ARRAYS

Friday

"[TC Doubts]",
+ Prefix Sum

Recap — container that a store a list of elements (objects) of same type.

Data Structure

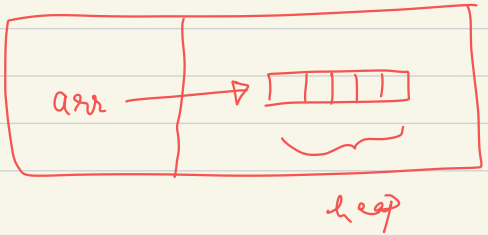
fixed

```
int [] arr = new int [5];
```

```
int [] arr = new int [n];
```

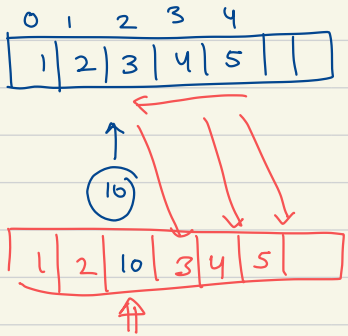
⇒ Allocate a new array if you want inc the size of array.

Allocated on heap during runtime.



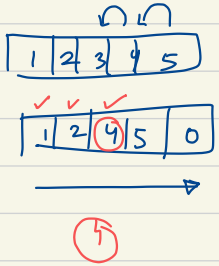
- ✓ Create ✓
- ✓ Insert ✓
- ✓ Remove ✓
- ✓ Search (Linear Search)
- ✓ Delete

Insert



Logic

Remove



Linear Search

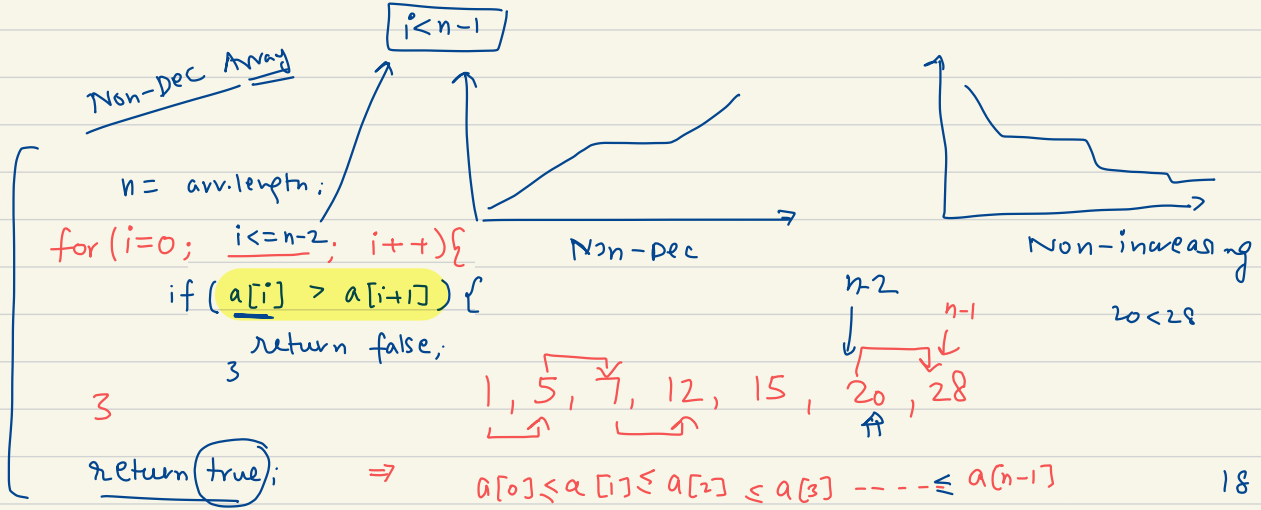
Binary Search ?

Binary Search → Given a Sorted array

Check if the array is a sorted Array ?

Q1

```
int[] arr = { 1, 5, 7, 12, 15, 20, 28 };
```



1 > 5
5 > 7
7 > 12
12 > 15
15 > 20
20 > 28

Not sorted

15 > 7

for (_____)
if (a[i] < a[i+1])
return true

1, 5, 7, 8, 10, 6, 15

will never get checked

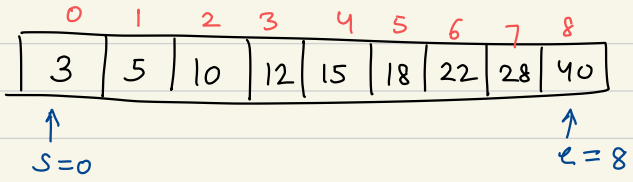
(a) Inc / Dec. 10, 8, 7, 5, 2, 1 →

boolean inc = true;

```
if (a(0) > a(1)) {
        inc = false;
    }
```

```
for ( _____ ){
    if (inc){
        a(i) > a(i+1)
    }
    else {
        a(i) < a(i+1)
    }
}
```

(9) Binary Search → technique that be used for efficient Searching in sorted arrays

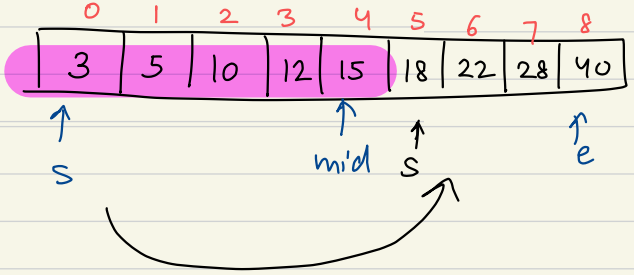


9 elements

Key = 28

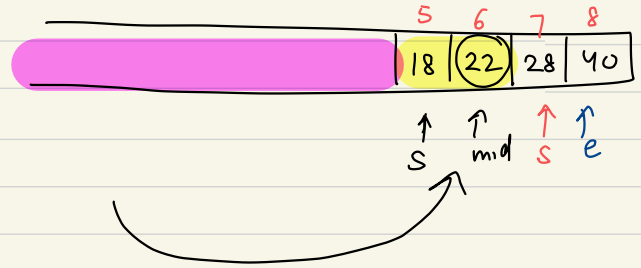
s = 0

e = arr.length - 1; mid = (s+e)/2

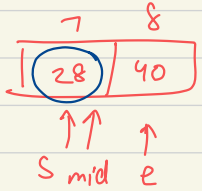



```
if (a[mid] < key){
    s = mid + 1;
}
```

28

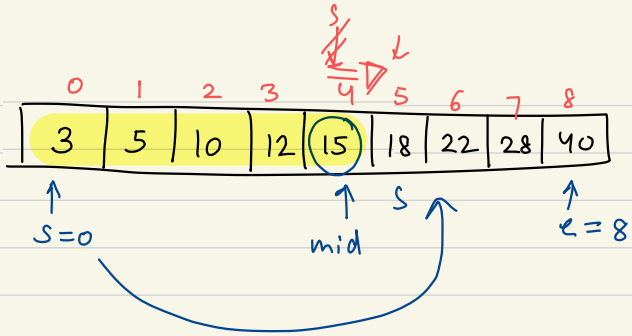


mid = $\frac{8+5}{2}$ = 6



mid = $\frac{(7+8)}{2}$ = 7

```
if (a[mid] == key)
    return mid;
```



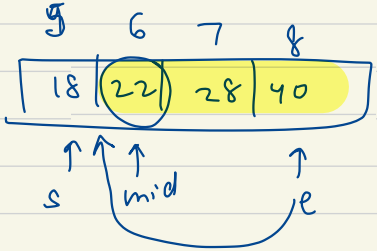
search → (18) key

```
if (a[mid] < key)
    s = mid + 1
```

if >

else if <

else ==



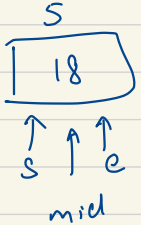
22 > 18

```
if (a[mid] > key ){
    e = mid - 1
```

3

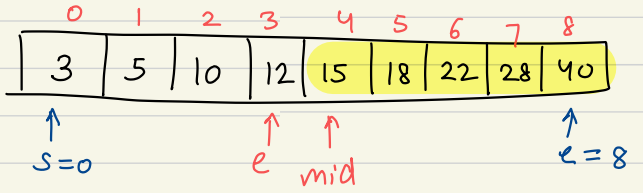
N
↓
N/2
↓
N/4
↓
.
.
.
0
1

Log N steps

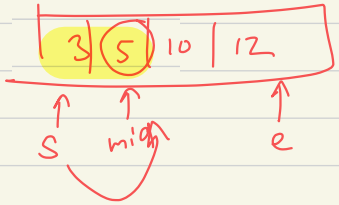


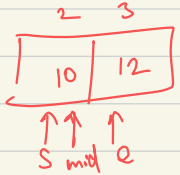
s = 5
e = 5
mid = 5

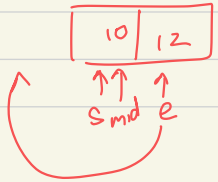
```
if (a[mid] == key)
    return mid;
```

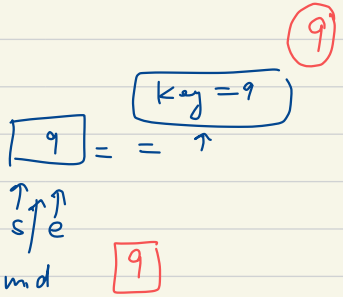


Key = 9









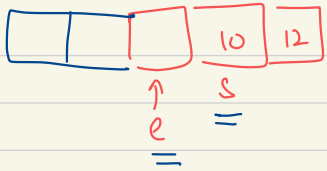
10 > key (9) → e = mid - 1

a[mid]

Stop

if (s > e)

↓ element is not present.



a[e], a[s]

s, e → Numbers

```
int binarySearch ( int [] arr , int key ){
    s = 0
    e = arr.length - 1 ;
    while ( s <= e ){
        mid = (s+e)/2;
        if (a[mid] > key) { e = mid-1; }
        else if (a[mid]==key) { return mid }
        else {
            s = mid + 1.
        }
    }
    print( Not Present )
    return -1;
}
```

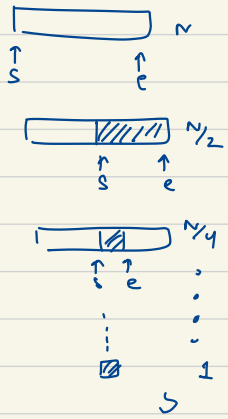
Run → false (s > e) → Not present

s > e

s < e

s ≤ e

s ≥ e



Log N times

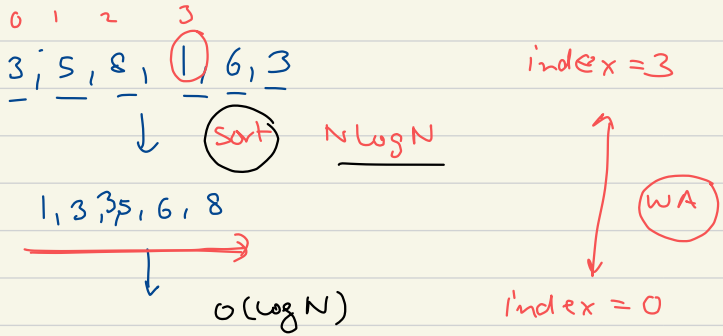
8 < 15

key mid

Time Complexity → O (Log N)

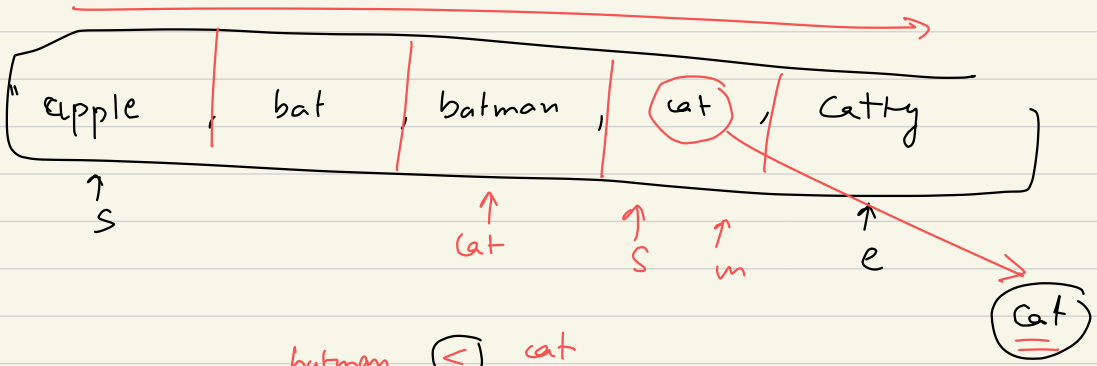
Space Complexity → O (1)

Unsorted



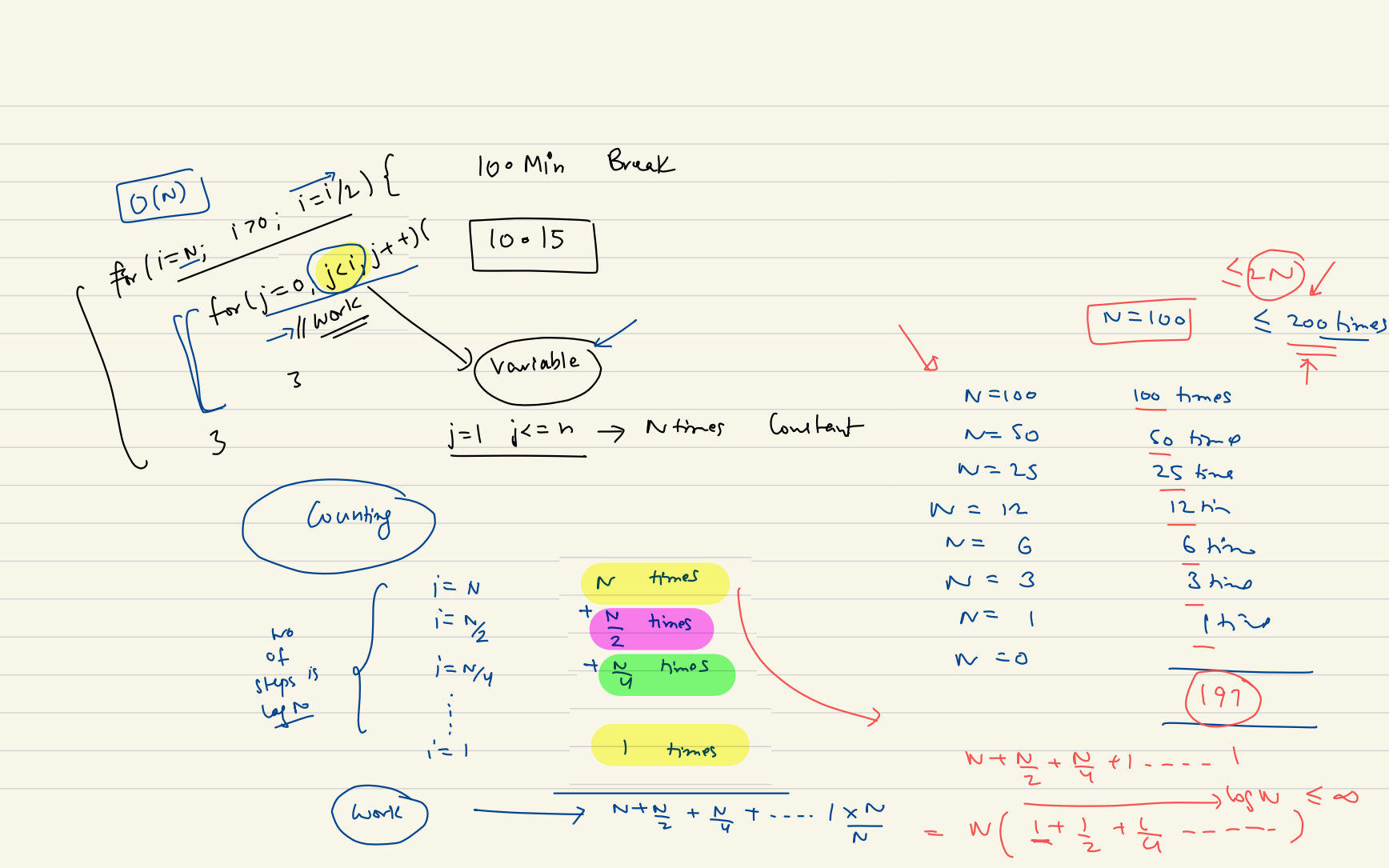
Sort + Bsearch → WA, O (N log N + log N)
= O (N log N)

Linear Search → O(N)



batman < cat

String [] names = new String [5];


O(N)
for(i=N; i>0; i=i/2){
for(j=0; j<i; j++){
// work
3
3
100 Min Break
10 to 15
Variable
j=1 j<=n → N times Constant
N=100
≤ 2N
≤ 200 times
N=100 100 times
N=50 50 times
N=25 25 times
N=12 12 min
N=6 6 times
N=3 3 times
N=1 1 time
N=0
197
Counting
No of steps is log N
i=N
i=N/2
i=N/4
i=1
N times
+ N/2 times
+ N/4 times
1 times
Work
N + N/2 + N/4 + ... 1 x N/N
N + N/2 + N/4 + 1 ....
log N ≤ 8
= N(1 + 1/2 + 1/4 ....)

$a = 1$

$r = \frac{1}{2}$

$= N\left(1 + \frac{1}{2} + \frac{1}{4} + \cdots + \frac{1}{N}\right)$ ($\log N$ terms, $\to 2$)

$= N\left(\frac{a(r^n - 1)}{r - 1}\right)$ — GP

$= N\left(\frac{1\left(\left(\frac{1}{2}\right)^n - 1\right)}{\left(\frac{1}{2} - 1\right)}\right)$ ($n \to$ no of terms $\to \infty$)

$= \frac{N}{1/2}\left(1 - \left(\frac{1}{2}\right)^{\log N}\right)$

$\leq 2N$

$= O(N)$

GP

$= N\left(\frac{a}{1 - r}\right)$

$= N \cdot \frac{1}{1 - 1/2} = 2N$

Sum of N terms of GP

$= a\frac{(r^n - 1)}{(r - 1)}$

$\log N < \infty$ ← overestimate

Seq

$1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \cdots + \frac{1}{N} \to \log N$ terms

HP

$1 + \frac{1}{2} + \frac{1}{4} + \cdots \to \infty$ terms

$= \frac{a}{1 - r}$

$= \frac{1}{1 - 1/2}$

```
i=N; i>=0 ; i/=2
for(i=1; i<=N; i=i*2){          Log N times
    for(j=1 ---- N){
        j++;
    }                           N times
}
```

$N \log N$

Log N

10.32

Q Given an array N elements, count the no elements having atleast 1 element greater than itself. (every NO)

arr = { -3, 2, 5, 8, 10, 6, 7, 8, 10, 10, -2 },

Algo-1 → go to every element a(i)

↳ iterate for the remaing 0 ——— n-1 except i
to find a greater elemnt

Bad :(

$O(N^2)$

for(i=0 ——— n-1){

// find a greter elemt for a(i)

for(i=0 ——— n-1){

3 → first occ of 10

Algo-2 { -3, -2, 2, 5, 6, 7, 8, 10, 10, 10 } // Duplicates

Sorting

↓

$O(N \log N)$ +

① Linear Search (N) → O(N)

② BS - LowerBound (logN) → O(logN)

③ N - count of 10 → ② → O(N)

$O(N \log N + \log N)$

$O(N \log N + N)$ > $O(N \log N)$

Algo-3

Simplest & fastest

① Find the largest no.

```
largest = -∞
for( i=0 ——— N ){
    if (a[i] > largest){
        largest = a[i]
    }
}
```

O(N) ⟶

② Count

O(N) {

```
cnt = 0
for( i=0 ——— N ){
    if(a[i] == largest)
        cnt ++
}
```

③ Ans = N − cnt

= 10 − 3

= 7

→ N + N + 1 = 2N

= O(N)

↓ ↓ ↓ ↓ ↓ ↓ ↓

a[] = −3, −2, 6, 8, 4, 8, 5

largest = ~~−∞~~ ~~3~~ ~~6~~ 8

cnt = 1 + 1 = (2)

2, 3, 8, 5, 8, (10), (10), 6

one loop

L = ~~9~~ 10

cnt = ~~1~~ 2

10

10

(10, 2)

output = 9 - 2 = (7)

Another way largest = $-\infty$

for(i=0 _____ i<=n-1){

if(a[i] > largest)

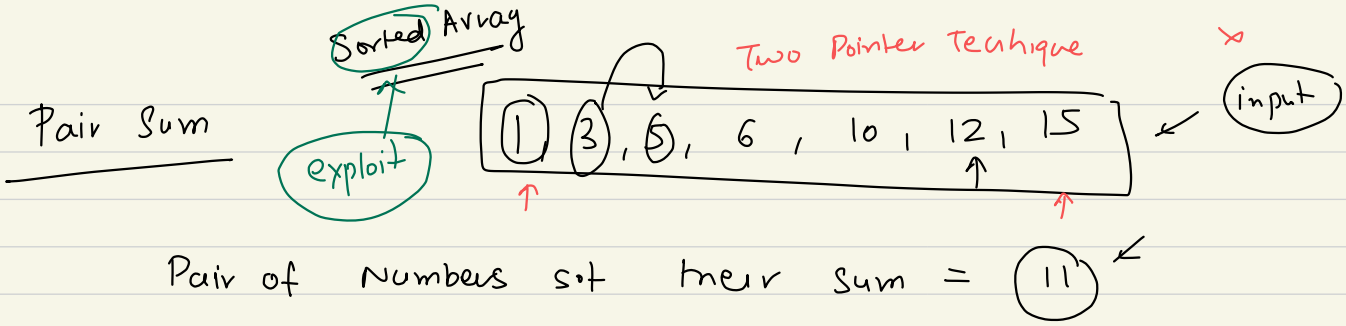
[largest = a[i]
cnt = 1]

else if(a[i] == largest)
cnt = cnt + 1

}

work $\propto$ N

$= O(N)$



All possibilities

- Two Nested loops

Algo-1

```
for (i=0 ____ N-2) {
    for (j=i+1, ____ N-1) {
        if (a(i) + a(j) == sum) {
            // print Pair
```

1+3	3+5	5+6
1+5	3+6	5+10
1+6	3+10	⋮
1+10	3+12	
1+12	3+15	
1+15		

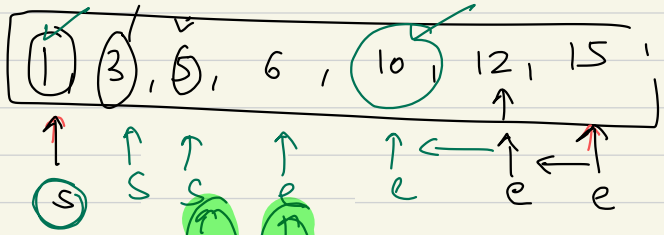
3 3 3

$O(N^2)$

[Not the best way of doing it]

Algo-2

Two Pointer Technique



Goal ⇒ 11

1 + 15 → 16 --→ 11

inc s would inc sum

dec e would dec the sum

a[s] + a[e] = 1 + 15 = 16 sum > goal e--

a[s] + a[e] = 1 + 12 = 13 sum > goal e--

a(s) + a(e) = 1 + 10 = 11 sum == goal print (a(s), a(e))

s++
e--

a(s) + a(e) = 3 + 6 = 9

a(s) + a(e) = 5 + 6 = 11 sum == goal

1, 11

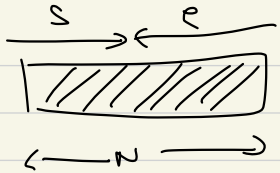
5, 6

O (N^2) ✓

↓

O (N)

```
while (s < e) {
    sum = a(s) + a(e)
    if sum > goal
        e--
    else if sum < goal
        s++
    else
        print (a(s), a(e)); s++ ; e--
}
```



Unsorted

1, 1, 3, 5, 6

Algo-3

K K K K K K K K

3, 5, 1, 2, 7, 6, 5, 8

Sorting + Two Pointer =

N log N + N

= O(N log N)

Upcoming classes

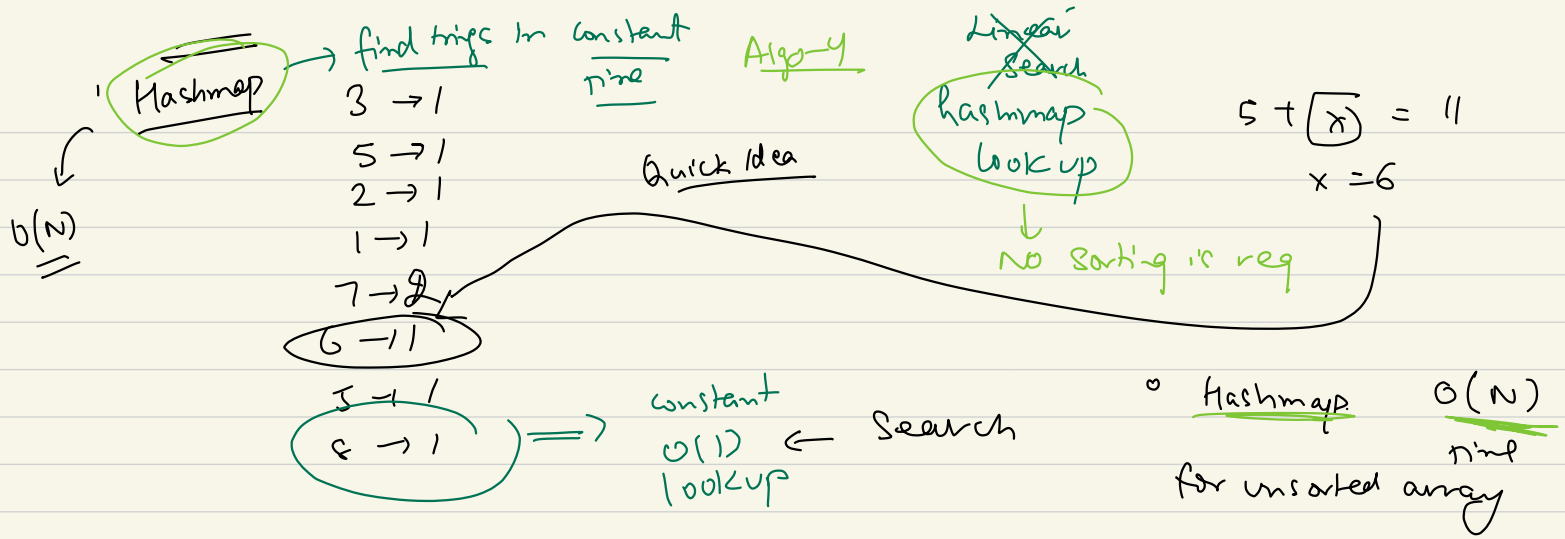
↓

O(1)

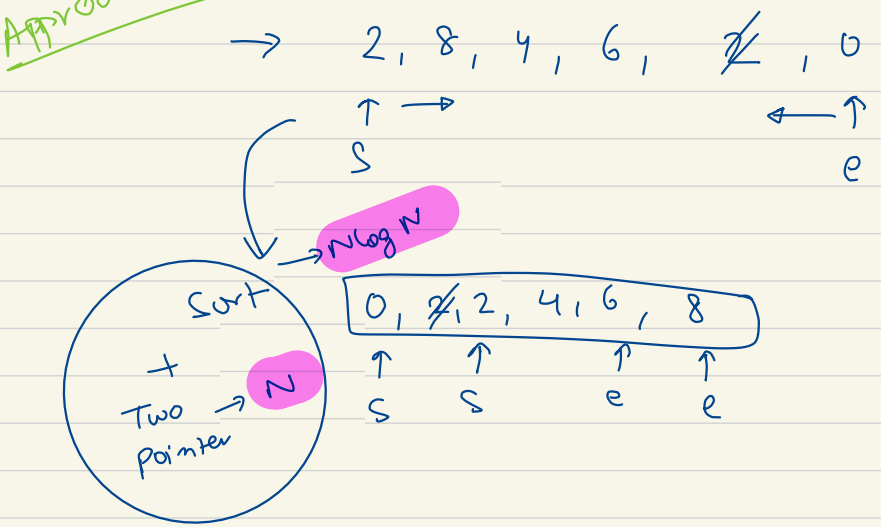
3 + X = 11

(3, 8)

↓


Hashmap
find trips in constant time
Algo-4
Linear Search
Hashmap lookup
5 + x = 11
x = 6
O(N)
Quick idea
No sorting is req
3 → 1
5 → 1
2 → 1
1 → 1
7 → 2
6 → 11
5 → 1
4 → 1
constant
O(1)
lookup
Search
Hashmap
O(N)
find
for unsorted array

Approach-3



Sum = (8)

2 + 0 = 0

s + e = (8)

2 + 6 = 8

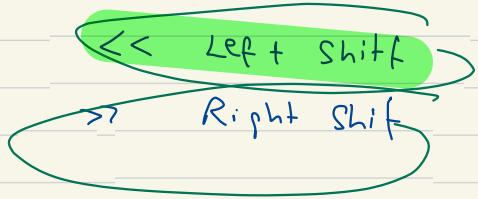
$= N \log N + N$

$= O(N \log N)$

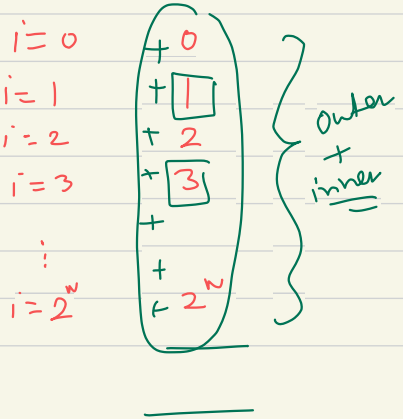
$i < 2^N$; i++

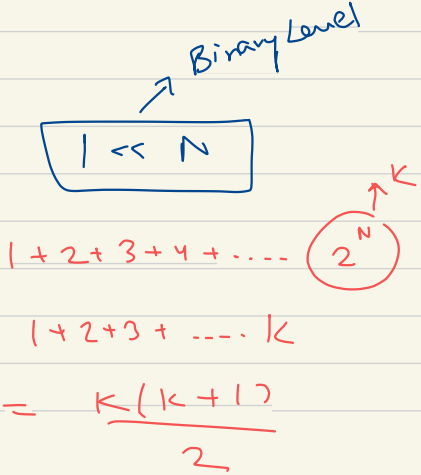
```
for( i=0;   i < (1<<N); i++){
    j=i;
    while(j>0){
        j=j-1;
    }
}
```

2^N



3



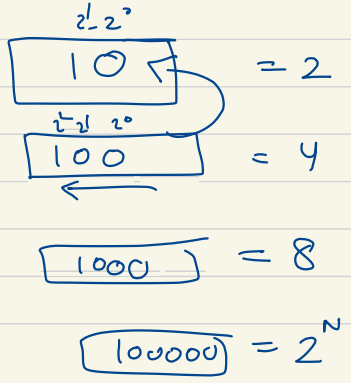


1 << 1

1 << 2

1 << 3

1 << N



$$= \frac{2^N\left(2^N + 1\right)}{2}$$

$$= \frac{2^N \cdot 2^N}{2} + \frac{2^N}{2}$$

$$= 2^{N+N}$$

$$O\left(2^{2N}\right)$$

$$= O\left(4^N\right)$$

a = 5

a<<1

5 << 2

101 << 2

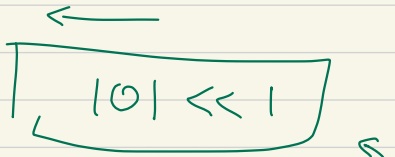
||

| 0 | 1 | 0 | 0 |

16 8 4 2 1

= 16 + 4

= 20



||

| 1 | 0 | 1 | 0 |

2^3 2^2 2^1 2^0

= 8 + 2 = 10

5 << 1 = 5 * 2

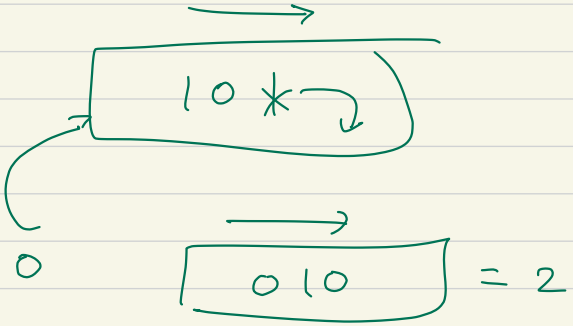
5 << 2 = $5 * 2^2$

= 20

a << b = $a * 2^b$

=> 1 << N = $1 * 2^N$

$a = 5$



$a >> 1$

$5 >> 1 = \frac{5}{2} = 2$

$a >> b = \frac{a}{2^b}$

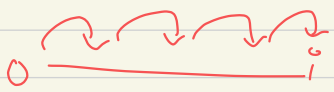
$1 >> N = \frac{1}{2^N}$

```
for ( i = N ;  i > 0 ;  i = i / 2 ) {

    for ( j = 0 ;  j < i ;  j = j + 2 ) {
        // work
    }

}
```

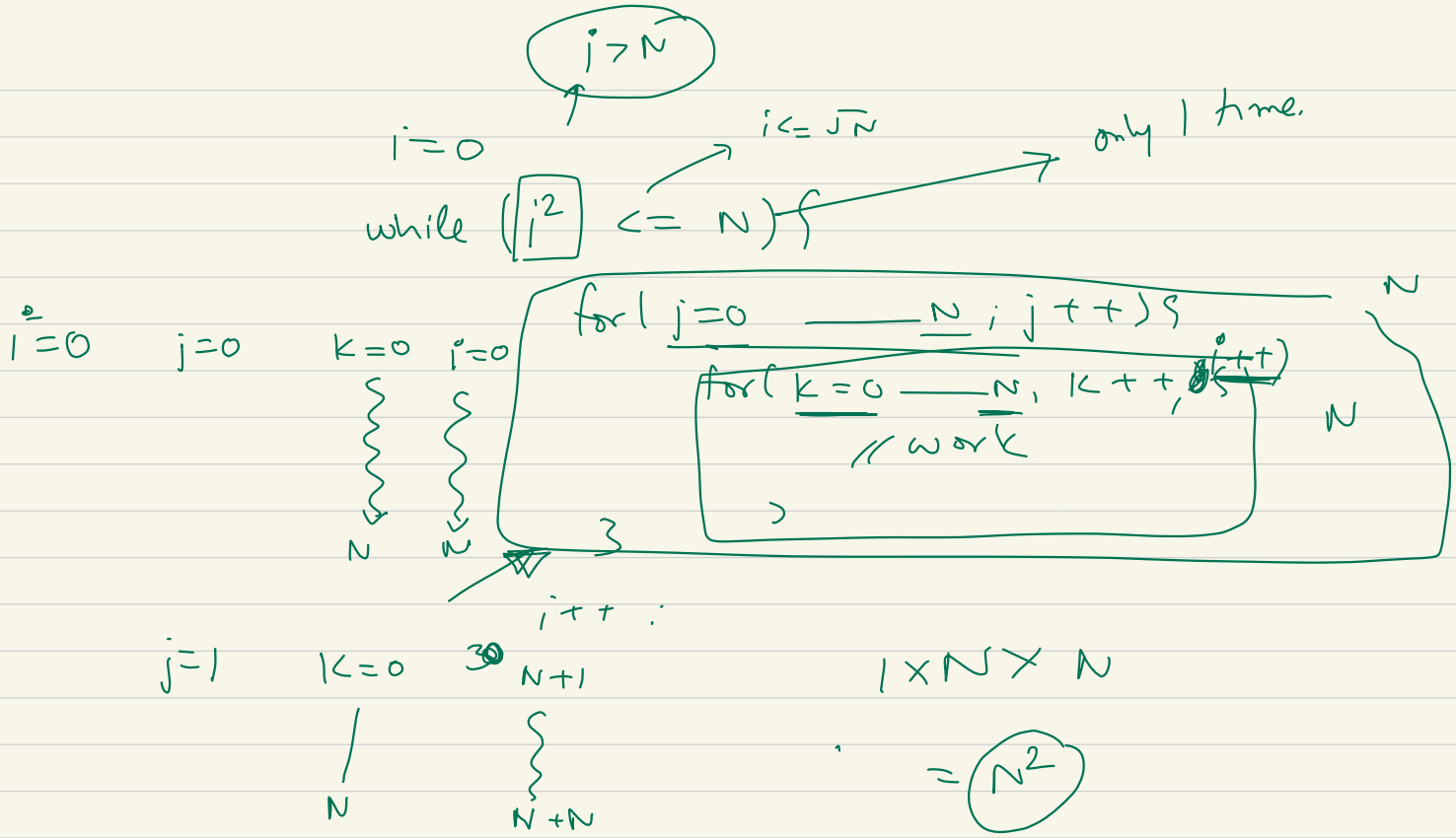
$i = N$	$N/2$
$i = N/2$	$N/4$
$i = N/4$	$N/8$
$\vdots$	$\vdots$
$i = 1$	0

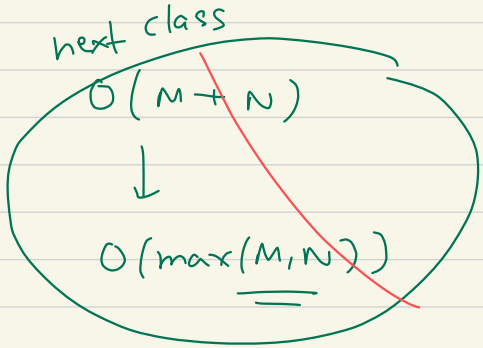


$$\frac{N}{2} + \frac{N}{4} + \frac{N}{8} + \cdots \quad 1$$

$$= N\left(\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \cdots\right)$$

$$= N\left(\frac{a}{1-r}\right) = N\left(\frac{1}{1-1/2}\right) = O(N)$$


i > N
i = 0
i <= √N
only 1 time.
while (i^2 <= N) {
i = 0
j = 0
k = 0
i = 0
for (j = 0 ____ N ; j ++) {
for (k = 0 ____ N ; k ++) {
// work
}
N
N
N
N
i ++ ;
j = 1
k = 0
30
N+1
N
N + N
1 x N x N
= N^2



for (———— M)

for(———— N)

```
fun (int n)
for (i=1; i*i<=n; i++) {
    for (j=1; j*j<=i; j=j+i) {
        // work
    }
}
```

1 ---- $\sqrt{N}$

$\sqrt{N}$ times

$\sqrt{N}$

once

once

3 3

2nd iteration

$j + i > \sqrt{i}$

i = 1
i = 2
:
$i = \sqrt{N}$

$O(\sqrt{N})$

j