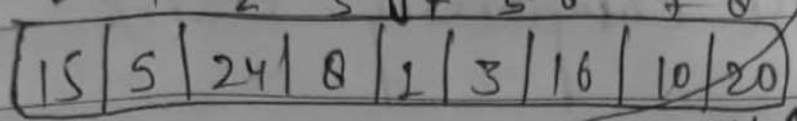
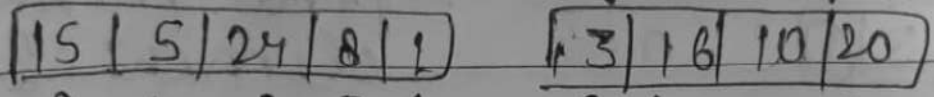
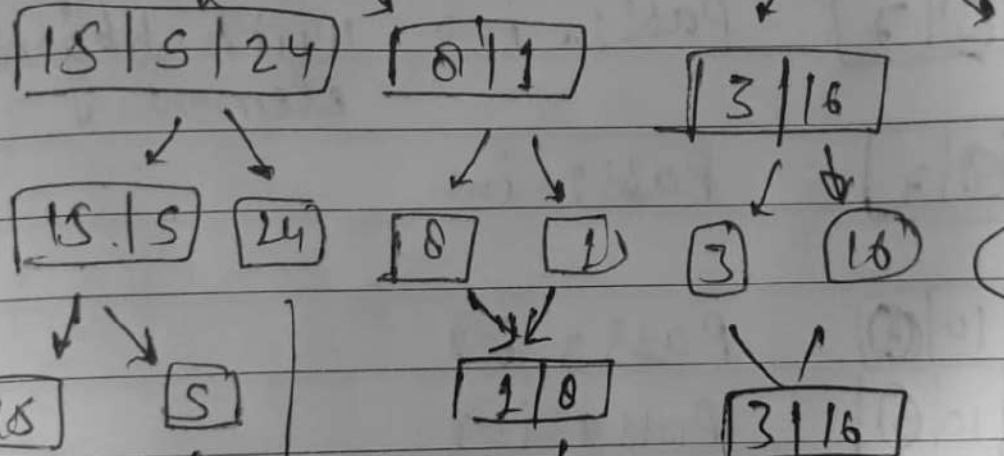
}

Merge Sort (O(m+n))

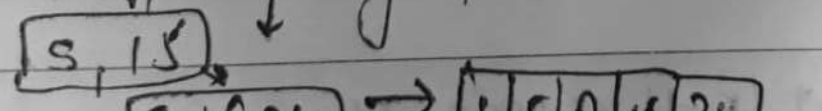


Here we divide it into sublists, $\frac{0+8}{2} = 4^{th}$ index





Now merge,



final sorted list

THURSDAY 29
DAY 333-032

DECEMBER

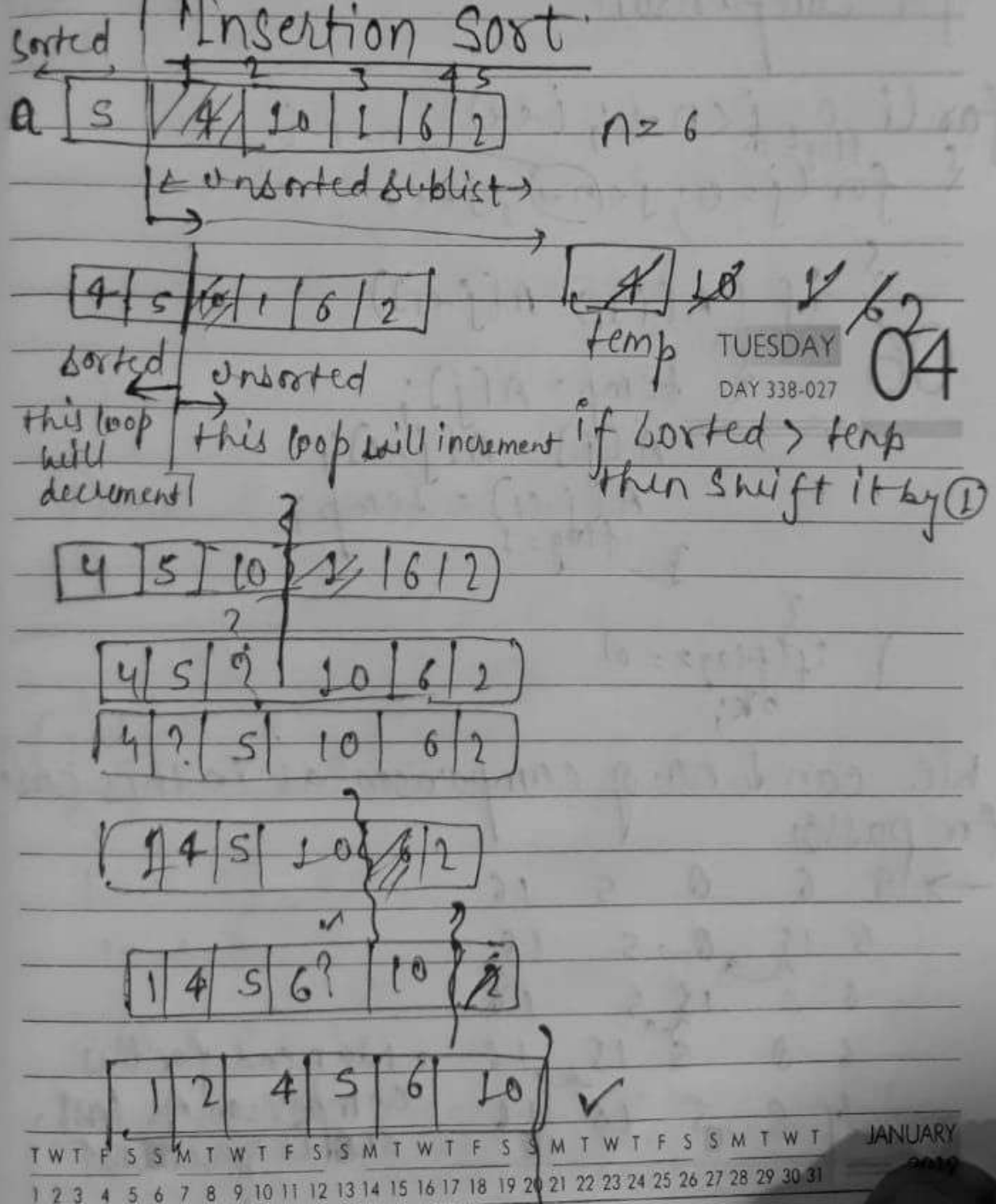
S S M T W T F S S M T W T F S S M T W T F S S M T W T F S S M
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

Similarly, as pass 7, no. of comparison can be ↓.

So, we can avoid unnecessary comparison by using (n-1-i)

NOTE:- Why I use flag, bcz there are cases when we get swapped elements before (n-1) execution so we will come out of loop if it happen.

Insertion Sort



TUESDAY 04
DAY 338-027

JANUARY 2019

15 6 8 15 16 → after Pass 4

DECEMBER 2018

If we have (n) element then for Bubble Sorting (n-1) passes are required bcz last one (or) first element get its position by default.

Atmost case not alwways true.

In this algo, we required (2) loops outer loop for passes & inner loop for comparison.

```
for(i=0; i<n-1; i++)          → n-1-i
{  flag=0;
   for(j=0; j<(n-1); j++)
   {  if (A[j] > A[j+1])
      {  temp = A[j];
         A[j] = A[j+1];
         A[j+1] = temp;
         flag = 1
      }
   }
   if(flag==0)
      bk;
```

06 THURSDAY DAY 340/25

We can ↓ no. of comparison as in this case for pass (2)

→ 15 6 8 5 16

6 15 8 5 16

6 8 15 5 16

6 8 5 15 16 → No need for this comparison As last one is greatest

6 8 5 15 16

```
if (start < end)
{ swap(a[start], a[end]);
  }
}
swap(a[lb], a[end]);
  return end;
}
```

Quicksort (A, lb, ub)

```
{ if (lb < ub)
  {
    loc = Partition (A, lb, ub);
    Quicksort (A, lb, loc-1);
    Quicksort (A, loc+1, ub);
  }
}
```


Time Complexity = $O(n)$ → best
$O(n^2)$ ⇒ worst
$O(n \log n)$ → ~~Avg~~

Space = $O(\log n)$

BUBBLE SORT

	0	1	2	3	4
A	15	16	6	0	5

Pass (1)

15 6 16 0 5
15 6 0 16 5
15 6 0 5 16 → Largest element

Now again, do pass 2, 3 etc. until we got correct order.

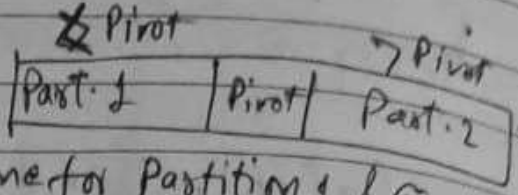
DECEMBER

Quick Sort

10 MONDAY 2018

0	1	2	3	4	5	6
10	15	1	2	9	16	11

Pivot = 10



NOW, again apply same for Partitions ① & ② Logic.

fix

0	1	2	3	4	5	6
2	1	9	10	15	11	16

0-2 4-6

Eg.

0	1	2	3	4	5	6	7	8
7	6	10	5	9	2	1	15	7

lb ub

Pivot = 7

start start end

7	6	7	5	3	2	1	15	10

start end

11 TUESDAY

7	6	7	5	1	2	9	15	10

start end

2	6	7	5	1	7	9	15	10

same apply

Array

```
Partition (A, lb, ub)
{
  Pivot = a[lb]
  start = lb;
  end = ub;
  while(start < end)
  {
    while [a[start] <= Pivot)
      { start++;
      }
    while (a[end] > Pivot)
      { end--;
```

2048 ALGORITHM FOR QUICK SORT- DECEMBER

Partition (ARR, BEG, END, LOC) WEDNESDAY 12

STEP 1. [INITIALIZE] SET LEFT = BEG, DAY 346-019
RIGHT = END, LOC = BEG, FLAG =
ST. 2. Repeat steps 3 to 6 while flag =
ST. 3. Repeat while ARR[LOC] <= ARR[RIGHT]
AND LOC != RIGHT
SET RIGHT = RIGHT - 1
[END OF LOOP]
ST. 4. If LOC = RIGHT
SET FLAG = 1
ELSE IF ARR[LOC] > ARR[RIGHT]
SWAP ARR[LOC] with ARR[RIGHT]
SET LOC = RIGHT
[END OF IF]
ST. 5. IF FLAG = 0 THURSDAY 13
Repeat while ARR[LOC] >= ARR[LEFT]
& LOC != LEFT.
SET LEFT = LEFT + 1
[END OF LOOP]
ST. 6. IF LOC = LEFT
SET FLAG = 1
ELSE IF ARR[LOC] < ARR[LEFT]
SWAP ARR[LOC] with ARR[LEFT]
SET LOC = LEFT
[END OF IF]
[END OF IF]
ST. 7. [END OF LOOP]
ST. 8. [END]

ASSING.2.

1. Insertion sort is simple sorting algorithm with quadratic worst-case time complexity.
Worst case :- when array is given in descending order & we have to sort in ascending order.
Time complexity ≃ $O(n^2)$.

For Best case :- Suppose, if we given already sorted data in ascending order then,
Time complexity ≃ $(O(n))$

So, this is the analysis to convert insertion sort time complexity $O(n^2)$ to $O(n)$ iff we are provided with already sorted data.

2. Quick sort → in-place
Bubble sort → in-place

Algo.	Best	Avg	Worst	space (worst)
Quick	O(nlogn)	O(nlogn)	O(n^2)	O(logn)
Bubble	O(n)	$O(n^2)$	$O(n^2)$	O(1)
Merge	O(nlogn)	O(nlogn)	O(nlogn)	O(n)
Insertion	O(n)	$O(n^2)$	$O(n^2)$	O(1)

Now, here is algorithm & discussion with an example.
* There are programs also.