Algorithms

Date: / / /

O(1) → space
O(n²) → time

Q Insertion sort → like sorting a hand of cards. Pick a card from right & give it a place in left hand.

time = O(n²)

pseudo code →

```
for(i = 1; i<n; i++)
    t = a[i];
    for(j=i; j>0 && temp<a[j-1]; j--)
        a[j] = a[j-1];
    a[j] = t;
```

best case → O(n) when already sorted

No. of

| 5 | 2 | 4 | 6 | 1 |

i, j, t

t = 2

t < a[j-1] swap

j != 0 a[0] = t ×

| 2 | 5 | 4 | 6 | 1 |

i, j, t

t = 4

a[2-1] = t swap

t < a[j-1] ∴ ×

a[j] = t

| 2 | 4 | 5 | 6 | 1 |

t = 6

a[j-1] < t ×

a[j] = t

| 2 | 4 | 5 | 6 | 1 |

t = 1

2 4 5 6 6

2 4 5 5 6

2 4 4 5 6

2 2 4 5 6

1 2 4 5 6

j != 0 ×

Scanned with CamScanner

Prefered where → complexity doesn't matter
→ short code needed

② BUBBLE SORT

$O(n^2)$ → time
$O(1)$ → space

In every pass 1st element takes it posn → eg in 3 element - in 1st iteration 1st element takes 3rd posn, 2nd it → 2nd highest take 3rd posn & ultimately last highest gets it last posn. ∴ n-1 iteration are required.

Total Max swaps needed → $\frac{n(n-1)}{2}$

Iteration ↓ $(n-1)$

swaps → 1st it → 1st highest = needs 2 swaps to reach 3rd posn max.

swap max to reach end posn, 2nd highest needs 1

pseudo code →

```
for (i=1 ; i<n ; i++)        → n-1 iteration
    for (j=0; j<n-i ; j++)   → n-i swaps for each iteration
        if a[j] > a[j+1]
            swap(a[j], a[j+1])
```

i → 1, 2

| 70 | 40 | 50 |

↓ ↓ j+1

| 40 | 70 | 50 |

↓ ↓ j+1

| 40 | 50 | 70 |

i = 1 J = 3-1 values = 2

| 40 | 50 | 70 |

↓ ↓ j+1

No swap needed

i = 2 J = 3-2 values = 0

| 40 | 50 | 70 |

| 40 | 50 | 70 | → done

Scanned with CamScanner

O(1) → space

③ SELECTION SORT → $O(n^2)$ → time

No. of comparisons → $\frac{n(n-1)}{2}$

min element is placed at its posn compar by each element next in the series

pseudocode →

```
for(i=0; i<n; i++)
{
    min = i;
    for(j=i+1; j<n; j++)
        if(a[j] < a[min])
            min = j;
    temp = swap(a[i], a[min])
}
```

10	30	20	50	40

i=0 j = 1,2,3,4

min = 0

all sorted steps

10	30	20	50	40

i=1 j = 2,3,4

10	20	30	50	40

i=2 j = 3,4

10	20	30	50	40

i=3 j = 4

10	20	30	40	50

done

Scanned with CamScanner

④ DIVIDE AND CONQUER APPROACH →

Date: 1/1/

DIVIDE → divide into smaller subprob

CONQUER → solving sub prob recursively

COMBINE → combine sol of sub into the sol of original prob

I MERGE SORT → time → $O(n \log n)$, space → $O(n)$

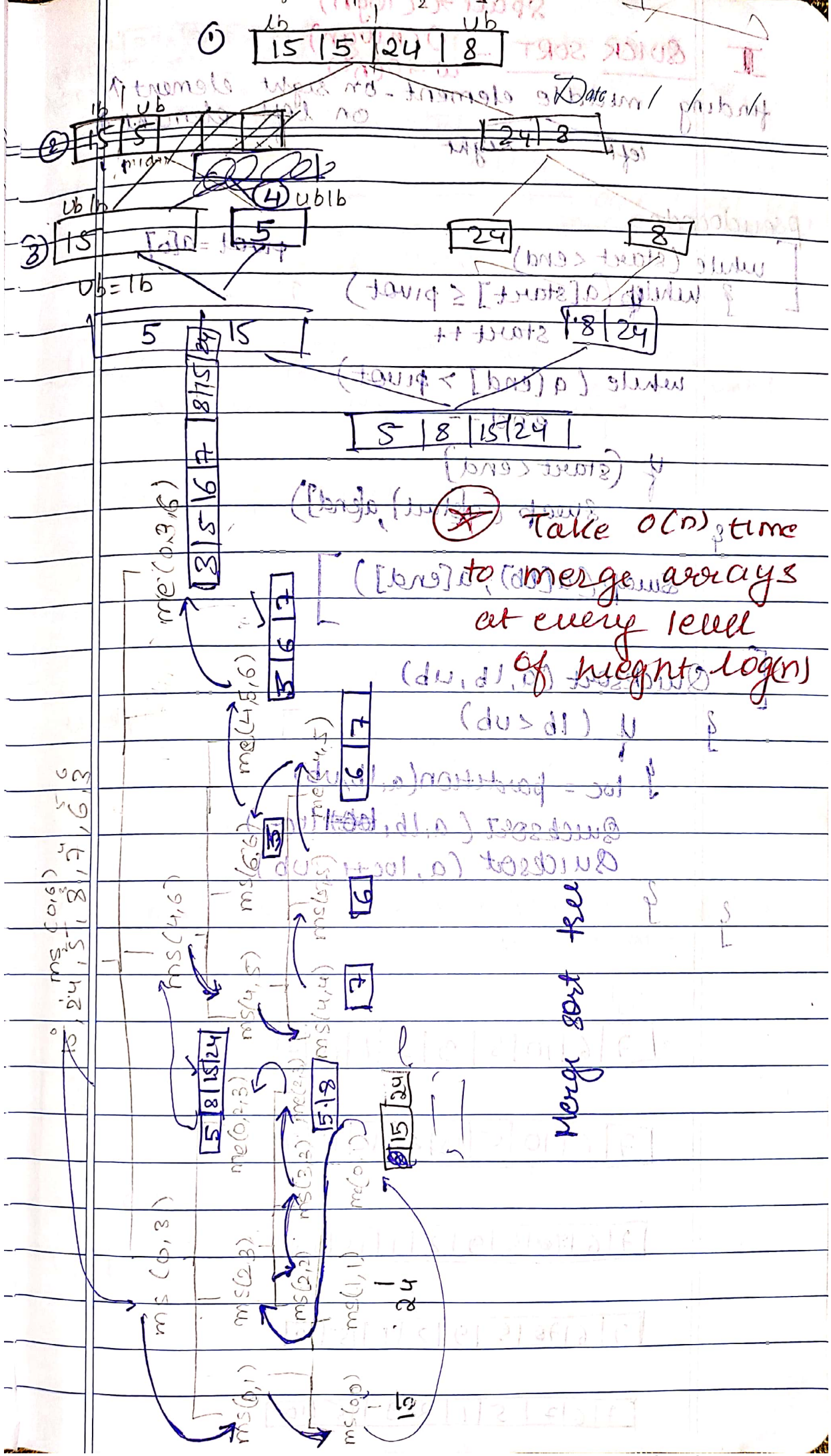
divide n subsequence into 2 subarrays of $n/2$, sort these subarrays recursively & Merge the sorted subarrays.

pseudocode → Merge sort (A, lb, ub)

```
{ if (lb < ub)
  { mid = lb+ub/2
    merge sort (A, lb, mid)
    merge sort (A, mid+1, ub)
    merge ( A, lb, mid, ub)
  }
}

Merge (A, lb, mid, ub)
{ i = lb  j = mid+1  k = lb
  while (i <= mid && mid+1 <= ub)
  { if (a[i] <= a[j])
    { b[k] = a[i];
      i++; k++; }
    else
      b[k] = a[j]
      k++  j++
  }
  if (i > mid)
    while (j <= ub)
      b[k] = a[j]
      j++  k++
  else if (j > ub)
    while (i <= mid)
      b[k] = a[i]
      i++  k++.
}
```

Scanned with CamScanner



Scanned with CamScanner

I

QUICK SORT → Space → O(log n)
→ O(n log n) → Tree of w = log n with every step n
→ w → O(n²)

finding middle element - on right element & on left element

left right

pseudocode →

```
while (start < end)
{
    while (a[start] ≤ pivot)
        start ++
    while (a[end] > pivot)
        end --
    if (start < end)
        swap (a[start], a[end])
}
swap (a[lb], a[end])
```

```
Quicksort (a, lb, ub)
{
    if (lb < ub)
    {
        loc = partition(a, lb, ub)
        Quicksort (a, lb, loc-1)
        Quicksort (a, loc+1, ub)
    }
}
```

pivot = a[0]

partition

Quicksort

| 7 | 6 | 10 | 5 | 9 | 2 | 1 | 15 | 7 |

pivot = 7

checking ① & ②

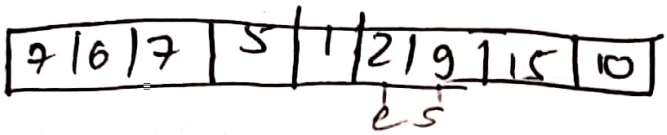
swap

checking ① & ②

swap

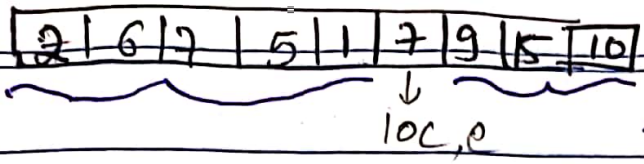
checking ① &

Scanned with CamScanner



end < start

swap(a(lb), end)



loc, e

repeat.

Choose pivot as middle element

SEARCHING

(1) LINEAR SEARCH → Start at one end check every item until find key.

good for n<=100

Pseudocode: search (a, n, key)

```
{ for (i=0, i<n, i++)
    if (a[i] = key)
      return i
  return -1 .}
```

if -1 → not found

else → found at index i

$f(n) = O(n)$

(2) BINARY SEARCH → sort elements search at middle $O(\log n)$

Scanned with CamScanner