9/6/2024

Analytical Assignment - 2

ABIRAMY. K. J
192311016
CSA0669
DAA

(1) If $t_1(n) \in O(g_1(n))$ and $t_2(n) \in O(g_2(n))$, then $t_1(n) + t_2(n) \in O(\max\{g_1(n), g_2(n)\})$. Prove the assertions.

Solution:

$$t_1(n) \in O(g_1(n)), \quad t_2(n) \in O(g_2(n))$$

Since, $t_1(n) \in O(g_1(n))$

$f(n) \leq c(g_1(n))$, we write it as

$$\boxed{t_1(n) \leq c\, g_1(n)}$$ where $c_1 \rightarrow$ some constant

Since $t_2(n) \in O(g_2(n))$

$f(n) \leq c(g_2(n)$ modify as.

$$\boxed{t_2(n) \leq c_2\, g_2(n)}$$ $c_2 \rightarrow$ some constant

$c_3 = \max\{c_1, c_2\}$

$$\begin{aligned} t_1(n) + t_2(n) &\leq c_1 g_1(n) + c_2 g_2(n) \\ &= c_3 g_1(n) + c_3 g_2(n) \\ &= c_3\{g_1(n) + g_2(n)\} \\ &\leq 2 c_3 \max\{g_1(n), g_2(n)\} \end{aligned}$$

$\therefore t_1(n) + t_2(n) \in O[\max(g_1(n), g_2(n))$

Thus Proved.

(2.) Find the Time complexity of the below recurrence equation:

(3) $T(n) = \begin{cases} 2T(n/2) + 1 & \text{if } n > 1 \\ 1 & \text{otherwise} \end{cases}$

$$T(n) = 2T(n/2) + 2^0 \longrightarrow (1)$$

$$T(n/2) = 2T\left(\frac{n/2}{2}\right) + 1$$

$$T(n/2) = 2T(n/2^2) + 1 \longrightarrow (2)$$

$$T(n/2^2) = 2T\left(\frac{n/2^2}{2}\right) + 1$$

$$= 2T(n/2^3) + 1$$

$$T(n/2^2) = 2T(n/2^3) + 1 \longrightarrow (3)$$

Substitute (2) in (1)

$$T(n) = 2\left[2T(n/2^2) + 1\right] + 1$$

$$= 4T(n/2^2) + 2 + 1$$

$$T(n) = 2^2 T(n/2^2) + 2^1 + 2^0 \longrightarrow (4)$$

Substitute (4) in (3)

$$T(n) = 4\left(2T(n/2^3) + 1\right] + 3$$

$$T(n) = 2^3 T(n/2^3) + 2^2 + 2^1 + 2^0 + 2$$

$$T(n) = \left[2\left(\frac{n}{2}\right)+1\right] + \left[2^2\left(\frac{n}{2^2}\right)+3\right] + \left[2^3\left(\frac{n}{2^3}\right)+7\right] + \ldots$$

$$T(n) = 2^k + \left(\frac{n}{2^k}\right) + \left(2^{k-1}+2^{k-2}\right) + 2^3+2^2+2^1+2^0$$

$$T(n) = 2^k T\left(\frac{n}{2^k}\right) + \left(2^k+2^{k-1}\right)$$

$\frac{n}{2^k} = 1$

$n = 2^k$

$\log n = \log 2^k$

$\boxed{\log n = k}$

$$= 2^k T(1) + \left(2^k+2^{k-1}\right)$$

$$= nT(1) + n + \left(2^{k-1}\right)$$

$$= 2n + 2^{k-1} = 2n + \frac{2^k}{2^1}$$

$$= -\frac{1}{2}(2n+2k) = \frac{1}{2}(3n)$$

$\Rightarrow \boxed{O(n)}$

(4) $T(n) = \begin{cases} 2T(n-1) & \text{if } n>0 \\ 1 & \text{otherwise} \end{cases}$

$\boxed{T(n) = 2T(n-1)} \rightarrow (1)$

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

$\boxed{T(n-1) = 2T(n-2)} \rightarrow (2)$

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

$\boxed{T(n-2) = 2T(n-3)} \rightarrow (3)$

Substitute (2) in (1)

$T(n) = 2[2T(n-2)]$

$\boxed{T(n) = 2^2 T(n-2)} \rightarrow (4)$

substitute (3) in (4)

$T(n) = 2^2[2T(n-3)]$

$\boxed{T(n) = 2^3 T(n-3)}$

$T(n) = 2(n-1) + 2^2(n-2) + 2^3(n-3) + \ldots\ldots$

$$T(n) = 2^k T(n-k)$$

$n-k = 1$

$n = k$

$T(n) = 2^k T(n-k)$

$= 2^n (1)$

$\Rightarrow O(2^n)$

(5) Big O Notation : Show that $f(n) = n^2 + 3n + 5$ is $O(n^2)$

$f(n) = n^2 + 3n + 5$, $g(n) = n^2$

$$f(n) \leq c\, g(n)$$

$$n^2 + 3n + 5 \leq c_1 n^2 \longrightarrow (1)$$

divide L.S by n^2

$$\frac{n^2}{n^2} + \frac{3n}{n^2} + \frac{5}{n^2} \leq c_1 \frac{n^2}{n^2}$$

$$1 + \frac{3}{n} + \frac{5}{n^2} \leq c_1$$

when $n = 1$,

$1 + 3 + 5 \leq c_1$

$c_1 \geq 9$

Sub $c_1 \geq 9$ in eq (1)

$n^2 + 3n + 5 \leq 9n^2$

when $n = 1$,

$1 + 3 + 5 \leq 9$

$9 \leq 9$

$\therefore f(n) = n^2 + 3n + 5$ is $O(n^2)$

$\therefore$ Thus Proved.

$c_1 = 6$

$n_0 = 1$

$\therefore n \geq 1$ we prove that $n^2 + 3n + 5$ becomes $O(n^2)$.

(6) Big omega notation : Prove that $g(n) = n^3 + 2n^2 + 4n$ is $\Omega(n^3)$

$g(n) = n^3 + 2n^2 + 4n$

rewrite it as

$f(n) = n^3 + 2n^2 + 4n$, $g(n) = n^3$

$f(n) \leq c.g(n)$

$n^3 + 2n^2 + 4n \geq cn^3$ → (1)

divide both sides by n^3

$$\frac{n^3}{n^3} + \frac{2n^2}{n^3} + \frac{4n}{n^3} \geq \frac{c.n^3}{n^3}$$

$$1 + \frac{2}{n} + \frac{4}{n^2} \geq c$$

when $n=1$,

$$1 + \frac{2}{1} + \frac{4}{1} \geq c$$

$$7 \geq c$$

Sub $7 \geq c$ in eqn (1)

$$n^3 + 2n^2 + 4n \geq 7n^3$$

when $n=1$

$$1 + 2 + 4 \geq 7$$

$$7 \geq 7$$

$\therefore c \geq 7, n_0 = 1$

$\therefore n \geq 1$, we prove that $n^3 + 2n^2 + 4$ is $\Omega(n^3)$

(7) Big Θ Notation : Determine whether $f(n) = 4n^2 + 3n$ is $\Theta(n^2)$ or not

rewrite it as,

$f(n) = 4n^2 + 3n$, $g(n) = n^2$

$f(n) \leq c_1 g(n)$

$4n^2 + 3n \leq c_1 n^2$ → (1)

div R.H.S by n^2

$\frac{4n^2}{n^2} + \frac{3n}{n^2} \leq \frac{c_1 n^2}{n^2}$

$4 + \frac{3}{n} \leq c_1$

when $n = 1$

$4 + 3 \leq c_1$

$7 \leq c_1$

Sub in eq (1)

$4n^2 + 3n \leq 7n^2$

when $n = 1$

$4 + 3 \leq 7(1)$

$7 \leq 7$

$c_1 \geq 7$, $n_0 \geq 1$

$c_1 \geq 7$, $n_0 \geq 1$

$f(n) \leq c_2 g(n)$

$4n^2 + 3n \geq c_2 n^2$ → (1)

div R.H.S by n^2

$\frac{4n^2}{n^2} + \frac{3n}{n^2} \geq \frac{c_2 n^2}{n^2}$

$4 + \frac{3}{n} \geq c_2$

when $n = 1$

$4 + 3 \geq c_2$

$7 \geq c_2$

so, $c_2 = 1$

sub in eq (2)

$4n^2 + 3n \geq (1) n^2$

when $n = 1$

$4 + 3 \geq 1$

$7 \geq 1$

$c_2 = 1$, $n_0 \geq 1$

$c_2 (g(n)) \leq f(n) \leq c_1 g(n)$

∴ The condition is true

(8) Let $f(n) = n^3 - 2n^2 + n$ and $g(n) = n^2$ show whether $f(n) = \Omega(g(n))$ is true or false justify your answer :

$$f(n) \geq c \cdot g(n)$$

$$n^3 - 2n^2 + n \geq cn^2 \rightarrow (1)$$

div both sides by n^2

$$\frac{n^3}{n^2} - \frac{2n^2}{n^2} + \frac{n}{n^2} \geq \frac{c \cdot n^2}{n^2}$$

$$n - 2 + \frac{1}{n} \geq c$$

when $n = 1$,

$$1 - 2 + 1 \geq c$$

$$0 \geq c$$

when $n = 2$,

$$2 - 2 + \frac{1}{2} \geq c$$

$$0.5 \geq c$$

substitute (1)

$$n^3 - 2n^2 + n \geq (0.5)\, n^2$$

when $n = 1$

$$1 - 2 + 1 \geq 0.5$$

$$0 \geq 0.5$$

when $n = 2$,

$$8 - 8 + 2 \geq (0.5)4$$

$$2 \geq 2$$

$c \leq 0.5$, $n_0 = 2$, $n \geq 2$ ∴ It is true.

(9.) Determine whether $h(n) = n\log n + n$ is in $O(n\log n)$ prove a rigorous proof.

1) upper Bound :

we need to find c_1 and n_0 such that

$h(n) \leq c_1 \cdot n\log n$ for all $n \geq n_0$.

$$h(n) = n\log n + n$$
$$\leq n\log n + n\log n$$
$$= 2n\log n$$

when $c_1 = 2$

then $h(n) \leq 2n\log n$ for all $n \geq 1$

So, $h(n)$ is $O(n\log n)$

2, Lower Bound.

we need to find c_2 and n_0 such that.

$h(n) \geq c_2 \cdot n\log n$ for all $n \geq n_0$.

$$h(n) = n\log n + n$$
$$\geq \tfrac{1}{2} \cdot n\log n \text{ for } n \geq 2.$$

Let $c_2 = \tfrac{1}{2}$ $h(n) \geq \tfrac{1}{2} n\log n$

$h(n)$ is $\Omega(n\log n)$

combining,

$h(n) = n\log n + n$ is in $\Theta(n\log n)$

(10) Solve the following recurrence relation.

$$T(n) = 4T(n/2) + n^2, \quad T(1) = 1.$$

$$T(n) = 2^2 T(n/2) + n^2 \rightarrow (1)$$

$$T(n/2) = 4T(n/2^2) + T(n/2)^2$$

$$\boxed{T(n/2) = 2^2 T(n/2^2) + (n/2)^2} \rightarrow (2)$$

$$T(n/2^2) = 2^2 T(n/2^3) + (n/2^2)^2$$

$$\boxed{T(n/2^2) = 2^2 T(n/2^3) + (n/2^2)^2} \rightarrow (3)$$

$$T(n) = 2^n \left[2^2 T(n/2^2) + (n/2)^2 \right] + n^2$$

$$= 2^4 T(n/2^2) + 2^2 (n/2)^2 + n^2$$

$$= 2^4 T(n/2^2) + n^2$$

$$\boxed{T(n) = 2^4 T(n/2^2) + 2n^2} \rightarrow (4)$$

$$T(n) = 2^6 T(n/2^3) + 3n^2$$

$$T(n) = 2^{k+2} T(n/2^k) + kn^2$$

$\frac{n}{2^k} = 1$

$n = 2^k$

$\log n = \log_2 2^k$

$\boxed{\log n = k}$

$$T(n) = 2^{k+2} T(n/2^2) + kn^2$$

$$= 2^{k+2} T(1) + kn^2$$

$$= 2^k 2^2 (1) + (\log n) n^2$$

$$= 4n + n^2 \log n$$

$$= \Theta(n \log n)$$

$\Theta(n \log n)$.

(11) array of [4, -2, 5, 3, 10, -5, 2, 8, -3, 6, 7, -4, 1, 9, -1, 0, -6, -8, 11, -9] find the max and min product that can be obtained by multiplying 2 int from array.

Solution:

```
def find_m(arr):
    if len(arr) < 2:
        raise value Error("Array atleast contain 2 elements")

arr.sort()
max_product = max(arr[-1] * arr[-2], arr[0] * arr[1])
min_product = min(arr[7] * arr[0], arr[0] * arr[1])
return max_product, min_product.

arr = [4, -2, 5, 3, 10, -5, 2, 8, -3, 6, 7, -4, 1, 9, -1, 0, -6, -8, 11,
-9]
max_product, min_product = find_m(arr)
print(find_m)
```

(12) Demonstrate Binary search method to search key=23 from the arr = [2, 5, 8, 12, 16, 23, 38, 56, 72, 91].

→ Start with 2 pointers 'low' and 'high'

low = 0, high = 9, key = 23

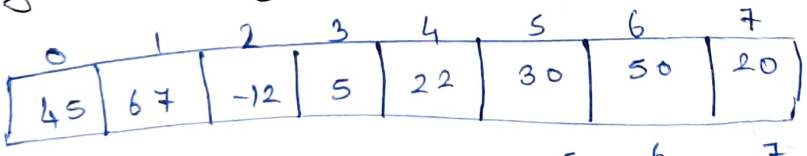
$$mid = \frac{(low + high)}{2} = \frac{0+9}{2} = 4.$$

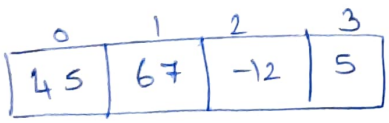
arr[4] = 16.

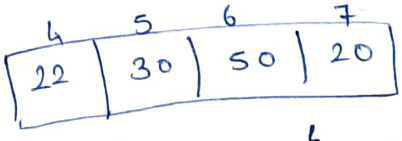
Code :

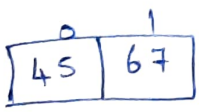
```
def binary_search (arr, key)

    low = 0
    high = len(arr) - 1
while (low <= high) :
    mid = (low + high)//2
        if arr [mid] == key :
            return mid
        if arr [mid] < key :
            low = mid + 1

        else :
            high = mid - 1

    return -1
arr = [2, 5, 8, 12, 16, 23, 38, 56, 72, 91]

key = 23
index = binary_search (arr, key)
print ("key", key, "found at index", index).
```

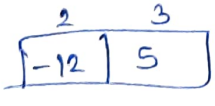
(13) merge sort : (45, 67, -12, 5, 22, 30, 50, 20).

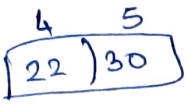


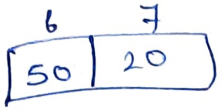


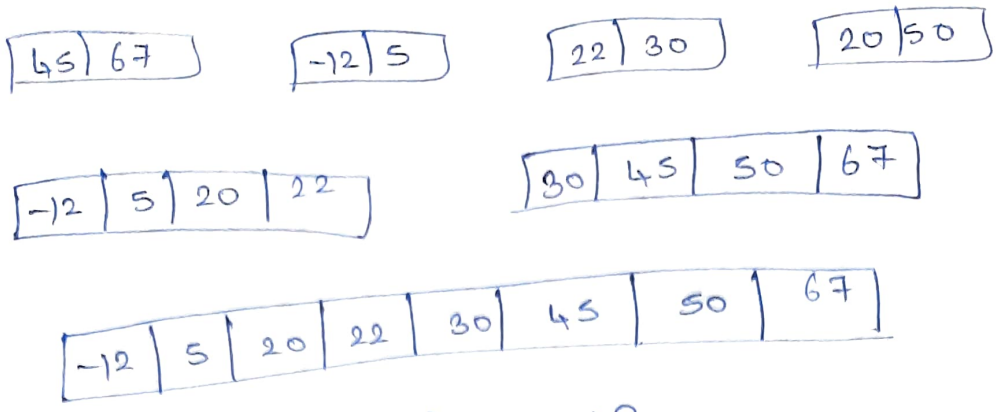












$\Rightarrow T(n) = 2T(n/2) + (n-1)$

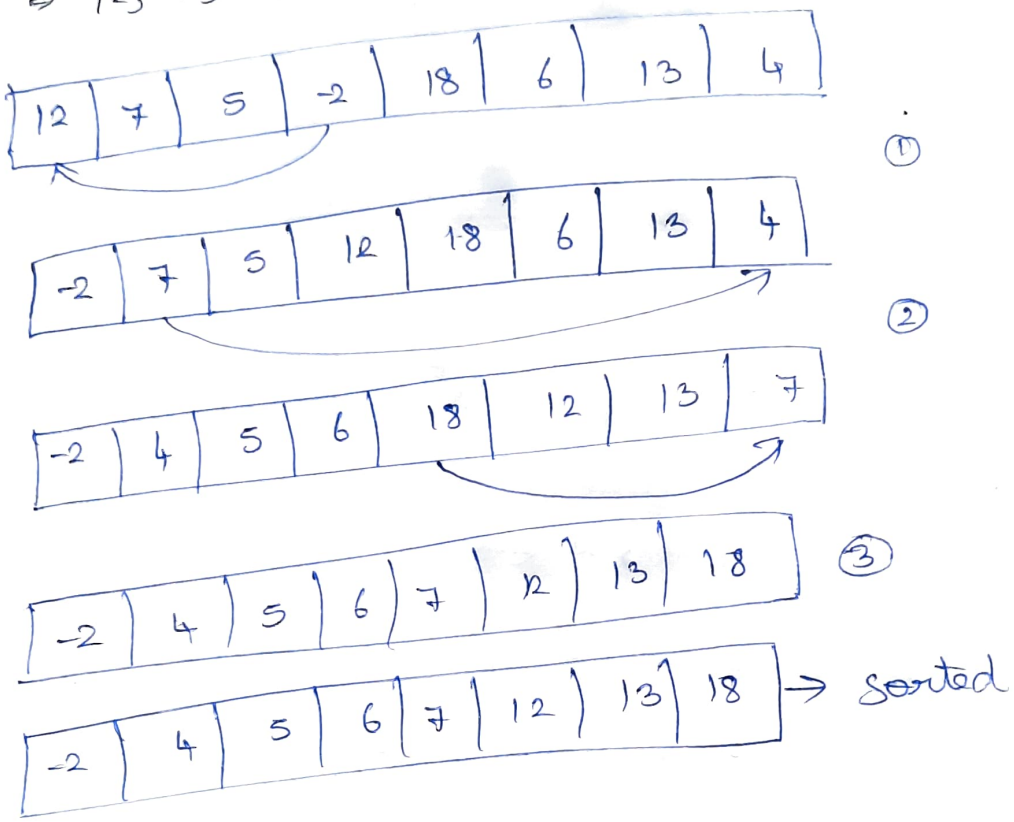
Code:

```
def merge_sort(arr):
    if len(arr) > 1:
        mid = len(arr) // 2
        left_half = arr[:mid]
        right_half = arr[mid:]
        merge_sort(left_half)
        merge_sort(right_half)
        i = j = k = 0
while i < len[left_half] and j < len[right_half]:
    if left_half[i] < right_half[j]:
        arr[k] = left_half[i]
        i += 1
    else:
        arr[k] = right_half[j]
        j += 1
        k += 1
while i < len(left_half):
    arr[k] = left_half
    i += 1, k += 1
```

```
while j < len(right_half):
    arr[k] = right_half[j]
    j += 1
    k += 1
```

(14) No. of times to perform selection sort and estimate time complexity.

⇒ 12, 7, 5, -2, 18, 6, 13, 4.



⇒ n-1 ⇒ 8-1 = 7 → swap.

⇒ Time complexity = $O(n^2)$

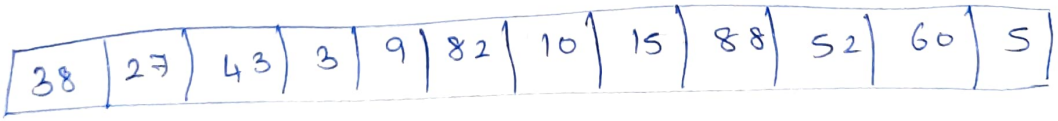
(15) Find the index of the target value 10 using binary search from the following list of elements

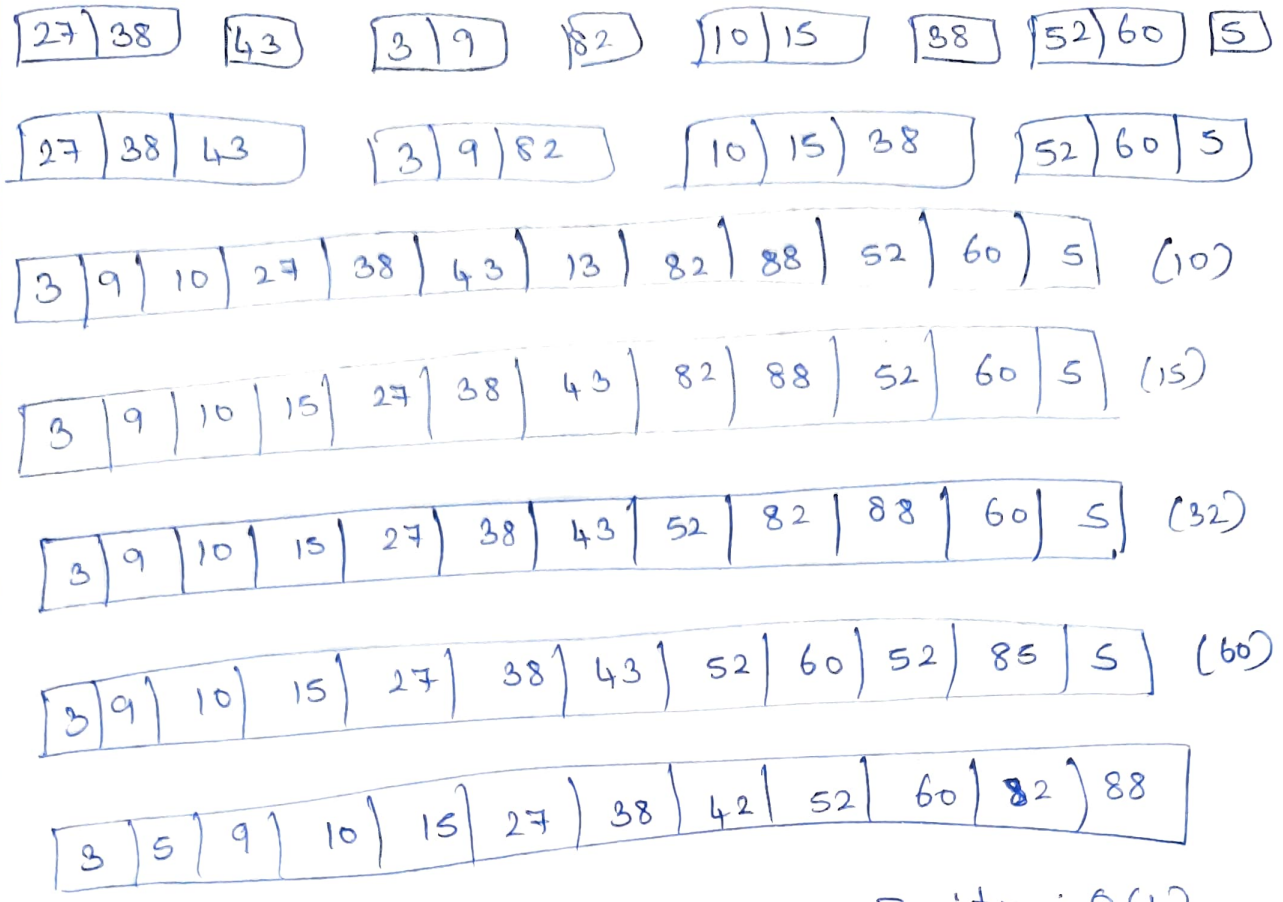
[2, 4, 6, 8, 10, 2, 14, 16, 18, 20].

```
def Binary Search (arr, target)
    low = 0
    high = len (arr) - 1
    while low <= high
    mid = (low + high) // 2
    if arr [mid] == target
        return mid.
    elif arr [mid] < target:
        low = mid + 1
    else:
        high = mid - 1
    return -1
arr = [2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
target = 10
index = Binary search (arr, target)
```

(16) Merge sort, divide and conquer strategy

[38, 27, 43, 3, 9, 82, 10, 15, 88, 52, 60, 5]





Time complexity : $O(n^2)$, space complexity : $O(1)$

(14) Sort the array 64, 34, 25, 12, 22, 11, 90 using bubble sort. what is the time complexity.

1st pass ⇒ [34, 64, 25, 12, 22, 11, 90]

[34, 25, 64, 12, 22, 11, 90]

[34, 25, 12, 64, 22, 11, 90]

[34, 25, 12, 22, 11, 64, 90]

2nd pass ⇒ [25, 34, 12, 22, 11, 64, 90]

[25, 12, 34, 22, 11, 64, 90]

[25, 12, 22, 34, 11, 64, 90]

[25, 12, 22, 11, 34, 64, 90]

3rd pass ⇒ [12, 25, 22, 11, 34, 64, 90]

[12, 22, 25, 11, 34, 64, 90]

[12, 22, 11, 25, 34, 64, 90]

4th pass ⇒ [12, 11, 22, 25, 34, 64, 90]

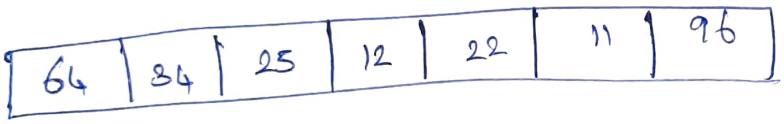
5th pass ⇒ [11, 12, 22, 25, 34, 64, 90]

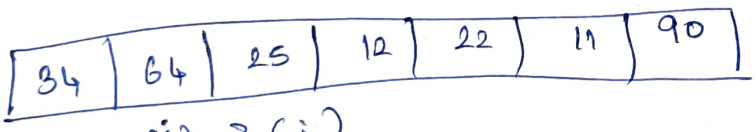
Best case : $O(n^2)$

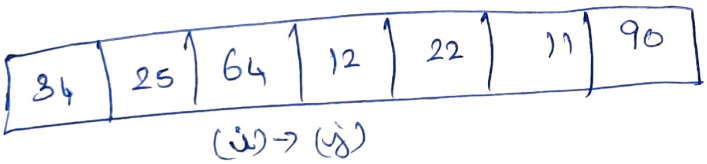
worst case : $O(n^2)$

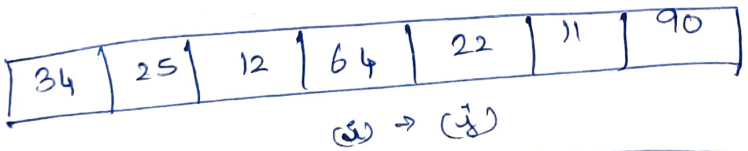
Average case : $O(n^2)$

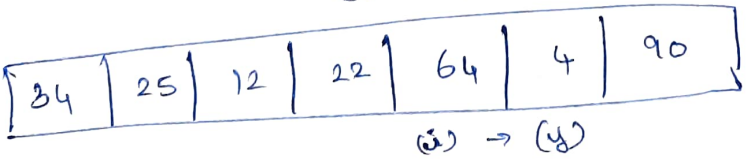
(18) Sort the array 64, 37, 25, 12, 22, 11, 90 using Bubble sort. what is the time complexity of solution.

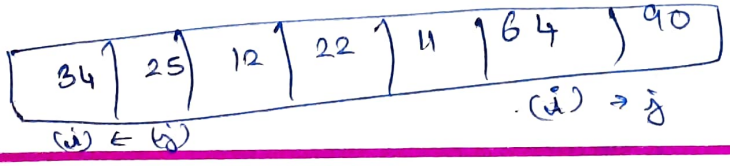


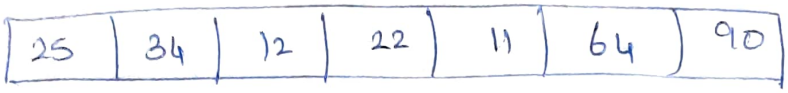


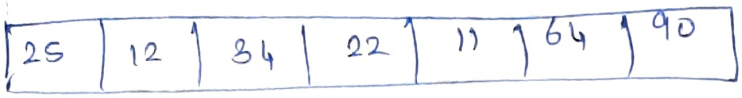


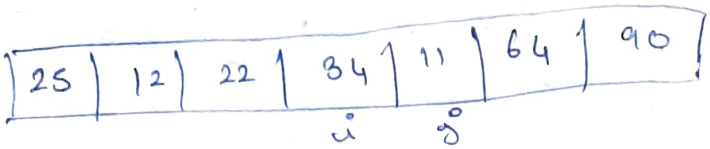


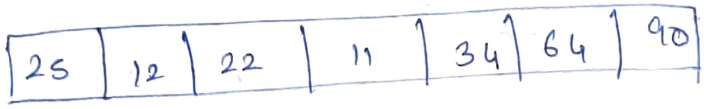


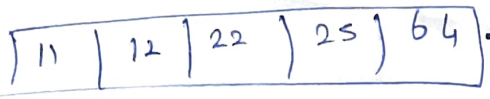












→ sorted.

Time complexity:

worst case : $O(n^2)$

$n \times (n-1)/2$

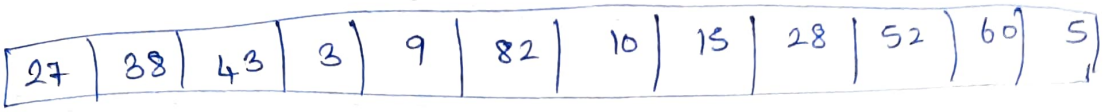
Best case : $O(n^2)$

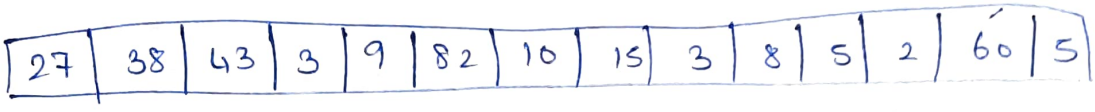
$n(n-1)/2$

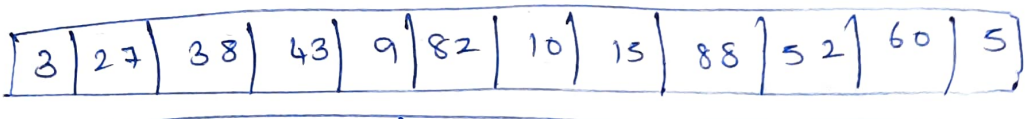
Average case : $O(n^2)$

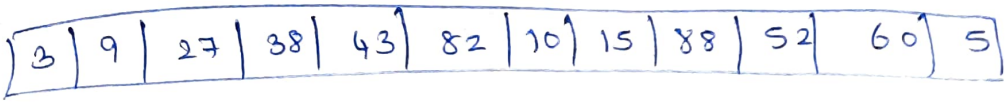
$O(n^2)$.

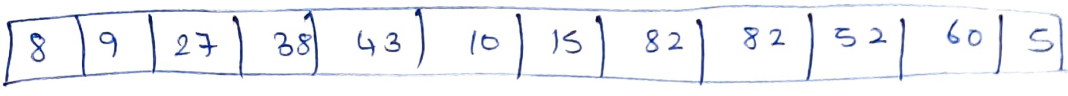
(19) Sort the array using insertion Sort using Brute Force approach [38, 27, 43, 3, 9, 82, 10, 15, 88, 52, 60, 5].

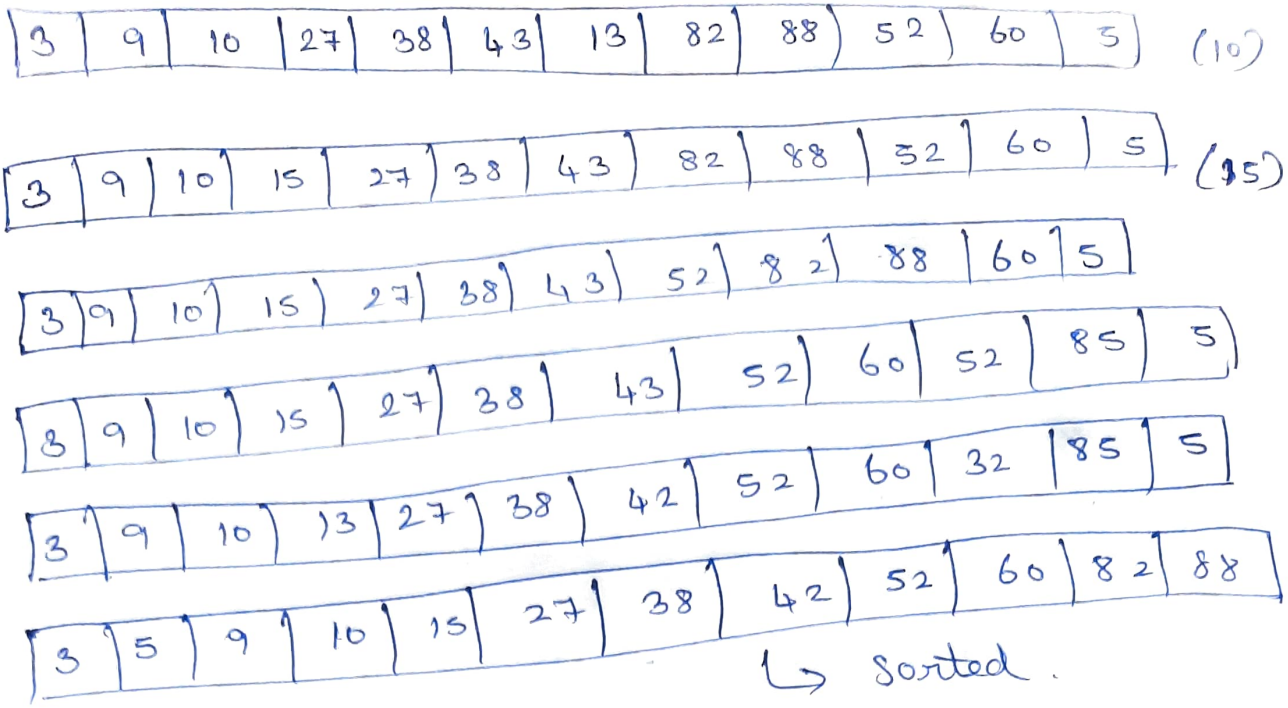












Time complexity : $O(n^2)$

Space complexity : $O(1)$.

(20) Given an arr sort using insertion sort.

[4, -2, 5, 3, 10, -5, 2, 8, -3, 6, 7, -4, 1, 9, -1, 0, -6, -8, 11, -9]

[-2, 4, 5, 3, 10, -5, 2, 8, -3, 6, 7, -4, 1, 9, -1, 0, -6, -8, 11, -9]

[-2, 3, 4, 5, 10, -5, 2, 8, -3, 6, 7, -4, 1, 9, -1, 0, -6, -8, 11, -9]

[-5, -2, 3, 4, 5, 10, 2, 8, -3, 6, 7, -4, 1, 9, -1, 0, -6, -8, 11, -9]

[-5, -3, -2, 2, 3, 4, 5, 8, 10, 6, 7, -4, 1, 9, -1, 0, -6, -8, 11, -9]

[-5, -3, -2, 2, 3, 4, 5, 6, 7, 8, 10, -4, 1, 9, -1, 0, -6, -8, 11, -9]

[-5, -4, -3, -2, 2, 3, 4, 5, 6, 7, 8, 9, 10, -1, 0, -6, -8, 11, -9]

[-5, -4, -3, -2, -1, 0, 2, 3, 4, 5, 6, 7, 8, 9, 10, -6, -8, -11, -9]

[-8, -6, -5, -4, -3, -2, -1, 0, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, -9]

[-9, -8, -6, -5, -4, -3, -2, -1, 0, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]

↳ [Sorted Array]

Time complexity ⇒ $O(n^2)$

Space complexity ⇒ $O(1)$.