SORTING

— Stable – same numbers order
Loop-Invariant – each iteration property
In-place – fixed small size memory
Adaptive
└ pre-sortedness

I BUBBLE SORT

Main Idea – keep bubbling the largest element to the end. Switch a pair of neighboring elements

1. Get current i & i + 1
2. Swap them in correct order
3. Start from the front of the array again when reached the end

a[3] > a[4]
Swap a[3] & a[4]

```
void BubbleSort(int a[], int N)   — no. of elements
{
    for(i = 0; i < N-1; i++)   // we are comparing i & i+1 elements
    {
        for(j=0; j < N-i-1; j++)   // N-i-1 → unsorted part length
        {
            if(a[j] > a[j+1])   <=
                swap(a[j], a[j+1])   – set the swap = false
        }
    }
}
```

total no. of iterations → (outer for loop)

i=0, j

4 23 17 25 8

17<23

j j j

4 17 23 25 8

8<25
swap

i = 0
j = 0
a[0], a[1]
j = 1

4 17 23 8 25 — Iteration I

i=2 j=0

j j j
4 17 23 8 25

8<23
swap

4 17 8 23 25 — Iteration II

i=3

j j
4 17 8 23 25

8< 17
swap

4 8 17 23 25 — Iteration III

- sorted

- unsorted

5 4 5′ 7

4 5 5′ 7

Stable - y

Loop-invariant - y

In place - y

Adaptive

10 element

↳ 6th Iteration

6

7th iteration

largest element will be bubbled to the end

max.

3 7

\#

1 2 3 4 — outer loop 'i' loop will run

$\boxed{O(N)}$ — Best

for improvement, add record for whether swapping occured or not — Improved Bubble Sort

#g 8 6 5 4 3

6 8 5 4 3 no. of swaps = 4

6 5 8 4 3

6 5 4 8 3

6 5 4 ~~8~~ 3 8

5 6 4 3 8 No. of swaps = 3

5 4 6 3 8

5 4 3 6 8

4 5 3 6 8 No. of swaps = 2

4 3 5 6 8

1

5 elements

4 + 3 + 2 + 1 = Total no. of swaps

N elements

$$(N-1) + (N-2) + (N-3) + \cdots + 1$$

$$\sum_{i=1}^{N-1} i = \frac{N(N-1)}{2} = \frac{N^2 - N}{2} \Rightarrow O(N^2)$$

II SELECTION SORT

Main idea – Increase the sorted sequence by selecting the smallest element from the unsorted

i) Select the smallest element from unsorted

ii) Append at the end of sorted side

ii) Append at the end of sorted side

Find 1st smallest, 2nd smallest, 3rd smallest

```
Selection Sort (int a[], int N)
{
        for(i = 0; i < N-1; i++)
        {
            min = i;
            for( j = i+1; j < N; j++)
            {   if ( a[j] < a[min])
                      min = j;
            }
        } swap[ a[min], a[i])
}
```

find position of minimum element in unsorted part

#g

Iteration 1

4 23 17 25 8

4<23
4<17
4<25
4<8

i = 0 = min
j = i+1 = 1
a[min] = 4
a[j]

4 | 23 17 25 8

i = 1 = min
j = 2
a[min] = 23
23 > 17
a[1] > a[2]

m = 2
a[min] = 17
a[j] = 8

4 8 | 17 25 23 Iteration 2

4 8 17 | 25 23 Iteration 3

4 8 17 23 25

#g

4_a 2 3 4_b 1

1 2 3 4_b 4_a

1 2 3 4_b 4_a

Stable - N
Loop-invariant - y
In-place - y
Adaptive

1 2 3 4_b 4_a

1 2 3 $\underset{i,\,min}{\underline{4_b}}$ $\underset{j}{\underline{4_a}}$

Best case $\overset{i,\,min}{1}$ $\overset{j}{2}$ 3 4 $O(N^2)$

Worst case $\overset{i,\,min}{8}$ $\overset{j,\,min}{6}$ $\overset{j,\,min}{5}$ $\overset{j,\,min}{4}$ $\overset{j,\,min}{3}$

4 updates

3 $\overset{i,\,min}{6}$ $\overset{j,\,min}{5}$ $\overset{j,\,min}{4}$ 8

3 updates

3 4 5 6 8

N elements

$(N-1) + (N-2) + \ldots \quad 1$

$$\sum_{i=1}^{N-1} i = \frac{N(N-1)}{2} = \frac{N^2 - N}{2} \Rightarrow O(N^2)$$

III INSERTION SORT

Main Idea: Keep inserting next element at its correct place in the sorted sequence

i) Get the 1st element from unsorted side

ii) Find its correct position in the sorted side

iii) Shift elements to make space

```
void InsertionSort( int[] a, int N)
{
    for( int i = 1; i<N; i++)
    {
        int j = i;
        int current = a[i]; // this is the element which
                            //   will be put in the right position
   =>   while( (j>0) && a[j-1] > current)
        {
            a[j] = a[j-1];
```

Find correct

Find the correct position of "current" ← { a[j] = a[j-1]; --j; }

a[j] = current

// j - correct position

#9

7 4 1 3 8 6 2 — i=1, j=1, current = 4

4 7 1 3 8 6 2

1 4 7 3 8 6 2 — i=2, j=2, current = 1

a[j] > current, j=1, j=0

9 7 1

1 3 4 7 8 6 2 — i=3, j=3, current = 3

1 3 4 7 8 6 2 — i=4, j=4, current = 8

1 3 4 6 7 8 2 — i=5, j=5, current = 6

1 2 3 4 6 7 8 — i=6, j=6, current = 2

Best - O(N)

Worst -

Stable - y

Loop-Invariant - y

In-place - y

Adaptive - y

1 2 3 4 5

1 2 | 3 4 5

1 2 3 | 4 5

1 2 3 4 | 5

5 | 4 3 2 1

4 5 | 3 2 1 — 1 shift

3 4 5 | 2 1 — 2 shifts

2 3 4 5 | 1 — 3 shifts

2 3 4 5 | 1 — ~~4~~ shifts

$$T(N) = 1 + 2 + 3 + \cdots (N-1)$$

$$= \sum_{i=1}^{N-1} i = \frac{N(N-1)}{2} \Rightarrow O(N^2)$$

BS	SS	IS
- not used much	- not more than ~~N~~ - swaps	faster than SS.
- theoretical	if writing is expensive	fewer comparisons
		good performance
		(pre-sortedness)
		sorted lists