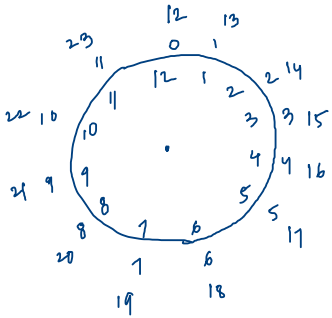
★ Time Conversion :



12 - hour		24 - hour	
06 : 30 : 00 AM	as it is →	06 : 30 : 00	(Just remove AM)
11 : 25 : 30 AM	→	11 : 25 : 30	
12 : 25 : 12 AM	→	00 : 25 : 12	(make hrs = 0)
[midnight]			
1 : 25 : 52 PM	→	13 : 25 : 52	(hrs + 12)
11 : 59 : 59 PM	→	23 : 59 : 59	
12 : 00 : 00 PM	→	12 : 00 : 00	as it is
12 : 12 : 32 PM	→	12 : 12 : 32	
[AFTERNOON]			

★ "AM"
- h == 12 → as it is
- h == 12 → make h = "00"

★ "pm"
- h != 12 → h + 12
- h == 12 → as it is

★ Sorting :

→ [5, 2, 1, 3, 4] → [1, 2, 3, 4, 5] (ascending)
→ [5, 4, 3, 2, 1] (descending)

→ class of students → sort them by heights (assembly line)
→ sort them by marks (rank)

→ how to do ?

- Bubble sort
- selection sort
- Insertion sort

not optimal

★ count sort
→ Technique

- Merge sort
- Quick sort

} rec, divide-conquer used by inbuilt sort() (optimal)

Bucket sort
radix sort

- heap sort → heaps / priority Queue

* Bubble sort :

→ try to bring max_element to last index

arr[i] , arr[i+1]

0 1 2 3

arr: [6, 0, 3, 5]

1st pass (3)

0 1 2 3

[6, 0, 3, 5]

[0, 6, 3, 5]

[0, 3, 6, 5]

[0, 3, 5, $\boxed{6}$ *]

2nd pass (2)

[0, 3, 5, $\boxed{6}$ *]

[0, 3, 5, 6]

[0, 3, $\boxed{5}$ *, 6]

3rd pass (1)

[0, 3, $\boxed{5}$ *, $\boxed{6}$ *]

[0, $\boxed{3}$ *, 5, 6]

4th pass (0)

[0, $\boxed{3}$ *, $\boxed{5}$ *, $\boxed{6}$ *]

* only one element is remaining i.e; it is already fixed hence 4th pass not needed

* Bubble sort does not know that array is sorted until all passes are completed.

```
function bubbleSort(arr) {
  const n = arr.length;
  let swaps = 0;
  // passes
  for (let fixIdx = n - 1; fixIdx > 0; fixIdx--) {
    // comparing adjacent elements
    for (let i = 0; i < fixIdx; i++) {
      if (arr[i] > arr[i + 1]) {
        [arr[i], arr[i + 1]] = [arr[i + 1], arr[i]];
        swaps++;
      }
    }
  }
  return swaps;
}
```

1st

[a, b, c, $\boxed{d}$]

[a, b, c, d]

[a, b, c, d]

3 swaps

3 iterations

(n-1)

2nd

[a, b, $\boxed{c}$, d]

[a, b, c, d]

2 swaps

2 iterations

(n-2)

3rd

[a, $\boxed{b}$, c, d]

1 swap

1 iteration

★ TC : $n-1+n-2+\cdots+1$

$$\frac{n(n-1)}{2} = O(N^2)$$

★ max_swaps in bubble sort $= \frac{n(n-1)}{2}$

★ SC : $O(1)$

★ Selection sort :

★ pick minimum element and place in right position

arr : $[\overset{0}{4}, \overset{1}{3}, \overset{2}{2}, \overset{3}{1}]$

① what will be the correct element at $\boxed{idx = 0}$,

$\Rightarrow \min([4, 3, 2, 1]) = 1$

$[4, 3, 2, 1] \Rightarrow [\boxed{1}, 3, 2, 4]$ ★

② what will be the correct element at $\boxed{idx = 1}$,

$\Rightarrow \min([3, 2, 4]) = 2$

$[1, 3, 2, 4] \Rightarrow [1, \boxed{2}, 3, 4]$ ★

★ swapping with itself is unneccessary

③ what will be the correct ele at $\boxed{idx = 2}$

$\Rightarrow \min([3, 4]) = 3$

$\Rightarrow [1, 2, 3, 4] = [1, 2, \boxed{3}, 4]$ ★

~~④~~ what will be the correct ele at $\boxed{idx = 3}$

not required

$\Rightarrow \min([4]) = 4$

$\Rightarrow [1, 2, 3, 4] = [1, 2, 3, \boxed{4}]$ ★

★ only one element left it is already fixed.

```javascript
function selectionSort(arr) {
  const n = arr.length;
  for (let fixIdx = 0; fixIdx < n - 1; fixIdx++) {
    // find the minimum ele
    let minEle = Infinity;
    let minEleIdx = -1;
    for (let i = fixIdx; i < n; i++) {
      if (arr[i] < minEle) {
        minEle = arr[i];
        minEleIdx = i;
      }
    }

    // avoid un-necessary swappings
    if (fixIdx != minIdx) {
      [arr[fixIdx], arr[minEleIdx]] = [arr[minEleIdx], arr[fixIdx]];
    }
  }
}
```

[a, b, c, d]

1. min(a, b, c, d) → 4 iterations → 1 swap fI = 0
2. min(b, c, d) → 3 iterations → 1 swap fI = 1
3. min(c, d) → 2 iterations → 1 swap fI = 2

★ TC: $n + n-1 + n-2 + \cdots + 2$

$\Rightarrow \frac{n(n+1)}{2} - 1 \Rightarrow O(N^2)$

★ max_swap in selection $\Rightarrow n-1$

★ SC: $O(1)$

★ which is better Bubble / selection ?
(worst case)

A: Selection due to less no-of swaps

★ selection and Bubble are $O(N^2)$ in all cases best, average, worst

[1,2,3,4] [1,2,4,3] [4,3,2,1]