DAY 57

Time And Space Complexity continued..

Date :- 9 November, 2021
Day :- Tuesday
Agenda :- for * for loop = O(n)
Bulelele Sort = $O(n^2)$
Selection Sort = $O(n^2)$
fibonacci series = $O(2^n)$

There is a for loop inside another for loop But still the Time Complexity is O(n) Why?

for * for but still O(n)

```
for (int i=1; i<=n; i+=c) {
    for (int j=1; j<=c; j++) {
        // do some K work.
    }
}
```

for for ↓ but not $O(n^2)$

यें loop चलेगा n times क्योंकि i की value c से बढ़ती जाती है n/c times चलेगा

अंदर वाला loop c times चलेगा, क्योंकि j की value 1 से increment हो रही है

अब हर बार i की value जब भी बढ़ेगी तो i की value से हम c से increment करेंगे और दो के पास चले गयें कि ऊपर/बाहर वाला loop n/c times चलेगा

* बाहर वाला loop चलेगा $\frac{n}{c}$ times.
* अंदर वाला loop चलेगा c times for each entry in for loop of variable i
* तो Total अंदर वाला loop कितनी बार चलेगा?

$$\frac{n}{c} \times c = n$$

* तो अगर हर बार अंदर वाला loop चलने में K time लगता है, तो Total Time = Tk

(*) तो Total Time consumed in this for * for loop is nk.

eg → मान लो $n = 1000$
$c = 50$

अगर c 50 है तो i की value increment होगी c से हर बार

तो i = 1 +50
i = 51 +50
i = 101 +50
i = 151 +50
i = 201 +50
i = 251 +50
i = 301 +50
i = 351 +50
i = 401 +50
i = 451 +50
i = 501 +50
i = 551 +50
i = 601 +50
i = 651 +50
i = 701 +50
i = 751 +50
i = 801 +50
i = 851 +50
i = 901 +50
i = 951 +50
i = 1001

20 times i की value increment हुई

Total no. of times Inner loop works = Total no. of times outer loop work × (No. of iteration of inner loop for each i value)

$\frac{n}{c} = \frac{1000}{50} = 20$

∴ We can say that बाहर वाला loop n times चलेगा।

MULTIPLY

for each value of i, the inner loop runs 50 times

j=1 j=2 j=3 j=4 j=5 ——— j=50

तो 20 × 50 = 1000 = n ∴ n times loop चलेगा।

∴ We can say that

अगर outer loop चलेगा n/c times
और inner loop चलेगा c times
तो inner loop overall $\frac{n}{c} \times c$ times चलेगा
और हर बार inner loop को चलने में अगर k time लगता है
तो Total time कितना लगेगा overall for * for loop
को चलने में → $\frac{n}{c} \times c \times k$

$n \quad k$

→ nk

∴ Now, we can say that $T(n) \rightarrow Kn$ (K → constant, n → size)

$T(n)$ → Time taken for Input of Size n

∴ Time Complexity = $O(n)$

Big Oh Notation

Bubble Sort →

Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in wrong order.

Bulelele Sort

↳ अगर हमे कभी ये बोला जाए कि Bubbleing करदो तो इसका मतलब होगा कि Bubblele Sort वाली Algorithm लगादो।

Eg-1) | 20 | 90 | 30 | 40 | 10 | → Array

Journey 1 → 20 90 30 40 10

C1
↳ comparision 1.

Bulelele Sort algorithm में हम पिछले element को next element से compare करते हैं और हर iteration में यही process repeat होता रहता है।

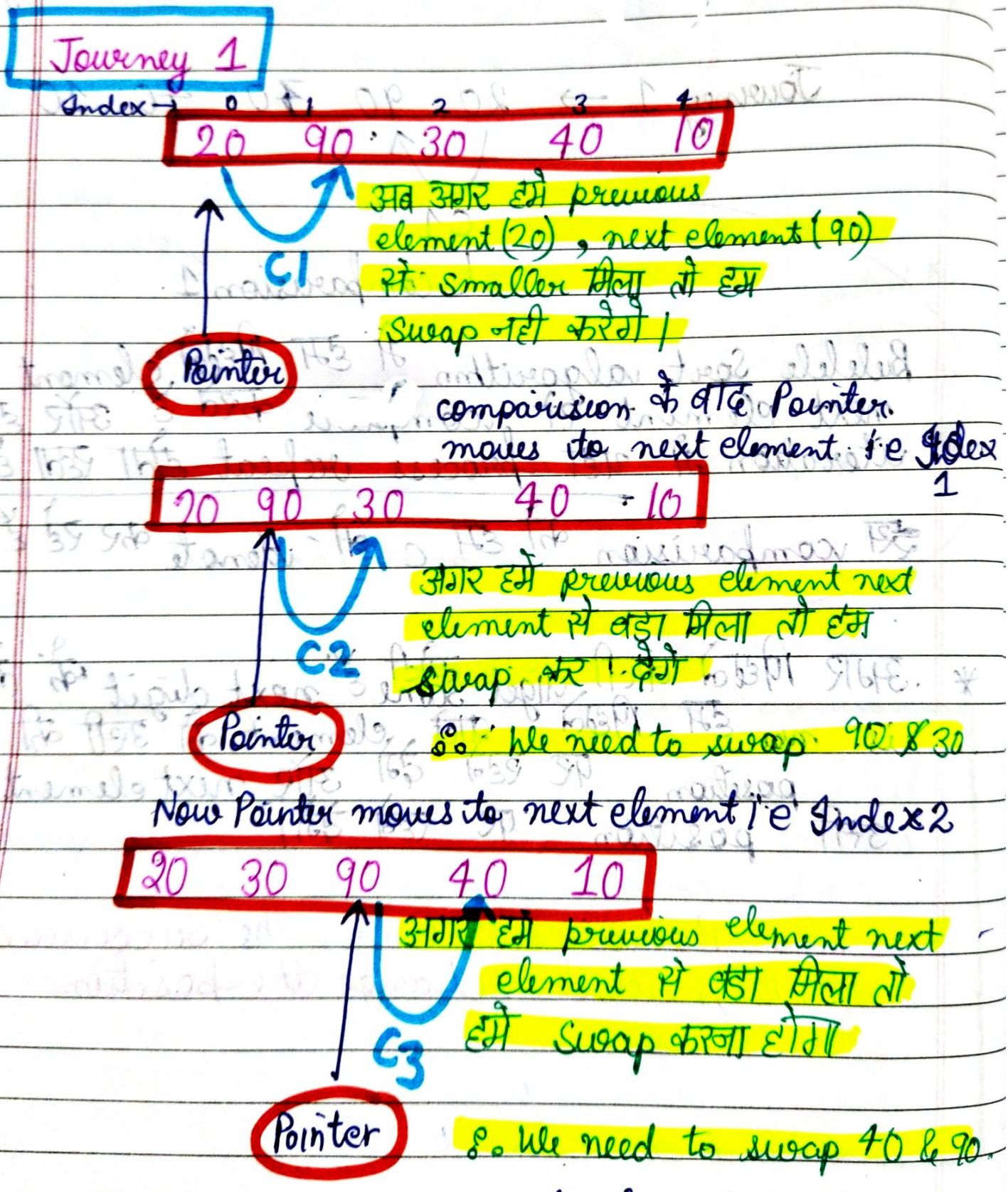
इस comparision को हम C से denote कर रहे हैं यहाँ।

* अगर पिछले वाली digit छोटी है next digit से तो हम पिछले वाले element को उसी की position पर रहने देंगे और next element को भी उसी position पर रहने देंगे।

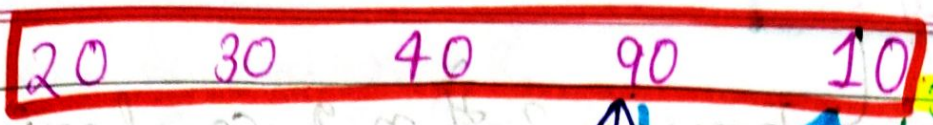
When Previous number < The Successive Number
No need to change the positions.

* लेकिन अगर next number is less than the previous number, then हमें previous number और next number को swap करना होगा क्योंकि हम चाहते कि largest element end में आए array में।

Journey 1



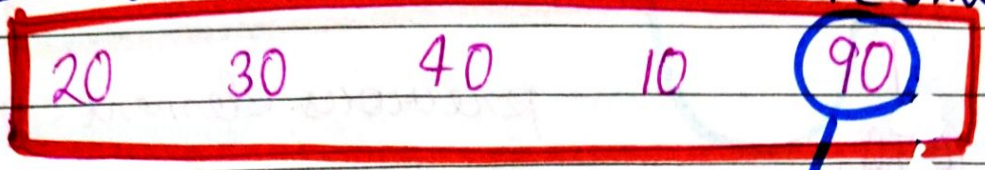
Now Pointer moves to next element i.e Index 3



अगर हमें previous element next element से बड़ा मिला then swap

Pointer

Now Pointer moves to next element i.e Index 4



This is the output of Journey 1

UnSorted

Sorted

Now, when the Pointer reaches the (n-1)(last)Index, here the Journey 1 ends.

तो हमें पता चला कि n-1 comparision होंगे in Journey 1.

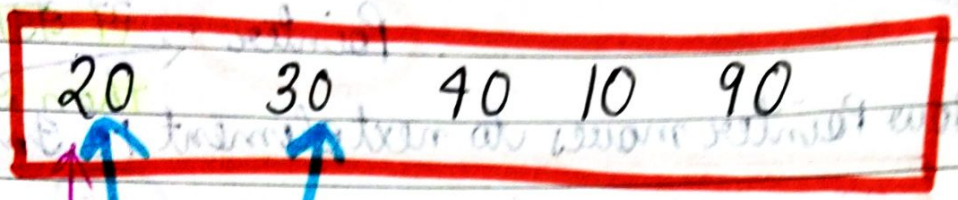
सबसे बड़ा element सबसे end में पहुँच जाएगा after Journey 1.

So, we have 2 conclusion after Journey 1

1. → Journey 1 में (n-1) comparisions होंगे।
2. → Journey 1 खत्म होने के बाद हम last index पर सबसे बड़ा element देख पाएँगे।
3. → अब बस starting के n-1 elements sorted नहीं हैं, last element sorted है।

Journey 2.

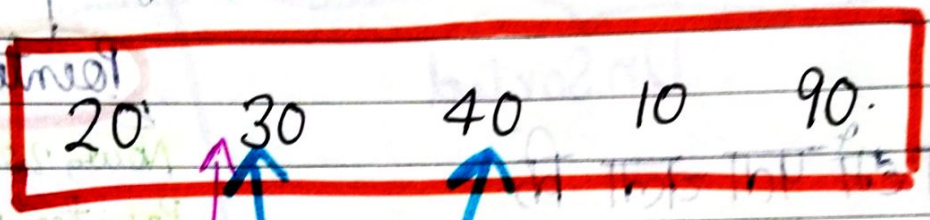
→ Journey 2 में n-2 comparision होंगे और (n-2)th index तक Pointer जाएगा.



Pointer

C1

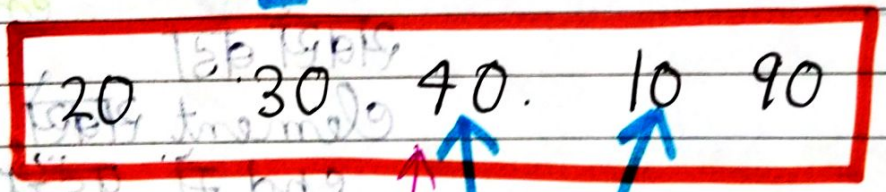
previous element < next element
∴ no swap



Pointer

C2

previous element < next element
∴ No swap.



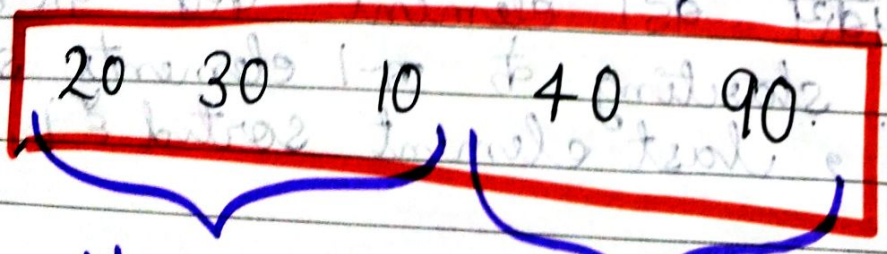
Pointer

C3

previous element > next element

Pointer has reach n-2th index.
∴ We need to stop Journey 2 here

∴ Swap 40 & 10



UnSorted

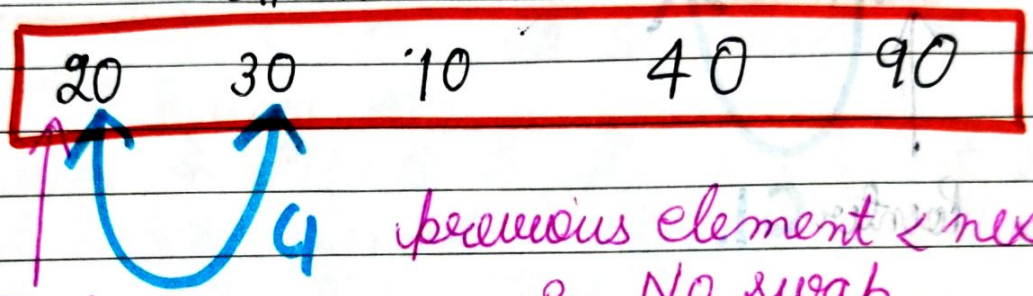
Sorted

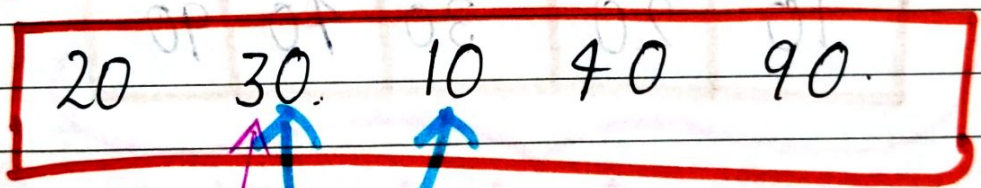
Conclusions of Journey 2

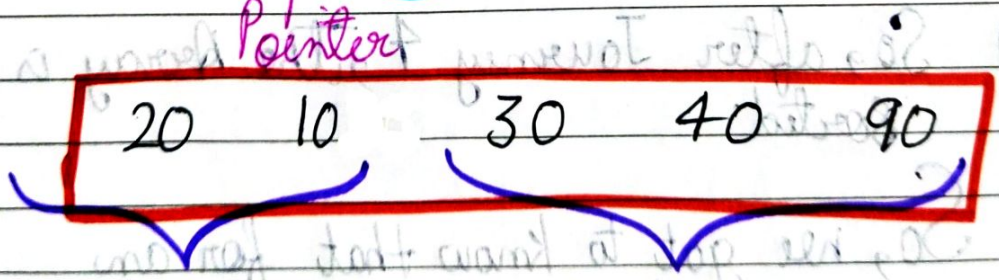
→ Journey 2 में n-2 comparision होंगे
→ Journey 2 खत्म होने के बाद हम देख पाएंगे कि second Largest element (n-2)th index पर है।
→ अब बस starting के n-2 elements sorted नहीं हैं, last के 2 elements sorted हैं।

Journey 3

→ Journey 3 में n-3 comparisions होंगे। और (n-3)th index तक ही Pointer जाएगा।







→ Conclusion of Journey 3
→ Journey 3 में n-3 comparision होंगे
→ Journey 3 खत्म होने पर हम देख पाएँगे कि Third Largest element (n-3)th index पर है।
→ अब बस starting के n-3 elements - हमें sort करने हैं, last के 3 elements तो sorted ही हैं।

Journey 4

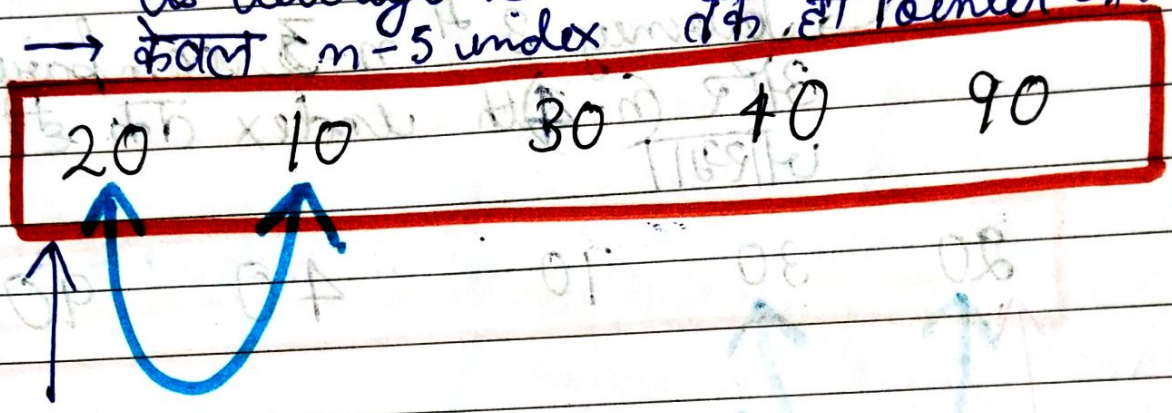
Conclusion of Journey 4

→ Journey 4 में n-4 comparision होंगी

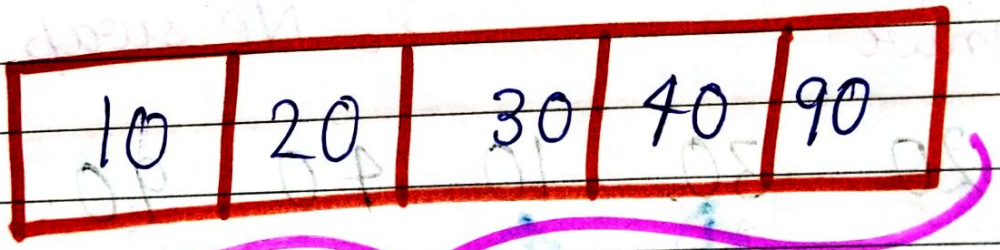
→ Journey 4 खत्म होने के बाद हम देखपाएँगे कि forth largest element is at its correct position i.e n-4

→ हमें अब दस starting से n-4 (i.e 1) element sort करना है & 1 element is always sorted

→ केवल n-5 index तक ही Pointer जाएगा



Pointer C1.



Sorted

→ So, after Journey 4, the Array is sorted.

So, we got to know that for an array of size n, we need to do n-1 Journey

Pointer goes till index n-1	JOURNEY 1	COMPARISION = 4
Pointer goes till index n-2	JOURNEY 2	COMPARISION = 3
Pointer goes till index n-3	JOURNEY 3	COMPARISION = 2
Pointer goes till index n-4	JOURNEY 4	COMPARISION = 1

हम ये भी कह सकते हैं कि 1 Journey के बाद एक element sort हो जाएगा। तो अगर हमारे पास 5 elements थे array में हमने यहाँ 4 Journey ली।
4 Journey के बाद 5 में से 4 element sort हो गई तो 5th element तो खुद ही अपनी correct position पर पहुँच जाएगा।

Array Elements = 5.
Journey = 4
Elements to be sorted = 4. (अगर 4 elements sort हो गए तो 5th element automatically sort हो जाएगा।)

Bubble Sort <code>

```
import java.util.*;
import java.io.*;
public class Main{
public static void bubblesort(int[]arr)
{
for(int itr=1; itr<=arr.length-1; itr++)
{
for(int j=0; j<=arr.length-1-itr; j++)
{
if(isSmaller(arr, j+1, j)==true)
```

इसके पास argument में array भेजेंगे

itr loop चलेगा ई Journey के लिए

Journey 1 से n-1 तक होती है (i.e. 5 elements 4 journey और n elements के लिए)

ई Journey में n-1-itr comparision होंगे

eg→ element = 5
तो first Journey में (n-1-itr) 5-1-1=3 (0-3) i.e (4) comparision होंगे

n-1-itr 0 से 3.

अगर true return करेगा तो swap function call होगा

अगर arr[j+1], arr[j] से बड़ा मिला तो उन्हें swap करेंगे

swap(arr, j+1, j);

जिस order में compare किया उसी order में swap करदो

अगर arr[j+1] बड़ा मिला arr[j] से तो swap करदो

}

}

}

// हमने bubble sort function तो लिख दिया है।
// अब हमें swap function लिखना है, इस function से हम arr[j+1] और arr[j] को swap कर सकते हैं। j+1 j

```
public static void swap(int[] arr, int i, int j)
{
    System.out.println("Swapping " + arr[i] +
        " and " + arr[j]);
    int temp = arr[i];
    arr[i] = arr[j];
    arr[j] = temp;
}
```

// अब हमें isSmaller function लिखना है। इस function का return type boolean है। ये function हमें true return करेगा अगर arr[j+1], arr[j] से बड़ा हुआ otherwise false return होगा।

```
public static boolean isSmaller(int[] arr,
        int i, int j)
{
    System.out.println("Comparing " + arr[i]
        + "and" + arr[j]);
    if (arr[i] < arr[j]) {
        return true;
    } else { return false; }
}
```

// This function is used to print the array which is sorted

```
public static void print (int [] arr)
{ for(int i=0; i<arr.length; i++)
  {
    System.out.println(arr[i]);
  }
}

public static void main (String[] args)
{ Scanner scn = new Scanner (System.in);
  int n = scn.nextInt();
  int[] arr = new int[n];

  for(int i=0; i<n; i++)
  { arr[i] = scn.nextInt();
  }
  bubbleSort(arr);
  print(arr);

}
}
```

* Now, we need to analyse the Time Complexity of Bubble Sort Algorithm.

~~Wife~~ first Journey की बात करेंगे हम अभी

$$T(n) = K(n-1) + T(n-1)$$

- $T(n)$ → Time taken for Input of Size n
- $K(n-1)$ → n-1 comparisions. और हर comparision में K time लगा. तो Time taken = $K(n-1)$ n-1 comparision करने पर 1 element sort होगया तो अब हमारे पास n-1 element है जो हमे sort करने है.
- $T(n-1)$ → Time taken for sorting n-1 elements

$$T(n) = K(n-1) + T(n-1)$$

$$T(n-1) = K(n-2) + T(n-2)$$

$$T(n-2) = K(n-3) + T(n-3)$$

$$T(2) = K$$ → 1 comparision में K time

(last में 2 element के बीच में 1 comparision होगा.

We need to add all these equations

The output that we get after adding all these equations are :-

$$T(n) = K(n-1) + K(n-2) + K(n-3) \text{ ---- } K(1).$$

$$= K\left((n-1) + (n-2) + (n-3) + (n-4) + \text{ ---- } 1\right)$$

$$= K\left\{\frac{(n)(n-1)}{2}\right\}$$ AP.

$T(n) \propto n^2$.

$T(n)$ ↓ Time taken for Input of Size n in Bubble Sort.

n^2 → is directly proportional to n^2.

Selection Sort

The Selection Sort algorithm sorts an array by repeatedly finding the minimum element (considering ascending order) from unsorted part and putting it at beginning. The algorithm maintains two subarrays in a given array.

1. The subarray which is already sorted.
2. The subarray which is unsorted

* In every iteration of Selection Sort, the minimum element (considering ascending order) from the unsorted array is picked and moved to the sorted subarray

ये array हमें given है
हमें, इस array को Selection sort use करके sort करना है।

→ Selection Sort में भी n-1 journey होगी।

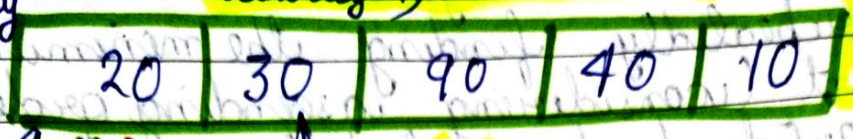
→ Journey 1 में n-1 comparison होंगे

JOURNEY 1

→ Index 0 के आगे सभी element को 0th Index वाले element से compare करनी है।

(Journey-1)

After Journey 1, the smallest element will be at its correct position



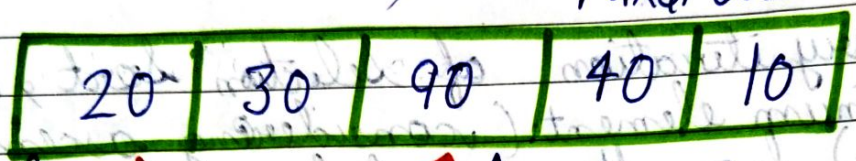
Pointer

minidx =0

C1

current element

अगर previous element (20) (minidx) बड़ा है current element (30) से, तो minidx को (current) कर दो otherwise नहीं



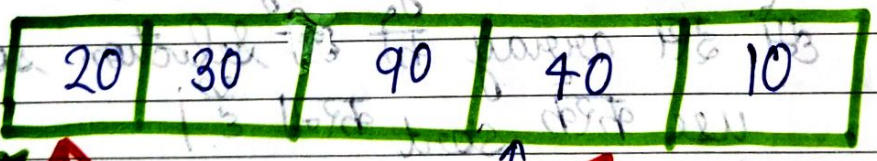
minidx=0

Pointer

current idx.

C2.

अगर minidx element (at 0th index) बड़ा है current element (at Index 2) से, तो minidx को current index कर दो



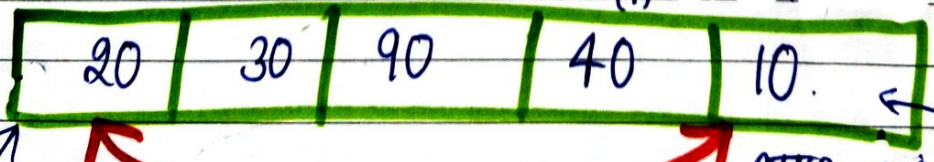
minidx=0.

Pointer

current element

C3

अगर minidx element (at Index 0) बड़ा है current element (at Index 3) से, तो minidx को current element के index के equal कर दो

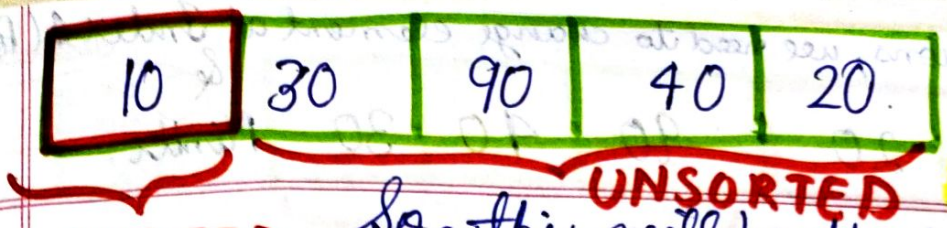


minidx = 4

Pointer

C4

अगर minidx Element (at Index 0) बड़ा है current element (at Index 4) से, तो minidx को current idx के equal कर दो



After 4 comparison swap the element at minidx & pointer

So, this will be the output after Journey 1.

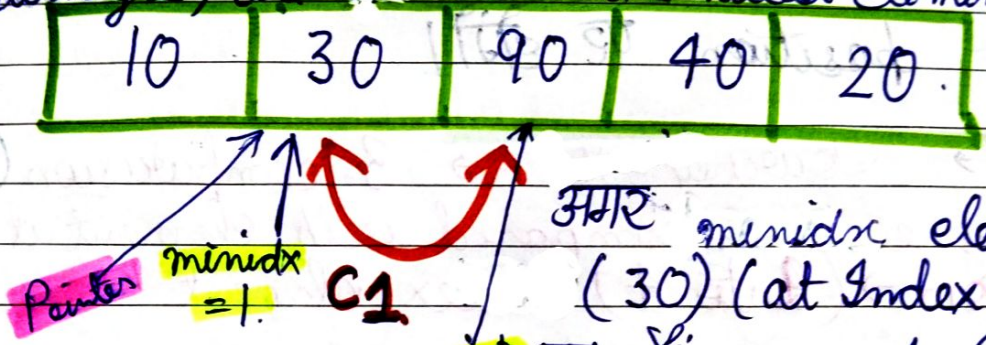
JOURNEY 2

Journey 1 → 4 comparision
With = 0th Index element
From = (1 – 4) Index element

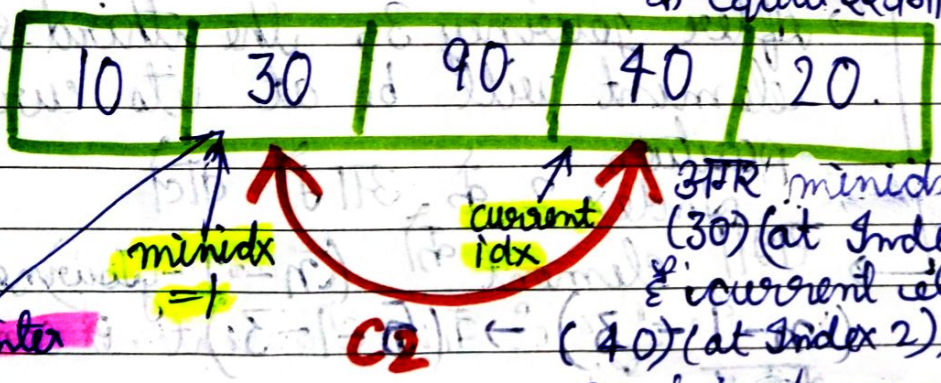
↳ सभी elements को (Journey-1) index i.e 1 index से compare करो

↳ After Journey 1, first element तो sort होगया है; Now we need to start the sorting for Index 1 till Index n-1.

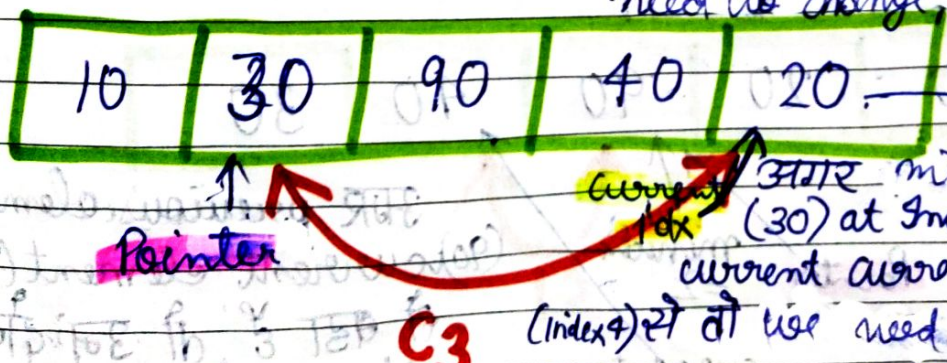
→ After Journey 2; the second smallest element will be at its current position



अगर minidx element (30) (at Index 1) बड़ा है current element (90) (at Index 2) तो हमें minidx को हमें current idx के equal रखना है।

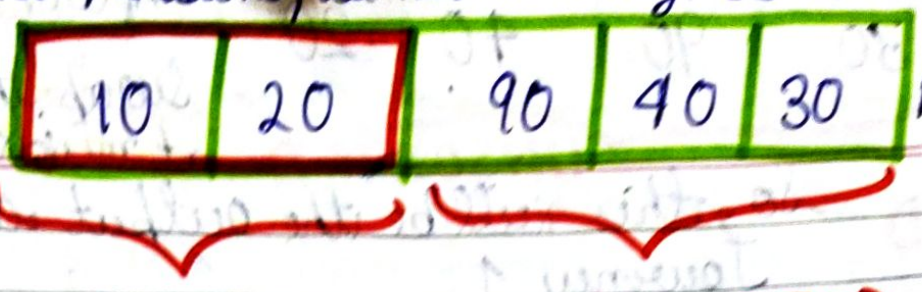


अगर minidx element (30) (at Index 1) बड़ा है current element से (40) (at Index 2), then we need to change minidx to current Idx → minidx = 4



अगर minidx element (30) at Index 1 बड़ा है current current element (20) (Index 4) से तो we need to change minidx to current idx

After 3 comparisions, we need to change element at Index 1 (minidx) & Minidx.



SORTED UNSORTED

So, this will be the output after Journey 2.

सबसे छोटे 2 elements उनकी correct position पर होंगे।

→ Journey = 2 → 3 comparision (n-Journey)

→ Every element compared with element at Index = 1.

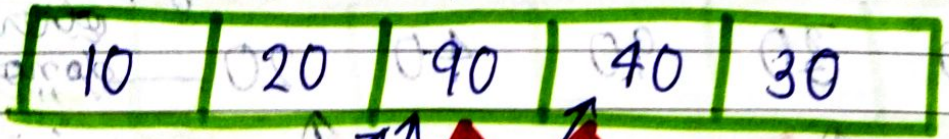
→ Elements compared → (2 to 4) Index element

JOURNEY 3

→ Journey 3 में (n-Journey) ; 2 comparision होंगे)

↳ After journey 3, the third smallest element will be at its current position.

→ Index 2 के आगे वाले सभी element को (Journey - 1) index ie. Index 2 वाले element से compare करना है।



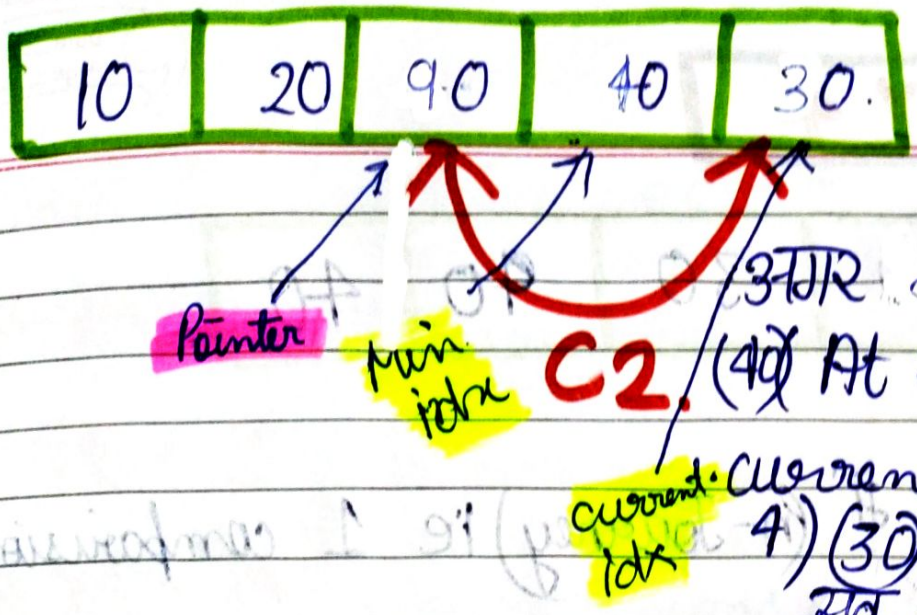
Pointer

minidx = 2

current idx = 3

C1

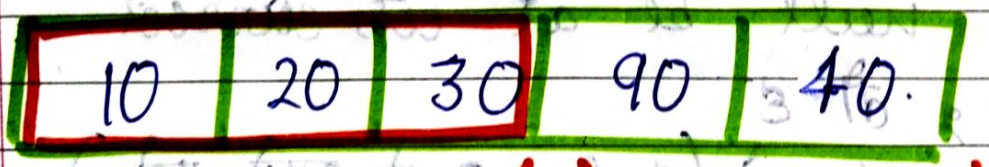
फिर minidx element (Index 2) (90) current element (Index 3) (40) से बड़ा है तो minidx की current idx के equal कर लो, so now minidx = 3



3rd R minidx Element (minidx) (40) At Index 3) वहाँ है current element (at Index 4) (30) से तो minidx की तरह current Idx. के equal कर दो।

∴ Now minidx = 4

Now swap the element at Pointer & Min idx.



SORTED **UNSORTED.**

So, this will be the output after Journey 3.

We can see that Smallest 3 elements are at their current position.

→ Journey 3 → Did (n-Journey) i.e 2 comparisons

→ Every element compared with element at Index → 2 (Journey - 1)

→ Elements compared → (3 to 4) Index elements.

JOURNEY 4

10	20	30	90	40

→ Journey 4 में (n-Journey) ie 1 comparision होगी

→ After Journey 4, the forth smallest element will be at its correct position

→ सभी elements को (Index 3 के आगे वाले) Index 3 के element से compare करनी है;

10	20	30	90	40

minidx = 3 Pointer currentidx

∴ अगर minidx element at Index 3 बड़ा है current element at Index 4 से, तो minidx को current idx के equal कर देंगे। ∴ Minidx = 4

∴ Now, swap element at Pointer & minidx

10	20	30	40	90

SORTED

SORTED

अगर n-1 elements अपनी correct position पर है तो nth element automatically अपनी correct position पर होगा।

JOURNEY 1	Comparisions = 4. (n - Journey) = n-1	Element Sorted = 1
JOURNEY 2	Comparisions = 3. (n-2)	Element Sorted = 2
JOURNEY 3	Comparisions = 2 (n-3)	Element Sorted = 3
JOURNEY 4.	Comparisions = 1 (n-4)	Element Sorted = 4

हम ये बोल सकते हैं कि 1 Journey के बाद एक element sort हो जाएगा

तो अगर हमारे पास array में 5 elements थे तो अगर हमारी Journey 4 हुई तो 4 Journey के बाद 5 में से 4 elements तो sort हो गए तो 5th element खुद ही अपनी correct position पर पहुँच गया और हमें 5th Journey की जरूरत ही नहीं पड़ी।

∴ Array Elements = 5.
Journey = 4.
Elements to be sorted = 4 (अगर 4 elements sort हो गए तो 5th element automatically sort हो जाएगा

Selection Sort <code>

→ import java.util.*;

→ import java.io.*;

→ public class main {

public static void selectionSort(int[] arr)

{

loop for Journey { for(int itr = 0; itr <= arr.length-2 ; itr++)

itr → iteration

अगर n 5 है तो Journey 4 होगी (0 से 3) = 4
(1 से 4) = 4

{ int minidx = itr ; → iteration / journey

minimum index (सबसे छोटा element इस index पर है)

वो loop comparision के लिए है { for (int j = itr+1 ; j < arr.length-1 ; j++)

हर Journey में (n-Journey/iteration) comparision होगा।

{ if (isSmaller(arr, j, minidx) == true)

{ minidx = j ;

अगर element at arr[j] < arr[minidx], then we need to swap them.

} } }

swap(arr, itr, minidx);

}

}

Now, we have successfully made function for Selection Sort
Now, we need to make a function for swapping the element at array[j] and arr[iteration].

→ public static void swap(int[] arr, int i, int j)

{ System.out.println("Swapping " + arr[i] + " and " + arr[j]);

int temp = arr[i];
arr[i] = arr[j];
arr[j] = temp;

}

Now, we need to create isSmaller function

→ public static boolean isSmaller(int[] arr, int i, int j)

{ System.out.println("Comparing " + arr[i] + " and " + arr[j]);

if (arr[i] < arr[j])
{ return true; }
else { return false; }

}

// Now we need to create the print function.

```
public static void print (int [] arr)
  {
    for (int i=0; i<arr.length; i++)
      { System.out.println (arr[i]);
      }
  }
// Here is the main function
public static void main (String[] args)
  {
    Scanner scan = new Scanner (System.in);
    { int n = scan.nextInt();
      int [] arr = new int [n];
      for (int i=0; i<n; i++)
        { arr[i] = scan.nextInt();
        }
      selectionSort(arr);
      print (arr);
    }
  }
```

Now, we need to analyse the Time Complexity of Selection Sort.

↳ Time Complexity of Selection Sort is same as of Bubble sort

→ Journey 1 → $T(n) = K(n-1) + T(n-1)$

→ Journey 2 → $T(n-1) = K(n-2) + T(n-2)$

→ Journey 3 → $T(n-2) = K(n-3) + T(n-3)$

Add these eqns. now

$T(2) = K$

last Journey में 2 elements ही compare किये थे.

तो 1 comparision b/w 2 elements

Now, add all these equations

$$T(n) = K(n-1) + K(n-2) + K(n-3) \text{ ---- } K(1)$$

$$= K((n-1) + (n-2) + (n-3) \text{ ---- } (1))$$

$$= K\left[\frac{(n)(n-1)}{2}\right]$$

$T(n) \propto n^2$

↳ Time taken for Input of Size n is directly proportional to n^2.

Both of the (Bubble Sort & Selection Sort) use $O(n^2)$ time complexity which is such a bad time complexity.

So, for now there is almost no real use case of Bubble Sort & Selection Sort, instead we will be using the Merge Sort and Quick Sort.

Recursion Question

↳ Fibonacci using Recursion

↓

Fibonacci Series में सबसे पहले 0 और 1 आते थे और आगे के सब elements will be the sum of previous 2 consecutive fibonacci elements.

$fib(0) = 0 \quad fib(1) = 1$

$fib(2) = fib(1) + fib(0)$

$= 1 + 0$

$= 1$

$fib(7) = fib(6) + fib(5)$

$= 8 + 5$

$= 13$

0 1 1 2 3 5 8 13 21 34 55

0 1 2 3 4 5 6 7 8 9 10

RECURSION

HIGH LEVE THINKING

LOW LEVEL THINKING

(older)

HIGH LEVEL THINKING

Expectation

हम expect करते हैं कि हमें nth position का fibonacci element ~~मिल जाएगा~~ मिलेगा

if n = 5, output →

0 1 1 2 3 5

→ 0 1 2 3 4 5

Faith

हम यहाँ faith रखते हैं कि fib(n-1) और fib(n-2) की value हमें Recursive calls करने से मिल जाएगी

```
fibnm1 = FIB(n-1);
fibnm2 = Fib(n-2);
```

if n = 5

fibnm1 = 3

fibnm2 = 2

Expectation meets faith

हमें अपनी expectations अपने faith से पूरी करनी है।

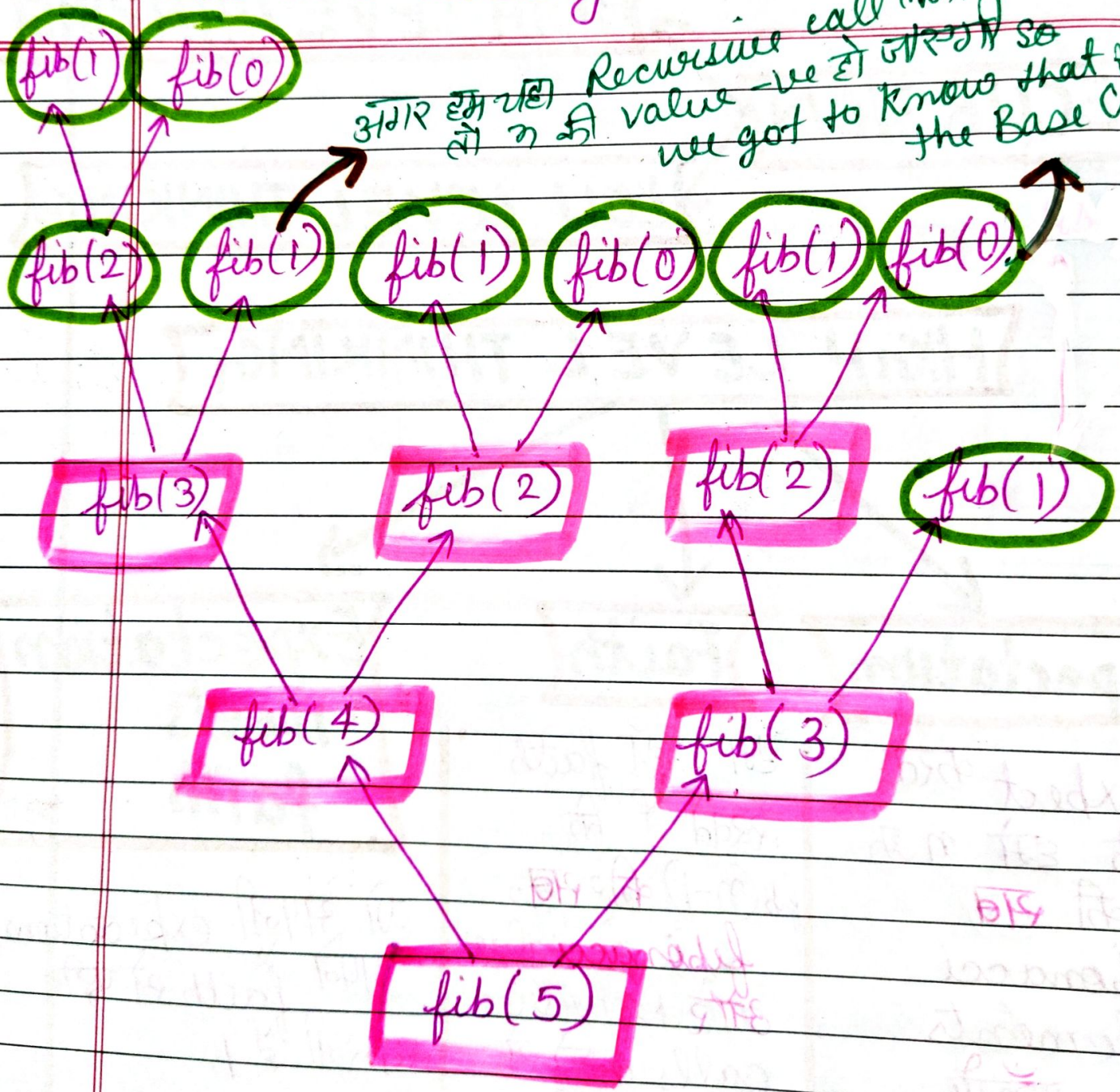
fib(n) = fibnm1 + fibnm2

fib(5) = fib(4) + fib(3)

= 3 + 2

= 5.

Low Level Thinking



आगे हम यहाँ Recursive call नहीं करेंगे क्योंकि यहाँ value -ve हो जाएगी so we got to know that this is the Base Case

Fibonacci using Recursion

<code>

```
import java.util.*;
import java.io.*;
public class Main
{
public static int Fib(int n) {
  if (n==0) || n==1)
  { return n;
  }.

  int fibnm1 = Fib (n-1) ;
  int fibnm2= Fib (n-2);

  int fib n = fibnm1 + fibnm2;

  return fib n;
}.
public static void main ( String [ ] args);
Scanner scan = new Scanner (System.in);
    int n= scan.nextInt ();
    int ans = Fib (n);
    System.out.println (ans);
}
}
```