ANALYSIS OF ALGORITHMS

Purvi Tandel, Kinjal Mistree, Fenil Khatiwala
Department of Computer Engineering, CGPIT, UTU

TOPICS TO BE COVERED

- ✓ Algorithm
- ✓ Efficiency of algorithms
- ✓ Performance analysis of algorithm
- ✓ Elementary operation
- ✓ Asymptotic Notation
- ✓ Analyzing control statements
- Average and worst case analysis
- Solving recurrences

AVERAGE AND WORST CASE ANALYSIS

SORTING ALGORITHMS

BUBBLE SORT

APPLICATIONS OF SORTING

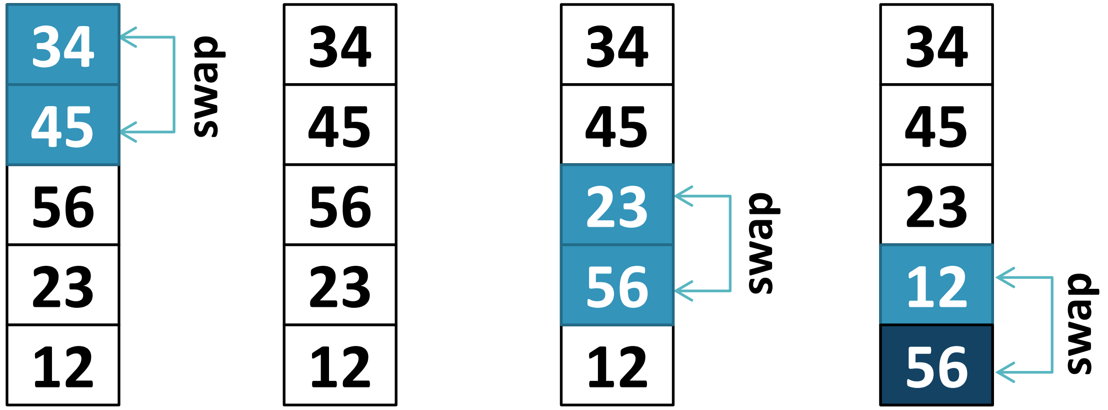
1. **Phone Bill**: the calls made are date wise sorted.
2. **Bank statement or Credit card Bill**: transactions made are date wise sorted.
3. **Filling forms online**: “select country” drop down box will have the name of countries sorted in Alphabetical order.
4. **Online shopping**: the items can be sorted price wise, date wise or relevance wise.
5. **Files or folders:** on your desktop are sorted date wise.

BUBBLE SORT

Sort the following array in Ascending order

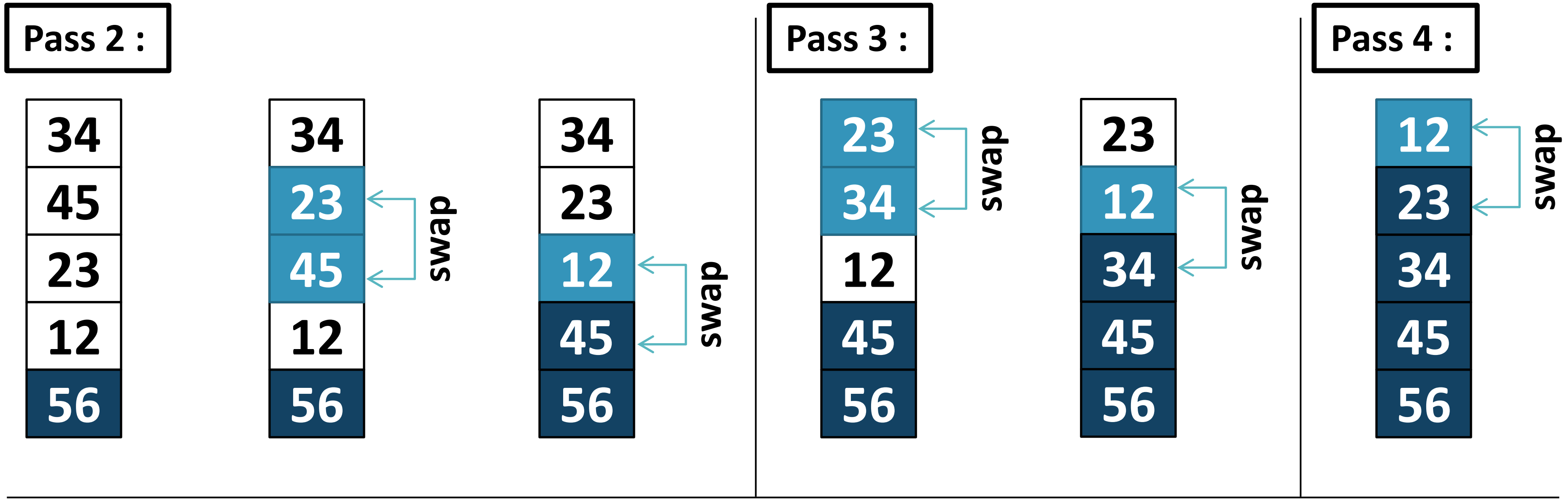
45	34	56	23	12

Pass 1 :




```
if(A[j] > A[j+1])
      swap(A[j],A[j+1])
```

BUBBLE SORT




```
if(A[j] > A[j+1])
        swap(A[j],A[j+1])
```

BUBBLE SORT

- It is a simple sorting algorithm that works by
  - Comparing each pair of **adjacent items** and swapping them if they are in the wrong order.
  - The pass through the list is **repeated** until no swaps are needed, which indicates that the list is sorted.

- As it only uses **comparisons** to operate on elements, it is a comparison sort.

- Although the algorithm is simple, it is **too slow** for practical use.

BUBBLE SORT – ALGORITHM

```
# Input: Array A
# Output: Sorted array A

Algorithm: Bubble_Sort(A)

for i ← 1 to n do
   for j ← 1 to n-i do
     if  A[j] > A[j+1] then
          temp ← A[j]
          A[j] ← A[j+1]
          A[j+1] ← temp
```

BUBBLE SORT: BEST CASE ANALYSIS

```
int flag=1;
for(i = 1; i <= n; i++)
   for(j = 1; j <= n-i; j++)
      if(A[j] > A[j+1])
            flag=0;
            swap(A[j],A[j+1])
   if(flag == 1)
      cout<<"already sorted"<<endl
      break;
```

Best case time complexity = $\Omega(n)$

Condition never becomes true

Pass 1 : i = 1

12	j = 1
23	j = 2
34	j = 3
45	j = 4
59	

Only one iteration takes place from 1 to $n-1$, i.e., Pass 1

BUBBLE SORT – WORST CASE ANALYSIS

```
# Input: Array A
# Output: Sorted array A

Algorithm: Bubble_Sort(A)

for i ← 1 to n do
    for j ← 1 to n-i do
      if  A[j] > A[j+1] then
          temp ← A[j]
          A[j] ← A[j+1]
          A[j+1] ← temp
```

c1 * $(n + 1)$ = O (n)

c2 * n (n – i +1) = O(n^2)

c3 * n (n – i) = O(n^2)

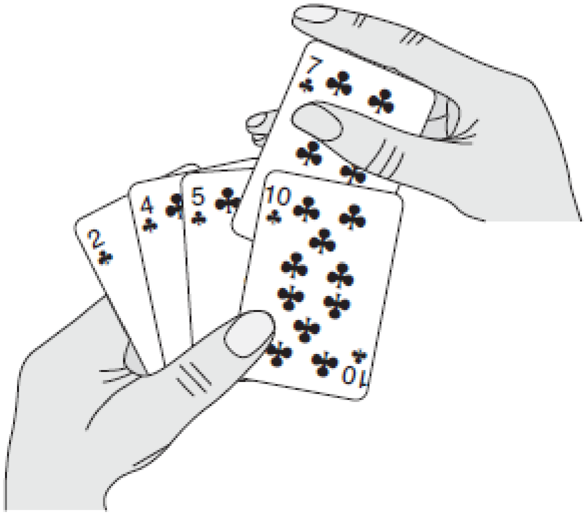
c4 * n (n – i) = O(n^2)

Worst case time complexity = $O\left(n^2\right)$

INSERTION SORT

INSERTION SORT

- Insertion sort works the way many people sort a hand of **playing cards.**
- We start with an empty left hand and the cards face down on the table.
- We then remove one card at a time from the table and insert it into the correct position in the left hand. To find the correct position for a card, we compare it with each of the cards already in the hand, from **right to left.**
- At all times, the cards held in the left hand are sorted, and these cards were originally the top cards of the pile on the table.



INSERTION SORT - EXAMPLE

Sort the following elements in Ascending order.

5	1	12	-5	16	2	12	14

Step 1 :

Unsorted Array

5	1	12	-5	16	2	12	14
1	2	3	4	5	6	7	8

Step 2 :

$\boldsymbol{i = 2, x = 1}$ $\boldsymbol{j = i - 1 \text{ and } j > 0}$

j

5 1	1	12	-5	16	2	12	14
1	**2**	**3**	**4**	**5**	**6**	**7**	**8**

Shift down

while $x < T[j]$ do

$T[j+1] \leftarrow T[j]$

$j--$

INSERTION SORT - EXAMPLE

Step 3 :

$\boldsymbol{i = 3, x = 12}$ | $\boldsymbol{j = i - 1 \ and \ j > 0}$

while $x < T[j]$ do
$T[j+1] \leftarrow T[j]$
$j--$

	j						
1	5	12	-5	16	2	12	14
1	2	3	4	5	6	7	8

No Shift will take place

Step 4 :

$\boldsymbol{i = 4, x = -5}$ | $\boldsymbol{j = i - 1 \ and \ j > 0}$

while $x < T[j]$ do
$T[j+1] \leftarrow T[j]$
$j--$

j		j					
-5 1	5	12	-5	16	2	12	14
1	2	3	4	5	6	7	8

Shift down
Shift down Shift down

INSERTION SORT - EXAMPLE

Step 5 :

$\boldsymbol{i = 5, x = 16}$ | $\boldsymbol{j = i - 1 \; and \; j > 0}$

j

-5	1	5	12	16	2	12	14
1	**2**	**3**	**4**	**5**	**6**	**7**	**8**

while $x < T[j]$ do
$\quad T[j+1] \leftarrow T[j]$
$\quad j--$

No Shift will take place

Step 6 :

$\boldsymbol{i = 6, x = 2}$ | $\boldsymbol{j = i - 1 \; and \; j > 0}$

j *j*

-5	1	5 2	12	16	2	12	14
1	**2**	**3**	**4**	**5**	**6**	**7**	**8**

while $\boxed{x < T[j]}$ do
$\quad T[j+1] \leftarrow T[j]$
$\quad j--$

Shift down Shift Shift down

INSERTION SORT - EXAMPLE

Step 7 :

$i = 7, x = 12$ | $j = i - 1 \text{ and } j > 0$

					j		
-5	1	2	5	12	~~16~~ 12	12	14
1	2	3	4	5	6	7	8

Shift down

while $\boxed{x < T[j]}$ do
$T[j+1] \leftarrow T[j]$
$j--$

Step 8 :

$i = 8, x = 14$ | $j = i - 1 \text{ and } j > 0$

						j	
-5	1	2	5	12	12	~~16~~ 14	14
1	2	3	4	5	6	7	8

Shift down

while $\boxed{x < T[j]}$ do
$T[j+1] \leftarrow T[j]$
$j--$

INSERTION SORT - ALGORITHM

```
# Input: Array T
# Output: Sorted array T
Algorithm: Insertion_Sort(T[1,…,n])
for i ← 2 to n do
      x ← T[i];
      j ← i - 1;
      while x < T[j] and j > 0 do
            T[j+1] ← T[j];
            j ← j - 1;
      T[j+1] ← x;
```

INSERTION SORT - ANALYSIS

```
# Input: Array T
# Output: Sorted array T
Algorithm: Insertion_Sort(T[1,…,n])
for i ← 2 to n do
      x ← T[i];
      j ← i - 1;
      while x < T[j] and j > 0 do
            T[j+1] ← T[j];
            j ← j - 1;
      T[j+1] ← x;
```

$O(n)$

$O(n^2)$

The time complexity of Insertion sort is $\mathbf{O(n^2)}$

INSERTION SORT - ANALYSIS

Algorithm: Insertion_Sort(T)	Cost	Times
for i←2 to n **do**		
x←T[i]		
j←i-1		
while x<T[j] **and** j>0 **do**		
T[j+1] ← T[j]		
j = j - 1		
T[j+1] = x;		

INSERTION SORT – ANALYSIS

- For each value of $i = 1, 2, \ldots.n$,
  T_i is the number of times while loop gets executed for that value of i.

$$T(n) = c_1 n + c_2 (n-1) + c_3 (n-1) + c_4 \sum_{i=2}^{n} T_i + c_5 \sum_{i=2}^{n} (T_i - 1) + c_6 \sum_{i=2}^{n} (T_i - 1) + c_7 (n-1)$$

INSERTION SORT – BEST CASE ANALYSIS

```
# Input: Array T
# Output: Sorted array T
Algorithm: Insertion_Sort(T[1,…,n])
for i ← 2 to n do
    x ← T[i];
    j ← i - 1;
    while x < T[j] and j > 0 do
          T[j+1] ← T[j];
          j ← j - 1;
    T[j+1] ← x;
```

Pass 1 :

12		
23	i=2	T[j]=12
34	i=3	T[j]=23
45	i=4	T[j]=34
59	i=5	T[j]=45

The best case time complexity of Insertion sort is $\Omega(n)$

INSERTION SORT – BEST CASE ANALYSIS

$$T(n) = c_1 n + c_2\,(n-1)\ + c_3\,(n-1) + c_4 \sum_{\mathrm{i=2}}^{\mathrm{n}} T_i + c_5 \sum_{\mathrm{i=2}}^{\mathrm{n}} (T_i - 1) + c_6 \sum_{\mathrm{i=2}}^{\mathrm{n}} (T_i - 1) + c_7 (n-1)$$

$$\sum_{\mathrm{i=2}}^{\mathrm{n}} T_i \quad = \sum_{\mathrm{i=2}}^{\mathrm{n}} 1$$

$$= n - 1$$

Substitute value in total time complexity equation

$$T(n) = c_1 n + c_2\,(n-1)\ + c_3\,(n-1) + c_4\,(n-1) + c_5\,(0) + c_6\,(0) + c_7 (n-1)$$

$$= (c_1 + c_2 + c_3 + c_4 + c_7)\,n - (c_2 + c_3 + c_4 + c_7)$$

$$\boxed{T(n) = \Omega\,(n)}$$

Best case analysis (time complexity)

INSERTION SORT – WORST CASE ANALYSIS

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4 \sum_{i=2}^{n} T_i + \sum_{i=2}^{n} T_i - 1(c_5 + c_6) + c_7(n-1)$$

For worst case, we must compare each element T[i] with each element in the entire sorted subarray T[1……i-1].

$T_i = i$

$$\sum_{i=2}^{n} T_i = \sum_{i=2}^{n} i = \frac{n(n+1)}{2} - 1$$

Substitute value in total time complexity equation

Use arithmetic series,

$\sum_{k=1}^{n} k = \frac{n(n+1)}{2}$

INSERTION SORT – WORST CASE ANALYSIS

If $\sum_{i=2}^{n} T_i = \sum_{i=2}^{n} i = \frac{n(n+1)}{2} - 1$ then $\sum_{i=2}^{n} T_i - 1 = ?$

$$\sum_{i=2}^{n} T_i - 1 = \sum_{i=2}^{n} T_i - \sum_{i=2}^{n} 1$$
$$= \frac{n(n+1)}{2} - 1 - (n-1)$$
$$= \frac{n(n+1)}{2} - n$$
$$= \frac{n^2+n-2n}{2}$$
$$= \frac{n^2-n}{2}$$
$$= \frac{n(n-1)}{2}$$

$$\sum_{i=2}^{n} T_i - 1 = \frac{n(n-1)}{2}$$

Substitute values in total time complexity equation

INSERTION SORT – WORST CASE ANALYSIS

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4 \sum_{i=2}^{n} \left(\frac{n(n+1)}{2} - 1\right) + (c_5 + c_6) \sum_{i=2}^{n} \left(\frac{n(n-1)}{2}\right) + c_7(n-1)$$

$$= c_1 n + c_2 n - c_2 + c_3 n - c_3 + c_4 \left(\frac{n^2 + n - 2}{2}\right) + c_5 \left(\frac{n^2 - n}{2}\right) + c_6 \left(\frac{n^2 - n}{2}\right) + c_7 n - c_7$$

$$= c_1 n + c_2 n - c_2 + c_3 n - c_3 + c_4 \frac{n^2}{2} + c_4 \frac{n}{2} - c_4 + c_5 \frac{n^2}{2} - c_5 \frac{n}{2} + c_6 \frac{n^2}{2} - c_6 \frac{n}{2} + c_7 n - c_7$$

$$= n^2 \underline{\left[\frac{c_4}{2} + \frac{c_5}{2} + \frac{c_6}{2}\right]} + n \underline{\left[c_1 + c_2 + c_3 + \frac{c_4}{2} - \frac{c_5}{2} - \frac{c_6}{2} + c_7\right]} - \underline{[c_2 + c_3 + c_4 + c_7]}$$

$$= an^2 + bn + c$$

The time complexity of Insertion sort algorithm is $\boldsymbol{T(n) = O(n^2)}$

INSERTION SORT – AVERAGE CASE ANALYSIS

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4 \sum_{i=2}^{n} T_i + \sum_{i=2}^{n} (T_i - 1)(c_5 + c_6) + c_7(n-1)$$

For average case, $T_i = \frac{i}{2}$

But we consider $\frac{i}{2}$ as j only, as $\frac{1}{2}$ is constant value itself.

So, average case time complexity $\boldsymbol{T(n)} = \theta(\boldsymbol{n^2})$

INSERTION SORT – WORST CASE EXAMPLE

- Sort given elements in **descending order** using insertion sort: 6, 5, 4, 3, 2, 1

Step 1 :

6	5	4	3	2	1

Step 2 :

5	6	4	3	2	1

Step 3 :

5	4	6	3	2	1

4	5	6	3	2	1

Step 4:

4	5	3	6	2	1

4	3	5	6	2	1

3	4	5	6	2	1

INSERTION SORT – WORST CASE EXAMPLE

- Sort given elements in **descending order** using insertion sort: 6, 5, 4, 3, 2, 1

Step 5 :

3	4	5	2	6	1
3	4	2	5	6	1
3	2	4	5	6	1
2	3	4	6	2	1
2	3	4	5	6	1

Step 6:

2	3	4	5	1	6
2	3	4	1	5	6
2	3	1	4	5	6
2	1	3	4	5	6
1	2	3	4	5	6
1	2	3	4	5	6

INSERTION SORT - EXAMPLE

- Sort given elements in **descending order** using insertion sort: 3, 9, 6, 5, 23, 14

Step 1 :

3	9	6	5	23	14

Step 2 :

9	3	6	5	23	14

Step 3 :

9	6	3	5	23	14

Step 4 :

9	6	5	3	23	14

Step 5:

23	60	5	3	23	14

Step 6 :

23	14	6	5	3	14

SELECTION SORT

SELECTION SORT

Sort the following elements in Ascending order

5	1	12	-5	16	2	12	14

Step 1 :

Unsorted Array

5	1	12	-5	16	2	12	14
1	2	3	4	5	6	7	8

Step 2 :

Unsorted Array (elements 2 to 8)

-5	**1**	**12**	**5**	**16**	**2**	**12**	**14**
1	**2**	**3**	**4**	**5**	**6**	**7**	**8**

Swap

Minj = 1, Minx = 5

Find the smallest value from the entire Unsorted array

Index = 4, value = -5

SELECTION SORT

Step 3 :

Unsorted Array (elements 3 to 8)

-5	1	12	5	16	2	12	14
1	2	3	4	5	6	7	8

Minj = 2, Minx = 1

Find min value from
Remaining unsorted array
Index = 2, value = 1

No Swapping as min value is already at right place

Step 4 :

Unsorted Array (elements 4 to 8)

-5	1	2	5	16	12	12	14
1	2	3	4	5	6	7	8

Swap

Minj = 3, Minx = 12

Find min value from
Remaining unsorted array
Index = 6, value = 2

SELECTION SORT

Step 5 :

Unsorted Array (elements 5 to 8)

-5	1	2	5	16	12	12	14
1	2	3	4	5	6	7	8

Minj = 4, Minx = 5

Find min value from
Remaining unsorted array
Index = 4, value = 5

No Swapping as min value is already at right place

Step 6 :

Unsorted Array (elements 6 to 8)

-5	1	2	5	12	16	12	14
1	2	3	4	5	6	7	8

Swap

Minj = 5, Minx = 16

Find min value from
Remaining unsorted array
Index = 6, value = 12

SELECTION SORT

Step 7 :

Unsorted Array (elements 7 to 8)

-5	1	2	5	12	12	16	14
1	2	3	4	5	6	7	8

Swap

Minj = 6, Minx = 16

Find min value from
Remaining unsorted array
Index = 7, value = 12

Step 8 :

Unsorted Array (element 8)

-5	1	2	5	12	12	14	16
1	2	3	4	5	6	7	8

Swap

Minj = 7, Minx = 16

Find min value from
Remaining unsorted array
Index = 8, value = 14

SELECTION SORT

- Selection sort **divides the array** or list into two parts,
  1. The sorted part at the left end
  2. and the unsorted part at the right end.
- Initially, the sorted part is empty and the unsorted part is the entire list.
- The **smallest element** is selected from the unsorted array and swapped with the leftmost element, and that element becomes a part of the sorted array.
- Then it finds the **second smallest element** and exchanges it with the element in the second leftmost position.
- This **process continues** until the entire array is sorted.

SELECTION SORT - ALGORITHM

```
# Input: Array A
# Output: Sorted array A
Algorithm: Selection_Sort(A)
for i ← 1 to n-1 do
      minj ← i;
      minx ← A[i];
      for j ← i + 1 to n do
            if A[j] < minx then
                  minj ← j;
                  minx ← A[j];
      A[minj] ← A[i];
      A[i] ← minx;
```

SELECTION SORT - EXAMPLE

```
Algorithm: Selection_Sort(A)
for i ← 1 to n-1 do
    minj ← i; minx ← A[i];
    for j ← i + 1 to n do
        if A[j] < minx then
            minj ← j ; minx ← A[j];
    A[minj] ← A[i];
    A[i] ← minx;
```

Pass 1 :

i = 1

minj ← 2

minx ← 34

j = 2 3

No Change

Sort in Ascending order

45	34	56	23	12
1	2	3	4	5

SELECTION SORT EXAMPLE

```
Algorithm: Selection_Sort(A)
for i ← 1 to n-1 do
    minj ← i ; minx ← A[i];
    for j ← i + 1 to n do
        if A[j] < minx then
            minj ← j ; minx ← A[j];
    A[minj] ← A[i];
    A[i] ← minx;
```

Pass 1 :

i = 1

minj ← 5

minx ← 12

j = 2 3 4 5

Sort in Ascending order

45	34	56	23	12
1	2	3	4	5

Unsorted Array

12	34	56	23	45
1	2	3	4	5

12	23	34	45	56

SELECTION SORT - ALGORITHM

```
# Input: Array A
# Output: Sorted array A
Algorithm: Selection_Sort(A)
for i ← 1 to n-1 do
      minj ← i;
      minx ← A[i];
      for j ← i + 1 to n do
            if A[j] < minx then
                  minj ← j;
                  minx ← A[j];
      A[minj] ← A[i];
      A[i] ← minx;
```

SELECTION SORT – BEST CASE ANALYSIS

```
# Input: Array A
# Output: Sorted array A
Algorithm: Selection_Sort(A)
for i ← 1 to n-1 do
      minj ← i;
      minx ← A[i];
      for j ← i + 1 to n do
            if A[j] < minx then
                  minj ← j;
                  minx ← A[j];
      A[minj] ← A[i];
      A[i] ← minx;
```

c1 * (n) = $\Omega(n)$

c2 * n(2) = $\Omega(n)$

c3 * n (n – 1 +1) = $\Omega(n^2)$

Condition never becomes true

c4 * n(2) = $\Omega(n)$

Pass:

12
23
34
45
59

i = 1,j = 2
i = 2,j = 3
i = 3,j = 4
i = 4,j = 5

minj = 1,minx = 12
minj = 2,minx = 23
minj = 3,minx = 34
minj = 4,minx = 45

The time complexity of Selection sort is $\Omega(n^2)$

SELECTION SORT - ANALYSIS

Algorithm: Selection_Sort(A)	Cost	Times
for i ← 1 to n-1 **do**		
minj ← i ; minx ← A[i];		
for j ← i + 1 to n **do**		
if A[j] < minx **then**		
minj ← j ; minx = A[j]		
A[minj] = A[i];		
A[i] = minx;		

SELECTION SORT – WORST ANALYSIS

$T(n)$

$$= c_1 n + c_2(n-1) + c_3\left(\sum_{i=1}^{n-1}(n \; - i + 1)\right) + c_4\left(\sum_{i=1}^{n-1} n \; - i\right) + c_5\left(\sum_{i=1}^{n-1} n \; - i\right) + c_6(n-1) + c_7(n-1)$$

$$\sum_{i=1}^{n-1}(n \; - i + 1) = \sum_{i=2}^{n-1} n - \sum_{i=1}^{n-1} i + \sum_{i=1}^{n-1} 1$$

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

$$= n^2 - \; n \; - \frac{n^2 + n}{2} + (n-1)$$

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

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

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

$$\sum_{i=1}^{n-1}(n \; - i) = \sum_{i=2}^{n-1} n - \sum_{i=1}^{n-1} i$$

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

$$= n^2 - \; n \; - \frac{n^2 + n}{2}$$

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

$$= \frac{n^2 - n}{2}$$

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

SELECTION SORT – WORST ANALYSIS

$T(n)$

$$= c_1 n + c_2(n-1) + c_3\left(\frac{n(n+1)}{2} \; - \; 1\right) + c_4\left(\frac{n(n-1)}{2}\right) + c_5\left(\frac{n(n-1)}{2}\right) + c_6(n-1)$$

$$+ c_7(n-1)$$

$$= c_1 n + c_2(n-1) + c_3\left(\frac{n^2 + \; n \; - \; 2}{2}\right) + c_4\left(\frac{n^2 - \; n}{2}\right) + c_5\left(\frac{n^2 - \; n}{2}\right) + c_6(n-1) + c_7(n-1)$$

$$= \; n^2\left[\frac{c_3}{2} + \frac{c_4}{2} + \frac{c_5}{2}\right] + n\left[c_1 + c_2 + \frac{c_3}{2} - \frac{c_4}{2} - \frac{c_5}{2} + c_6 + c_7\right] - [c_2 + c_3 + c_6 + c_7]$$

$$= an^2 + \; bn \; + \; c$$

The time complexity of Selection sort algorithm is $\boxed{\boldsymbol{T(n) = O(n^2)}}$

SORTING ALGO.'S TIME COMPLEXITY SUMMARY

Algorithm	Best Case	Average Case	Worst Case
Bubble Sort	O(n)	$O(n^2)$	$O(n^2)$
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Insertion Sort	O(n)	$O(n^2)$	$O(n^2)$