Exp. 1B

arr = [64, 25, 12, 22, 11]

1.) Insertn Sort

i = 1, j = i - 1 = 0 ⟹ 64 > 25 ⟹ shift 64 to right & place 25 at 0.

arr = 25, 64, 12, 22, 11

i = 2, j = 1 ⟹ 64 > 12 ⟹ shift 64 right

arr = 25, 12, 64, 22, 11

compare 25 & 12 ⟹ 25 > 12 ⟹ shift 25 right

place 12 at 0

⟹ arr = 12, 25, 64, 22, 11

$i=3, j=2 \Rightarrow$ $64 > 22 \Rightarrow$ shift 64 right; arr = 12, 25, 22, 64, 11

$25 > 22 \Rightarrow$ " 25 " ; arr = 12, 22, 25, 64, 11

$22 > 12 \Rightarrow$ place 22 at 1

$\therefore$ arr = 12, 22, 25, 64, 11

$i=4, j=3 \Rightarrow$ $64 > 11 \Rightarrow$ shift right

$25 > 11 \Rightarrow$ " "

$22 > 11 \Rightarrow$ " "

$12 > 11 \Rightarrow$ " "

place 11 at 0 $\Rightarrow$ arr = 11, 12, 22, 25, 64

Final arr = 11, 12, 22, 25, 64

2) Selection sort

i=0 : find min. in 0-4 ⇒ 11 @ 4
⇒ swap 11 & arr[i] ⇒ arr = 11, 25, 12, 22, 64

i = 1 : find min. in 1-4 ⇒ 12 @ 2
⇒ swap 12 & arr[i=1] ⇒ arr = 11, 12, 25, 22, 64

i = 2 : find min. in 2-4 ⇒ 22 @ 3
⇒ swap 22 & arr[2] = 25 ⇒ arr = 11, 12, 22, 25, 64

i = 3 : find min. in 3-4 ⇒ 25 @ 3
⇒ already in correct posn ; no swap needed.

* Insertion Sort :
→ builds sorted arr incrementally
→ shifts elements to make space for insertion
→ requires many shifts to place each element.
→ potentially O(n) for nearly sorted arrays.

* Selection Sort :
→ Finds min. in each pass
→ Swaps min. with current unsorted posn
→ fewer actual swaps as compared to shifts in insertion sort.