Algorithms

Date: / / /

$O(1)$ → space
$O(n^2)$ → time

① Insertion sort → like sorting a hand of cards. Pick a card from right & give it a place in left hand.

$\boxed{\text{time} = O(n^2)}$

pseudo code →

```
for(i = 1; i < n; i++)            best case → O(n)
{   t = a[i];                     when already sorted
                                  → this won't occur
    for(j = i, j > 0 && temp < a[j-1]; j--)
    {   a[j] = a[j-1];  }
    a[j] = t;
}
```

No. of n, $n-1$, n, $(n-1)^2$, $n-1$

5	2	4	6	1

j-1 → i, j, t

$t = 2$

$t < a[j-1]$ swap

5	5	4	6	1

$j \not> 0$ $a[0] = t$ ×

2	5	4	6	1

j-1 → i, j, t

$t = 4$

$a[j-1] > t$ swap

2	5	5	6	1

j-1 j

$t \not< a[j-1]$ ×

$a[j] = t$

2	4	5	6	1

j-1 i, j, t

$t = 6$

$a[j-1] \not> t$ ×

$a[j] = t$

2	4	5	6	1

j-1 → i, j, t

$t = 1$

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

$j \not> 0$ ×

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

② BUBBLE SORT $O(n^2)$ → time
$O(1)$ → space

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

In every pass 1st element takes it posn → eg in 3 elements - 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.

iteration ↓ (n-i) swaps → 1st it → 1st highest needs 2 ~~1~~ swap to reach 3rd posn max.
to reach 2nd posn, 2nd highest needs 1 swap max

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 | i=1   j = 3-1 value = 0,1
 j   j+1

| 40 | 70 | 50 |
 j   j+1

| 40 | 50 | 70 |

| 40 | 50 | 70 | i = 2   J: 3-2 value = 0
 j   j+1   No swap needed

| 40 | 50 | 70 |

| 40 | 50 | 70 | → done

③ SELECTION SORT

O(1) → space
→ O(n²) → 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++)
        ~~min =~~ if (a[j] < a[min])
            min = j;
}
~~temp~~ - swap [a[i], a[min]])
```

10 | 30 | 20 | 50 | 40

min

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

min = 0

all sorted steps

10 | 30 | 20 | 50 | 40

min

i=1   j = 2,3,4

min

10 | 30 | 20 | 50 | 40

10 | 20 | 30 | 50 | 40

i=2   j = 3,4

min

10 | 20 | 30 | 50 | 40

min

10 | 20 | 30 | 50 | 40

i=3   j = 4

10 | 20 | 30 | 50 | 40

min   j > n

10 | 20 | 30 | 40 | 50

done

1) DIVIDE AND CONQUER APPROACH →

Date: / /

DIVIDE — divide into smaller ~~subprob~~

CONQUER — solving sub prob ~~divided~~ recursively

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

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

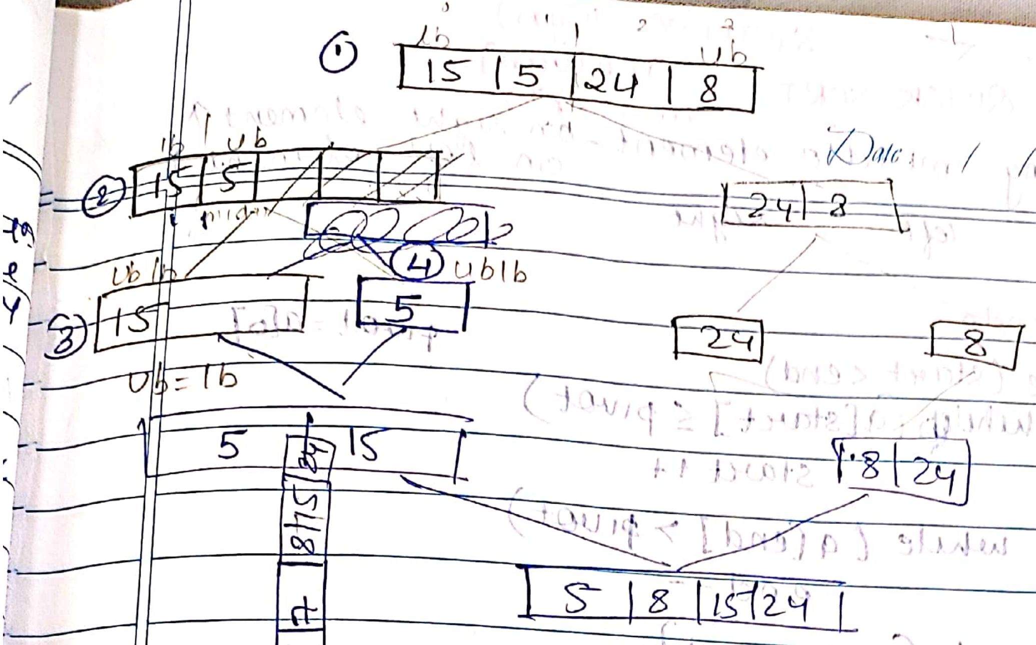
divide n subsequence into 2 subarrays of $n/2$, sort these subarrays recursively. Merge the sorted subarrays.

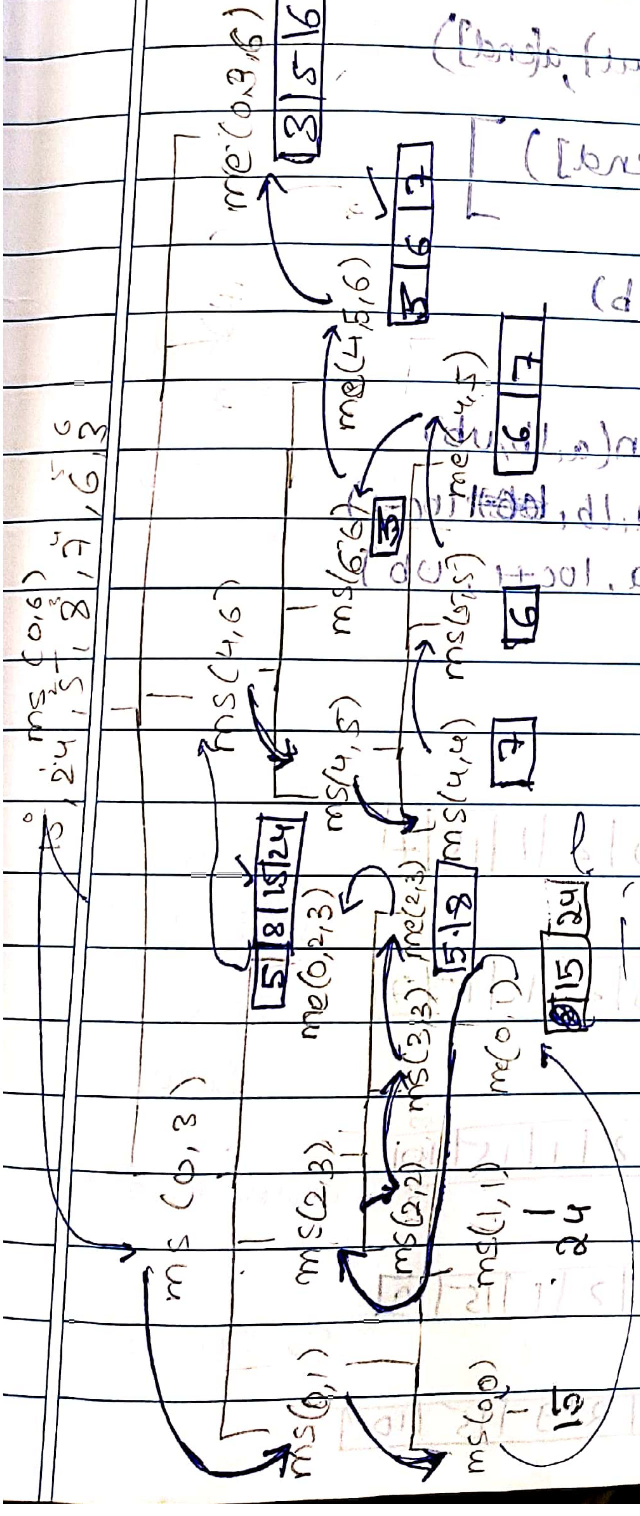
pseudocode → ~~[illegible]~~ Merge sort (A, lb, ub)

```
{ 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~~ j<=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++.
}
```





Take O(n) time to merge arrays at every level of height log(n)

Merge sort tree

I QUICK SORT → O(n log n) → Tree of h = log n with every step n

space → O(log n)

w → O(n²)

finding middle element - on right element ↑ / on left element ↓

left ← → right

s						e

pivot = a[0]

pseudocode →

```
while (start < end)
{
    while (a[start] ≤ pivot)   ①
        start ++
    while (a[end] > pivot)   ②
        end--
    if (start < end)
        swap (a[start], a[end])
}
swap (a[lb], a[end])
```

partition

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

Quicksort

s p								e
7	6	10	5	9	2	1	15	7

pivot = 7

checking ① & ②

7	6	10	5	9	2	1	15	7
		s						e

Swap

7	6	7	5	9	2	1	15	10
		s						e

checking ① & ②

7	6	7	5	9	2	1	15	10
				s		e		

Swap

7	6	7	5	1	2	9	15	10
				s		e		

checking ① & ②

7	6	7	5	1	2	9	15	10
					e	s		

end < start

swap (a(lb), end)

2	6	7	5	1	7	9	15	10
					↓ 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 ~~els~~ -1 . }
```

if -1 → not found

else → found at ndex i

$f(n) = O(n)$

(2) BINARY SEARCH → sort elements

search at middle

$\Theta(\log n)$