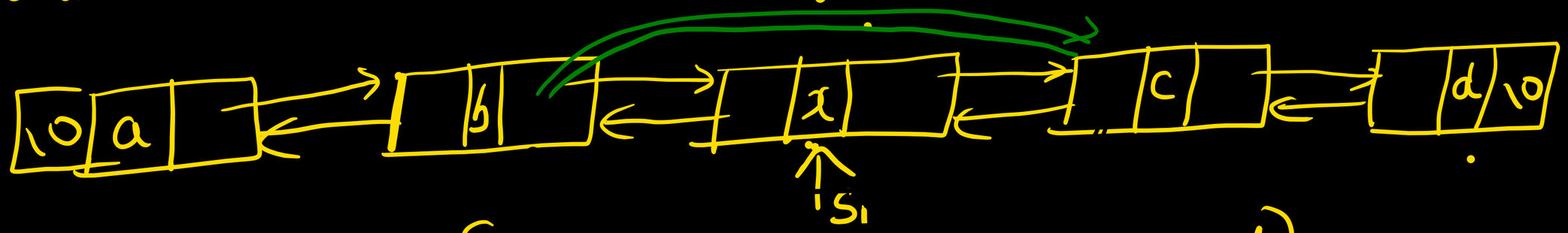
→ Delete a node with data x from a DLL



③ Cases:

1) x is I node ✓
2) x in last node ✓
3) x in middle node in between ✓

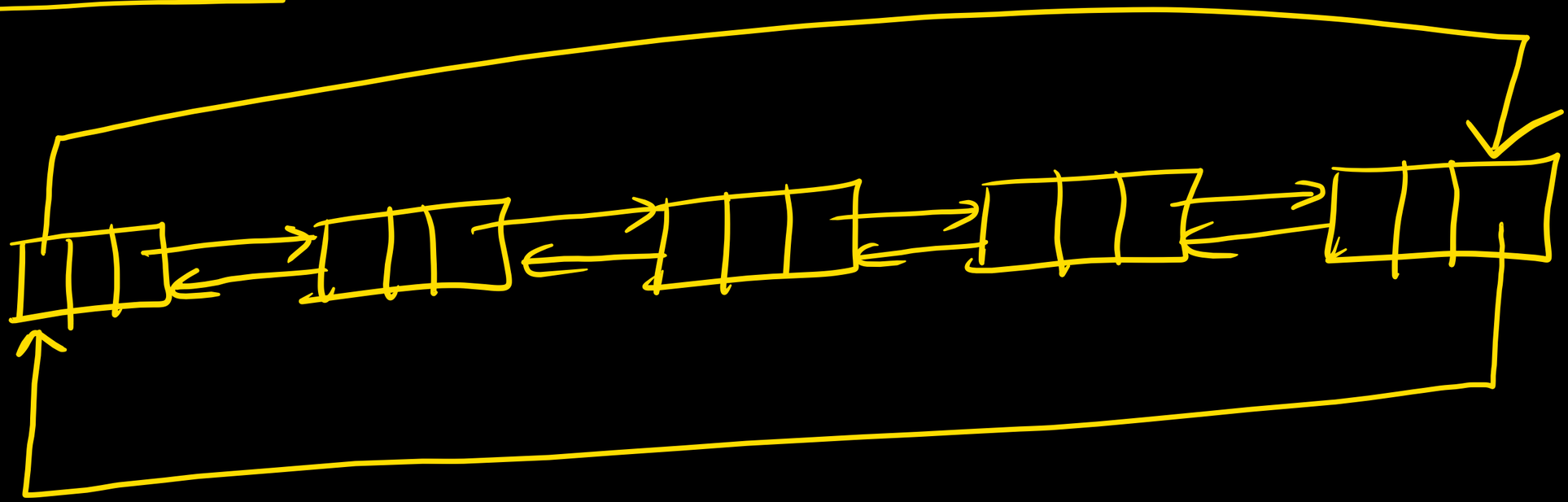
```
S1 = S
while( S1 -> data != x && S1 -> next != null)
{
    S1 = S1 -> next
}
if ( S1 -> data == x)
```

S1 -> prev -> next = S1 -> next

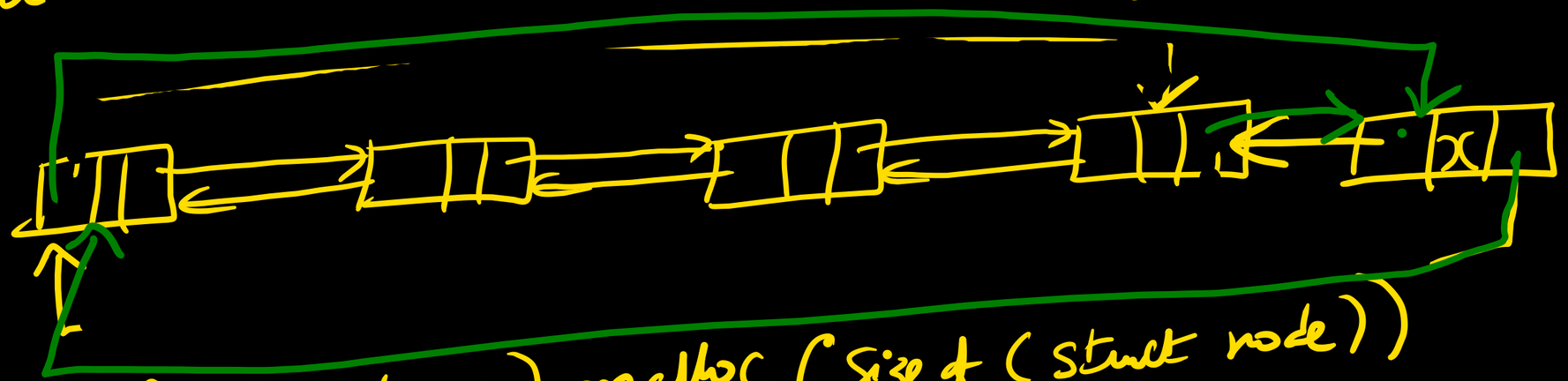
previous node should point next node exists

modify the code.

Circular DLL



add a node with data 'x' at the end of CDLL



p = (struct node *) malloc (sizeof (struct node))

p → data = x

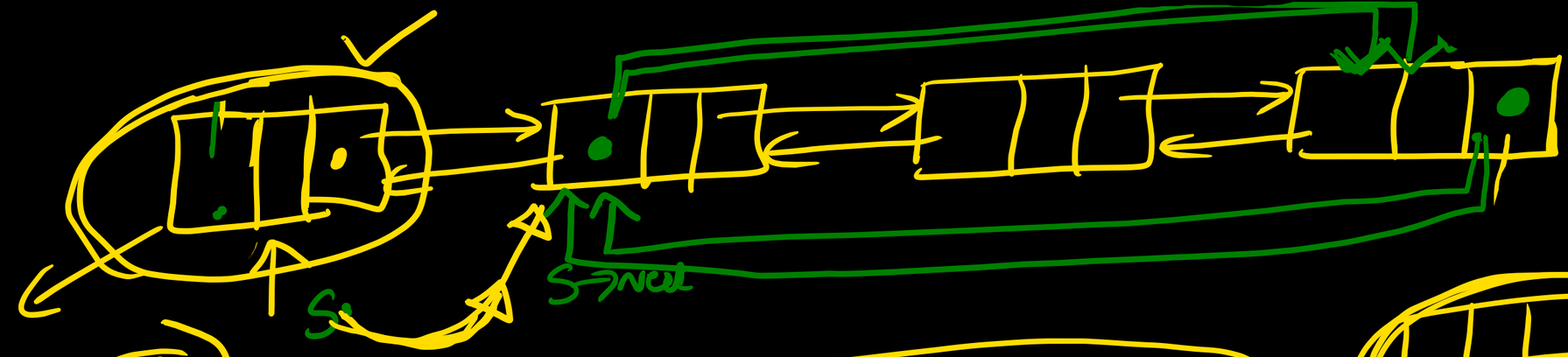
p → next = S

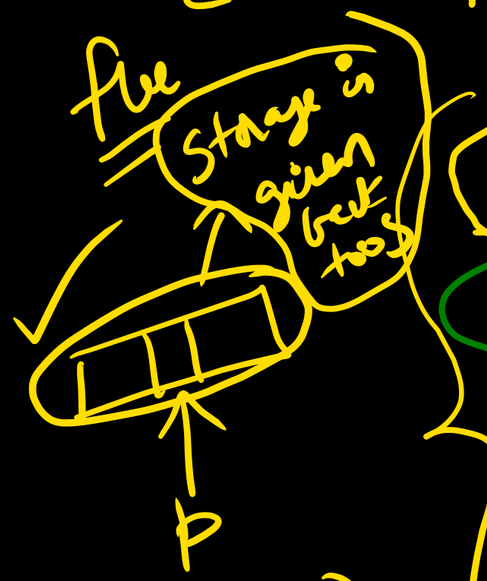
p → prev = S → prev.

S → prev = p.

p → prev → next = p.

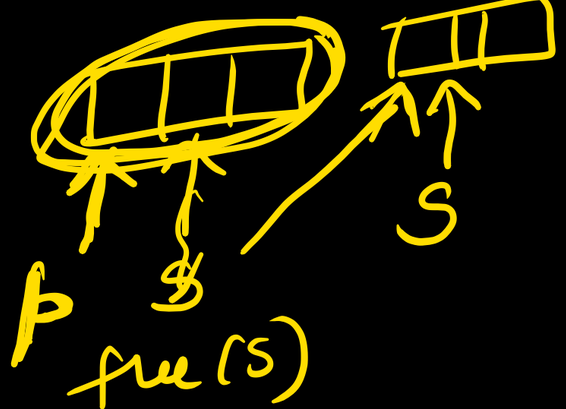
Delete I node in C-DLL



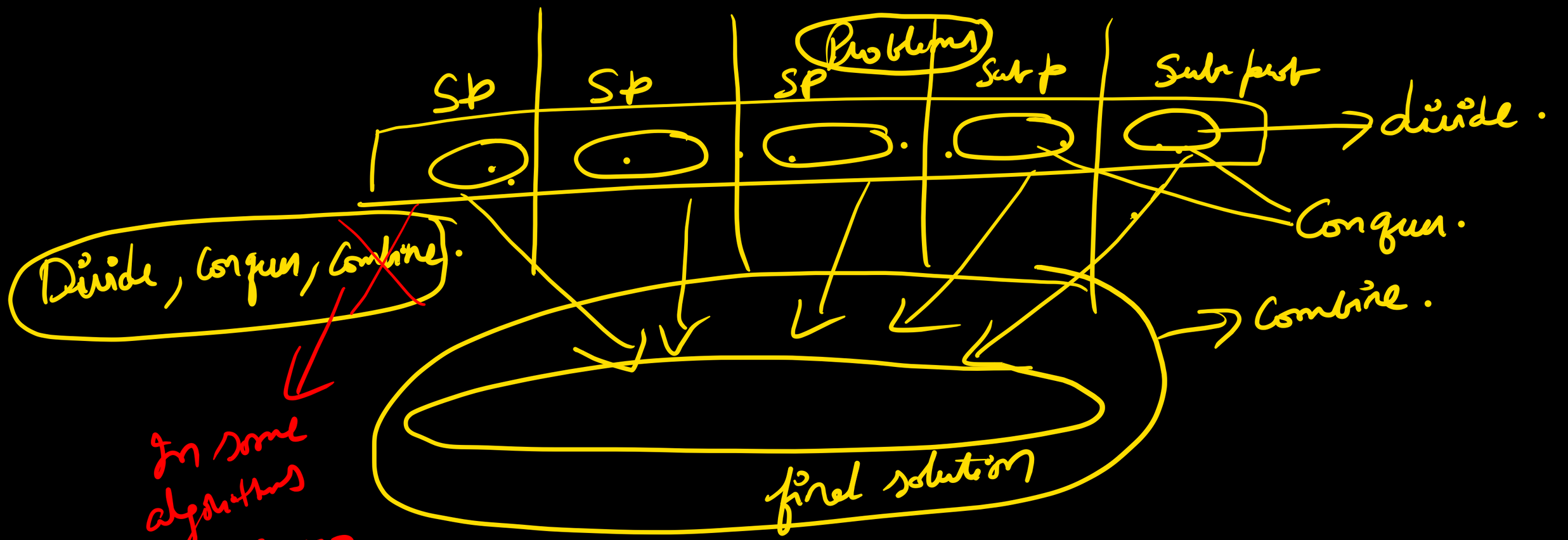



```
S->next->prev = S->prev
S->prev->next = S->next
node *p = S;
S = S->next;
free(p);
p = NULL;
```

dangling pointer.

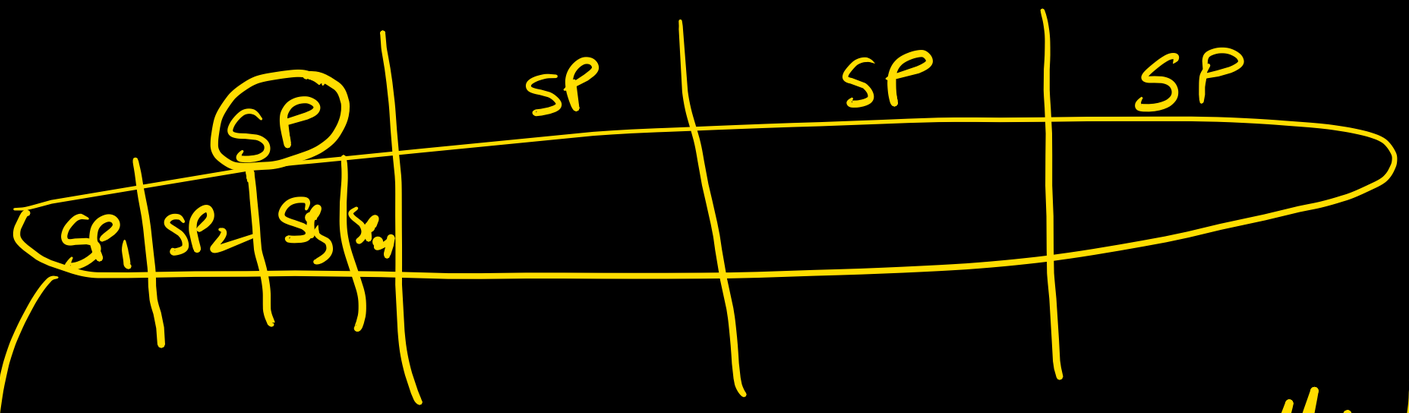


Divide and Conquer :-



Divide and Conquer:-

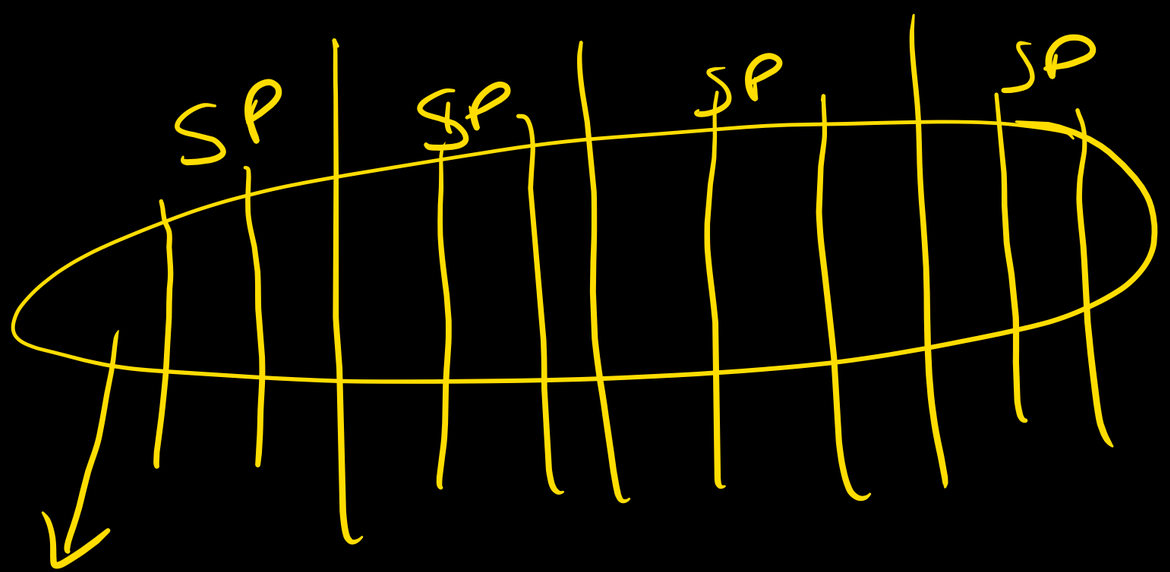
Given a function to compute on n inputs, the divide and conquer suggests splitting the i/p into k distinct ~~Sodj~~ Sub problems, $1 < k \leq n$, yielding k Sub problems. These Sub problems must be solved and a method must be found to combine Sub solutions into a solution of the whole. If the subdems are still relatively large, then divide and conquer strategy can be applied again



dividing Sub problems into even smaller problems
Can be done using recursion.

∴ D and C uses recursion

Division of a problem continues until it becomes the problem is small enough that can solved without splitting.



Small bubbles can be solved without diving further

Algorithm DAndC (P)

{ if Small (P) then return S(P); ← Solution

else

{ divide 'P' into smaller instances

$P_1, P_2, P_3, \ldots, P_K$ $K \geq 1$

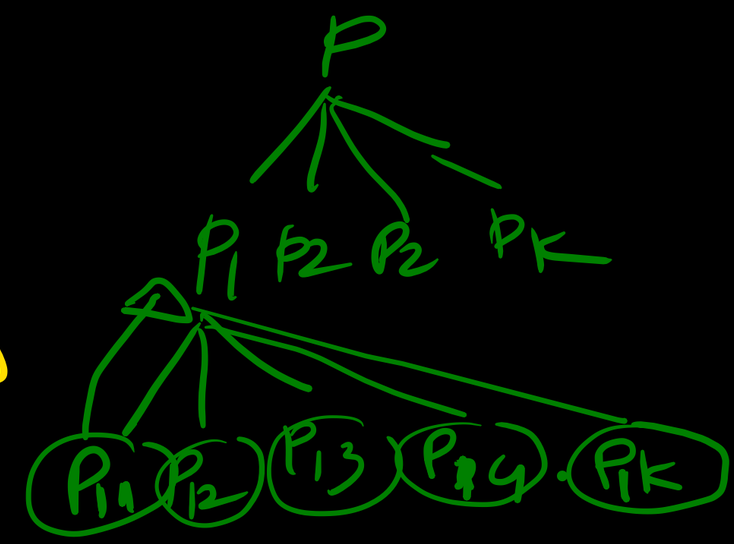
Apply DandC to each of these subproblems ← Recursion

return Combine (DandC(P_1), DandC(P_2)

.... DandC(P_K));

}

}



TC for D and C in recurrence relation

$$T(n) = \begin{cases} g(n) & \text{if } n \text{ is small} \\ T(n_1) + T(n_2) + \cdots + T(n_k) + f(n) & \text{otherwise} \end{cases}$$

$g(n)$ is time for small problems

$f(n) \rightarrow$ is the time for
dividing and Combining
Solutions to subproblems

99%

99% of DanC problems

$$T(n) = \begin{cases} T(1) & n = 1 \\ a\,T(n/b) + f(n) & n > 1 \end{cases}$$

$T(1)$ → Some Constant

where a and b are Constants

and b n is a power of b

i.e $n = b^k$

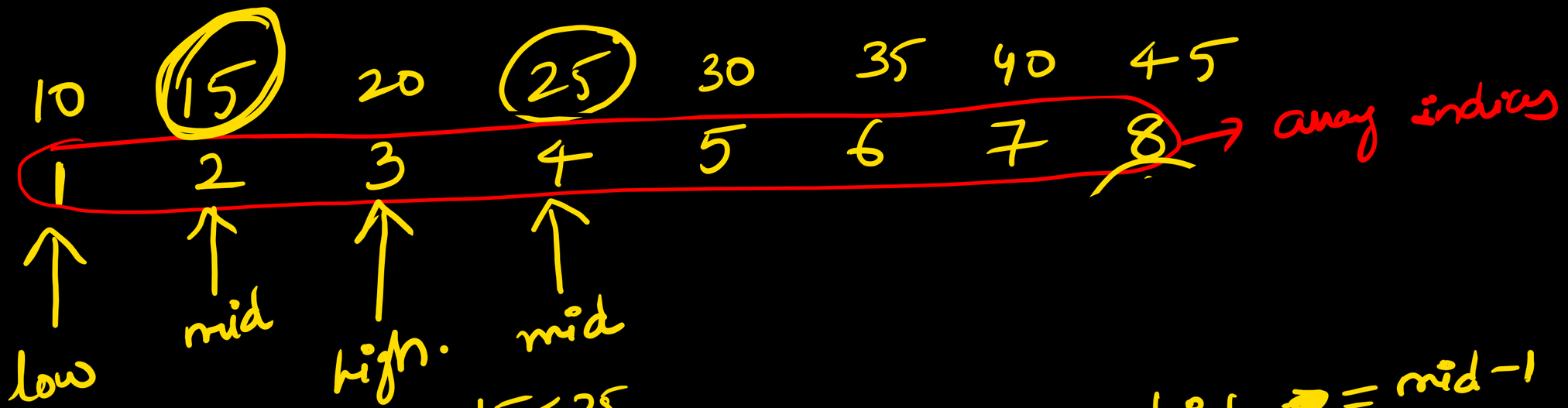
ms: $T(n) = 2T(n/2) + O(n)$ ✓

Subs Pec tum T.

BS $T(n) = T(n/2) + C$

qs $T(n) = 2T(n/2) + O(n)$ (avg case).

Searching for $X=15$



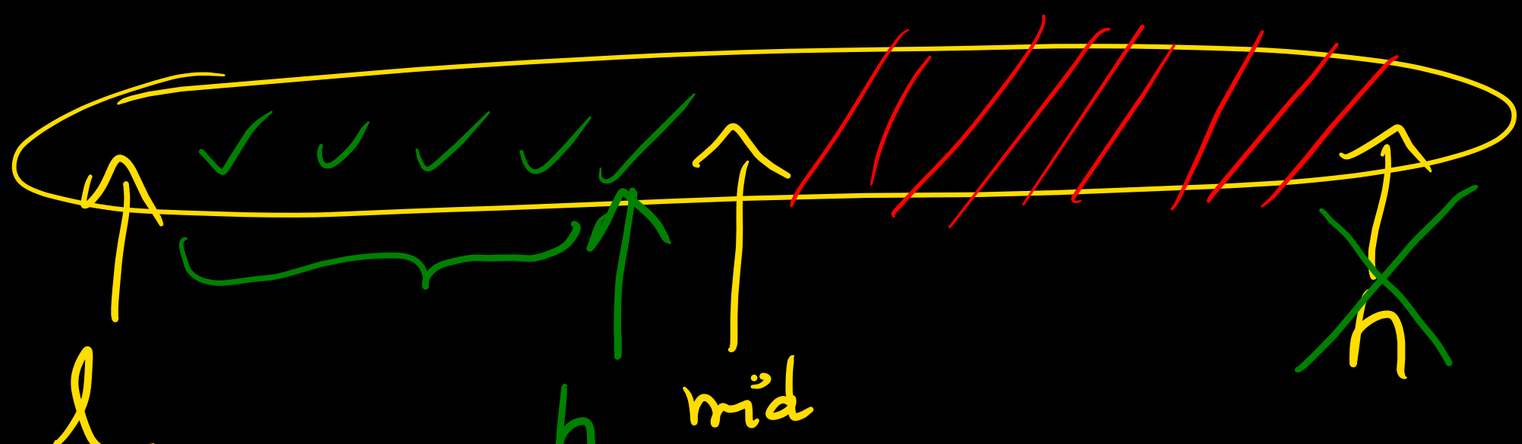
$15 < 25$

high = mid − 1

Binary search only works on sorted array.

$$mid = \frac{low + high}{2} = \left\lfloor \frac{1+8}{2} \right\rfloor = 4$$

$$mid = \left\lfloor \frac{low + hi}{2} \right\rfloor = \left\lfloor \frac{1+3}{2} \right\rfloor = 2$$



l to h in the search space.


x.
sc.
high
low
mid.
low
= mid+1

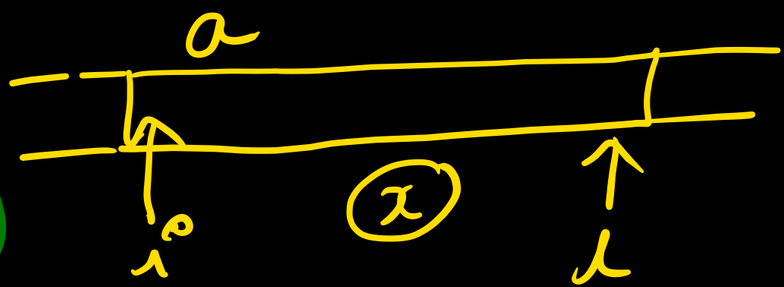
Algo BinSrch (a, i, l, x)

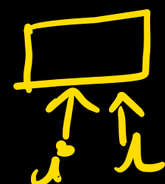
⇒ only one element

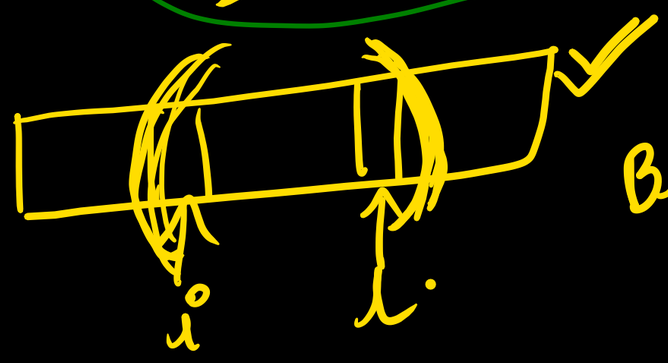
T(n)

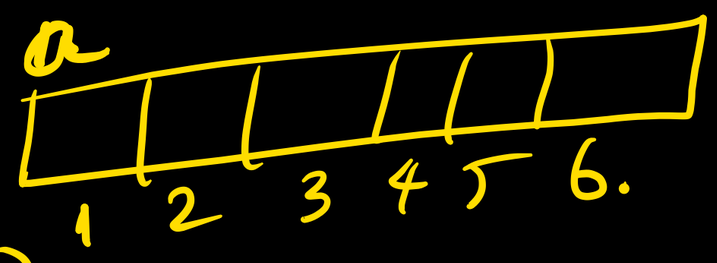
```
{ if (l == i) then
  { if (x == a[i]) then return i;
    else return 0;
  }
}
```

⇒ Small problem

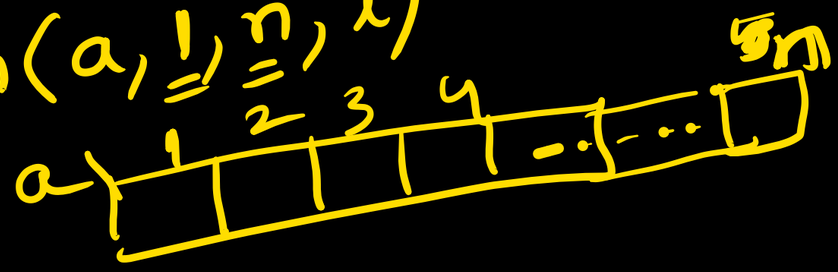








BinSrch(a, 1, n, x)



else
{

mid = ⌊(i+l)/2⌋ ; → dividing the problem

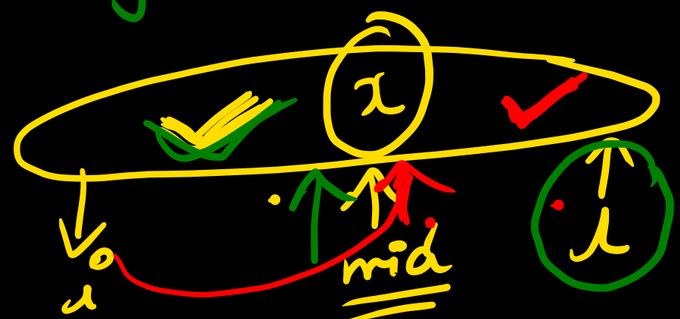
if x = a[mid] then return mid;

else if (x < a[mid]) then

return BinSrch(a, i, mid-1, x)

else return BinSrch(a, mid+1, l, x)

}



BS(a, i, l, x)

↙ T(n/2) mid+1 ↓ mid-1

T(n/2)

$$T(n) = \begin{cases} c & ; \quad n == 1 \end{cases}$$

BS (a, i, l, x) ⇒ T(n)

{

if BS (a, mid+1, l, x) ⇒ T(n/2)

BS (a, i, mid-1, x) ⇒ T(n/2)

T(n) = T(n/2) + C.

$$T(n) = \begin{cases} a & \text{when } n == 1 \\ \text{Comparing if } a[i] == x & \\ T(n/2) + c & \text{when } n > 1 \end{cases}$$

$T(n/2)$ + c — dividing the problem

master theorem

$= O(\log n)$ — $\log n$.

$f(n)$: c

$n^{\log_b a}$: $n^{\log_2 1} = 1$

~~T(n/2)~~ T(n) — c

|

T(n/2) — c

|

T(n/4) — c

⋮

T(n/2^k)

~~T=0/2~~

k = ?

levels = depth.

SC = O(n)

n/2^k = 1

⇒ k = log n

TC for BS for all cases is $O(\log n)$

SC for recursive BS is $O(\log n)$

BS(n)
{
if small ... ...
mid = (i+j/2)
BS(n/2)
BS(n/2)
}
small ✓
dividing
Conquering
X Combine.

BS(n)
{
if small ... ...
mid = (l+h/2)
BS(n/2)
BS(n/2)
}
small
dividing
Conquering
NOT Combine.

D and C → recursion.

iterative BS algo.

// → Comments

Algo Binsearch (a, n, x)

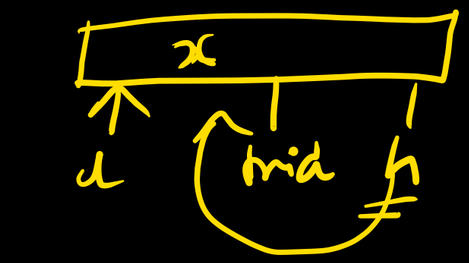
// Given an array a[1:n] of elements, search for x

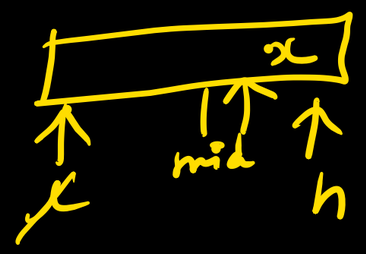
✓{ low = 1 ; high = n ;

while (low ≤ high) do

✓{ mid = $\lfloor \frac{low+high}{2} \rfloor$

```
if (x < a[mid]) then
        high = mid - 1
else if ( x > a[mid]
        low = mid + 1
else return mid;
}
return 0
}
```





demo

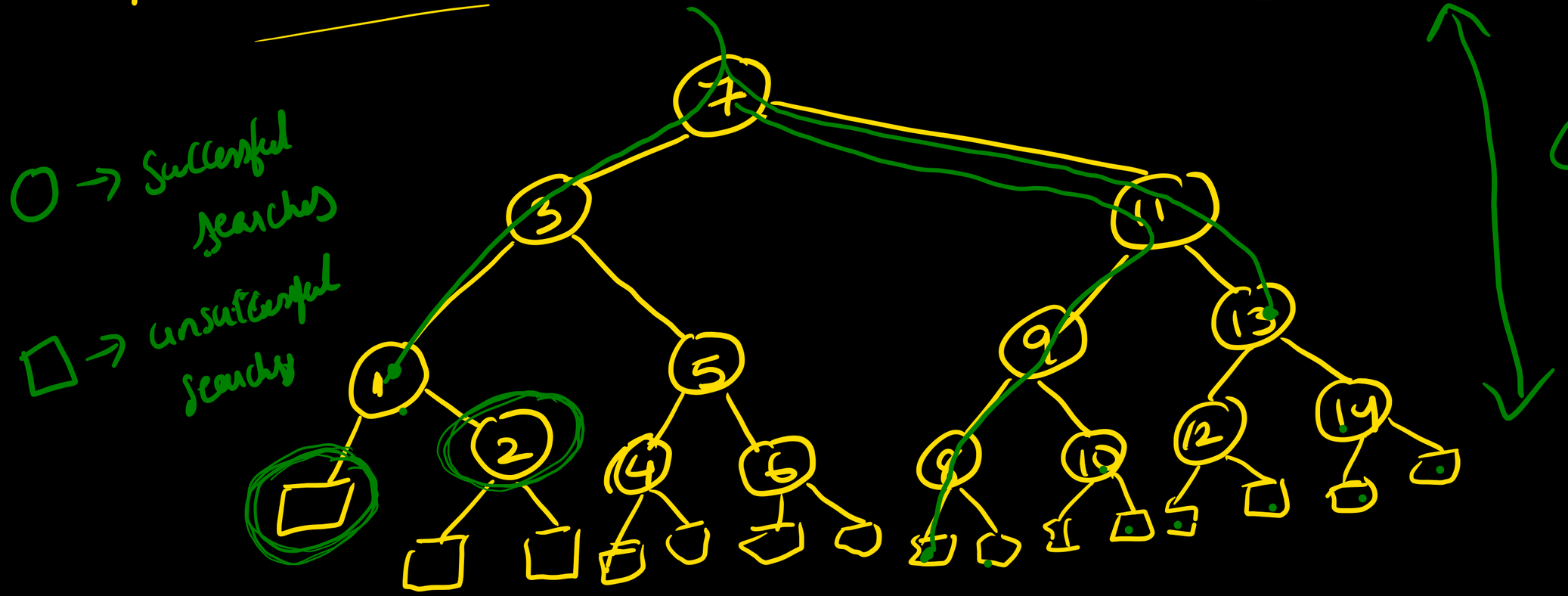
Iterative:

$TC = O(\log(n))$

$SC = O(1)$

iterative is better than recursive.

1 2 3 4 5 6 (7) 8 9 10 11 12 13 14

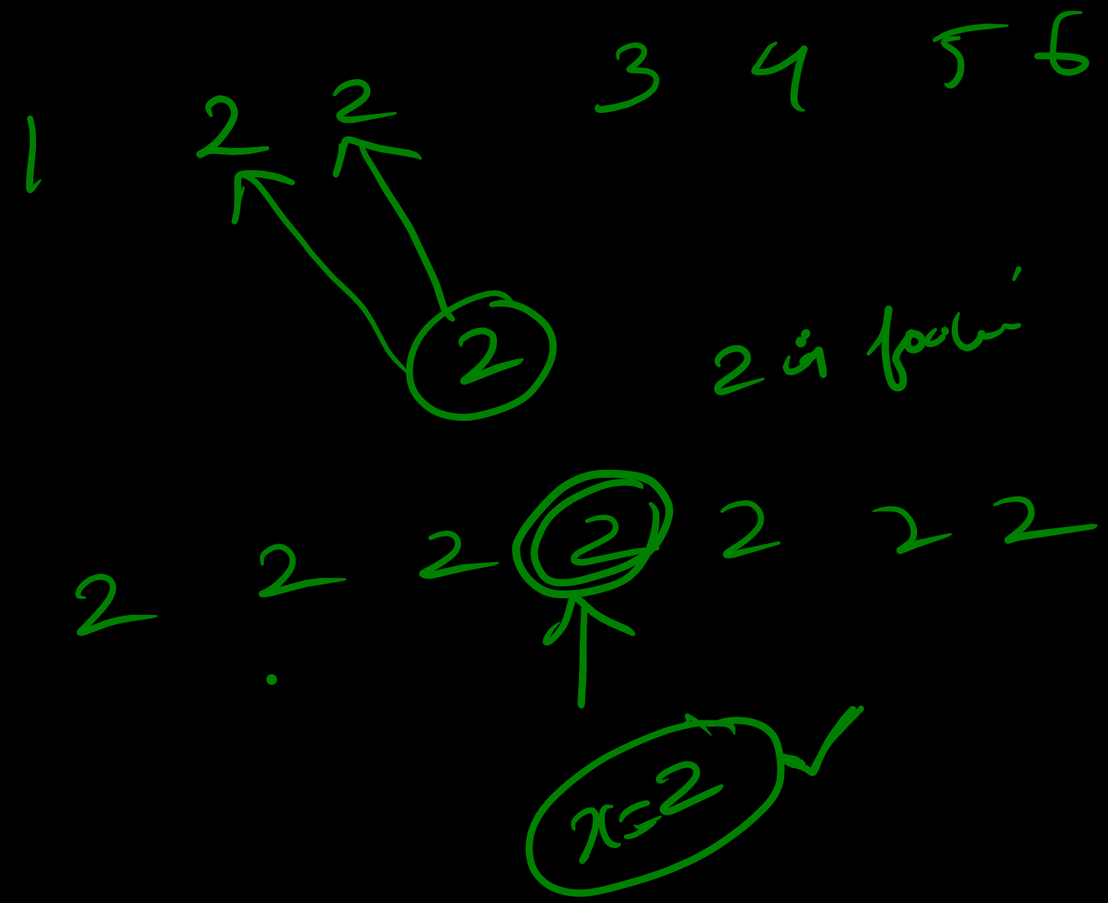


O → Successful searches

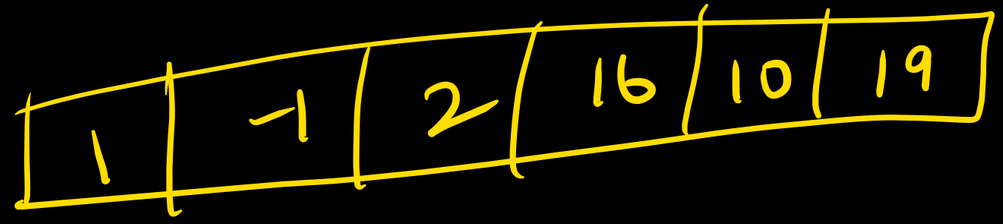
□ → Unsuccessful searches

O(log n).

Binary decision tree for binary search


1 2 2 3 4 5 6
2
2 is found
2 2 2 2 2 2 2
x=2

Another application of D and C in min max:-



min = -1

max = 19

min max problem.

Algorithm straightmaxmin(a, n, max, min)

{ max = min = a[1];

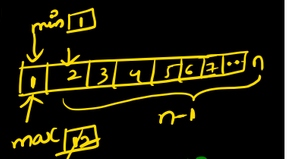
for (i = 2 to n do) (n-1) ✓

if a[i] > max → 1 comp
if a[i] < max → 2 comp.

{ if (a[i] > max) then max = a[i];
else if (a[i] < min) then min = a[i];
}

}

no of comparisons: 2(n-1)



max=10
min=1
a[i]=100 > max ✓
≠ < min ✓

Best case i/p: always a[i] > max; next element should be greater than max computed so far

BC: (n-1) WC: 2(n-1) ✓

on average, a[i] is greater than max half of the time

n/2 - 1/2 + n - 2