

①

Bubble sort

A →

8	5	7	3	2
---	---	---	---	---

Ist Pass ⇒

8	5	5	5	5
5	8	7	7	7
2	7	8	3	3
3	3	3	8	2
2	2	2	2	8

IInd Pass ⇒

5	5	5	5
7	7	3	3
3	3	7	2
2	2	2	7
8	8	8	8

~~Similar~~

~~One pass = Highest~~

One pass — Greatest no. sorted
 2 passes — 2 greatest no.s sorted

n passes — n numbers sorted.

n elements — (n-1) passes

Q

n elements

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

comparisons

$$O(n^2)$$

void BubbleSort(int A[], int n)

{
for (i=0; i < n-1; i++)

{
for (j=0; j < n-1; j++)

{
if (A[j] > A[j+1])

{
swap (A[j], A[j+1]);

}

}

}

}

Bubble Sort time

Min - $O(n)$

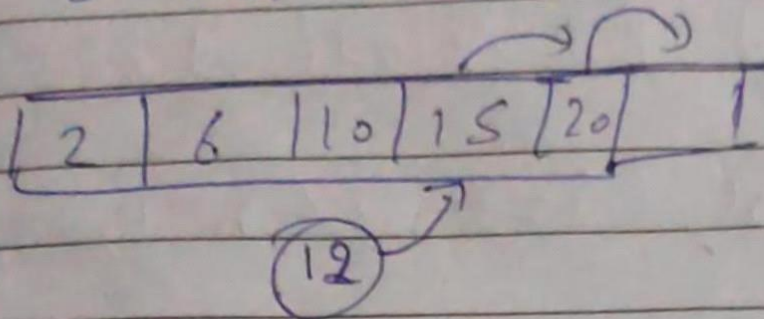
Max - $O(n^2)$

Insertion sort

A \Rightarrow

2	6	10	15	20
---	---	----	----	----

element = 12



A \Rightarrow

8	5	7	3	2
---	---	---	---	---

1st pass \Rightarrow

8	7	3	2
---	---	---	---

The diagram shows the first pass of the insertion sort. The array is [8, 5, 7, 3, 2]. A vertical line is placed between 8 and 7. An arrow points from 8 to 5, and another arrow points from 5 to 7, indicating a swap.

5	8	7	3	2
---	---	---	---	---

The diagram shows the array after the first pass: [5, 8, 7, 3, 2]. A vertical line is placed between 8 and 7.

2nd pass \Rightarrow

5	8	3	2
---	---	---	---

The diagram shows the second pass of the insertion sort. The array is [5, 8, 7, 3, 2]. A vertical line is placed between 8 and 7. An arrow points from 8 to 7, and another arrow points from 7 to 3, indicating a swap.

5	7	8	3	2
---	---	---	---	---

The diagram shows the array after the second pass: [5, 7, 8, 3, 2]. A vertical line is placed between 8 and 3.

3rd pass \Rightarrow

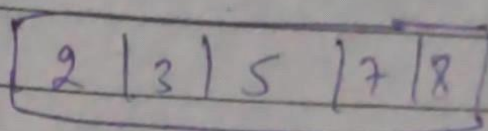
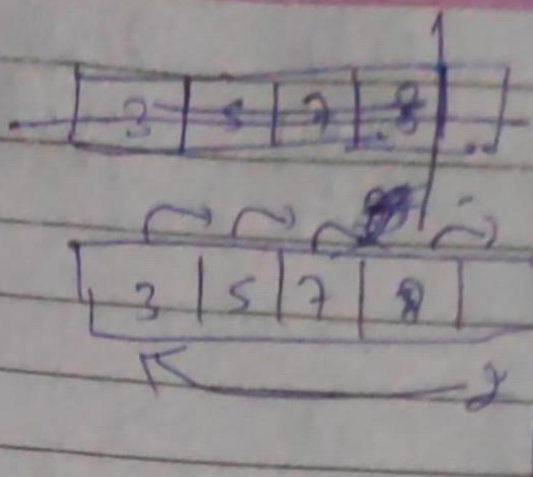
5	7	8	2
---	---	---	---

The diagram shows the third pass of the insertion sort. The array is [5, 7, 8, 3, 2]. A vertical line is placed between 8 and 3. An arrow points from 8 to 7, and another arrow points from 7 to 5, indicating a swap.

3	5	7	8	2
---	---	---	---	---

The diagram shows the array after the third pass: [3, 5, 7, 8, 2]. A vertical line is placed between 8 and 2.

4th pass



No. of passes $\Rightarrow n - 1$

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

void insertionSort(int A[], int n)

```
{
    for (i = 1; i < n; i++)
    {
        j = i - 1;
        n = A[i];
```

```
        while (A[j] > n && j > -1)
        {
```

```
            A[j+1] = A[j];
```

```
            j = j - 1;
```

```
        }
```

```
        A[j+1] = n;
```

```
    }
```

```
}
```


	Min	Max
Time	$O(n)$	$O(n^2)$
Space	$O(1)$	$O(n^2)$

Quick Sort

Pivot

(50) 20 60 90 40 80 10 20 30 -
i ————— j

(50) 30 60 90 40 80 10 20 70
i ————— j

(50) 30 20 90 40 80 10 60 70
i ————— j

(50) 30 20 10 40 80 90 60 70
j — i

(40 30 20 10) (50) (80 90 60 70)
↓
Partition

int Partition (int A[], int L, int R)

{

{

int pivot = A[R];

int i = L, j = R;

do

{

do { i++;

do { j--;

if (i < j)

swap (A[i], A[j])

}

while (i < j);

swap (A[L], A[j]);

return j;

}

Best case \Rightarrow

Time

$O(n \log n)$

Worst case \Rightarrow

time

$O(n^2)$

A

Merge Sort

A

[8 | 3 | 7 | 4 | 9 | 2 | 6 | 5]

0 1 2 3 4 5 6 7

[8 | 3 | 7 | 4 | 9 | 2 | 6 | 5]

\swarrow \swarrow \swarrow \swarrow
3 8 4 7 2 9 5 6

\swarrow \swarrow
3 4 7 8 2 5 6 9

\swarrow \swarrow
2 3 4 5 6 7 8 9

void MergeSort (int A[], int n)

{

int p; i, l, mid, h;

for (p = 2; p < n; p = p * 2)

{

for (i = 0; i + p - 1 < n; i = i + p)

{

l = i;

h = i + p - 1;

mid = (l + h) / 2;

Merge (A, l, mid, h);

}

}

if ((p / 2) < n)

merge (A, 0, p / 2, n - 1)

}

if ($A[i] < B[j]$)
 $C[k++] = A[i++];$

void merge (int A[], int l, int mid, int h)
{
 int i = l, j = mid + 1, k = l;
 int B[100];

while ($i \leq \text{mid} \ \&\& \ j \leq h$)
 if ($A[i] < A[j]$)
 $B[k++] = A[i++];$
 else

$B[k++] = A[j++];$
}

for ($i = l; i \leq \text{mid}; i++$)

$B[k++] = A[i];$

for ($i = \text{mid} + 1; i \leq h; i++$)

$B[k++] = A[i];$

for ($i = l; i \leq h; i++$)

$A[i] = B[i];$

}