount [i] =0
```

Q Given an array A comprising of 0's & 1's & an integer B. Find indices s.t. there are B alternating 0 & 1 on right & that idx.

```
seqCount = [0 ! _ \ \ ' ']
seqCount[0] = 0;
for(i = A.length-2 ; i >= 0 ; i--) {
    if(A[i] != A[i+1])
        seqCount[i] = seqCount[i+1] + 1
    else
        seqCount[i] = 0
```

Q- Given an array A of length N comprising of 0 & 1 & an integer B. Find indices of A that can be treated as a center for 2*B+1 length of alternating 0-1 subarray.

2*B + 1

2*1 + 1 = 3

2*B + 1

A: [1 0 1 0 1]

0 1 2 3 4

B = 2

[2]

B = 1

[1, 3, 2]

Q) A of N integers & an element B.

0 1 2 3 4

A = [5, -2, 3, 1, 2] B = 3

s e-B e

[5, -2, 3, 1, 2], [5, -2, 3, 1, 2.

(i+1) % A size

B <= A.size.

i = 1

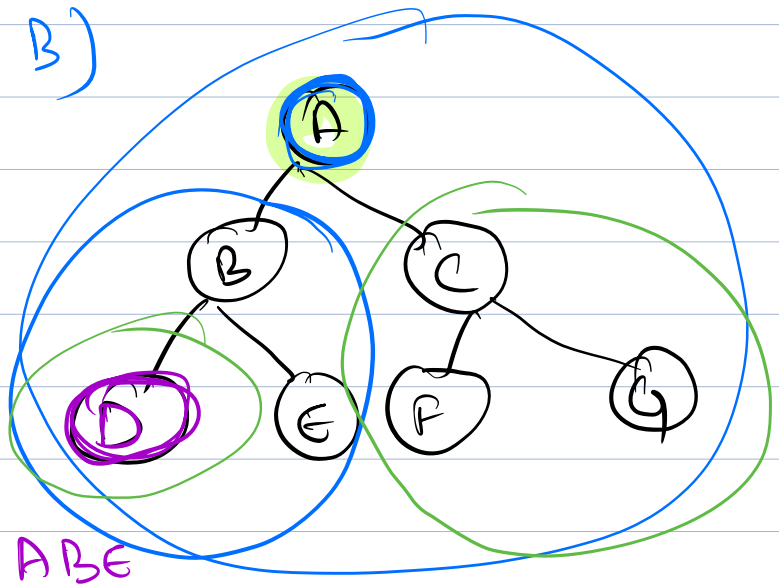
2.

i = 3
i = 4
i = 5
i = 6
i = 7

4
0
1
2
3

(-3 , 3)

(-B , B)



ABD

ABE

array = { }.

traverse (A) { → print path & return.
// if(A is null) return.
array.push(A) ✓ // did
{ traverse (A.left) ✓ } // recursion.
{ traverse (A.right) ✓ }
array.removeLastElement(); // undo.
}.

ABD ✓
ABD ✓

array = { A B ~~D~~ E }.

#BDE