SORTING ALGORITHMS

SORTING A SET OF RECORDS CONTAINING KEYS, SO THAT KEYS ARE ORDERED ACCORDING TO SOME WELL DEFINED ORDERING RULE, SUCH AS NUMERICAL, ALPHABETICAL OR CHRONOLOGICAL.

REARRANGING THINGS ON A PARTICULAR ORDER

MANY ALGORITHMS BUILT AROUND SORTING

IT IS ONE OF THE FIRST THINGS AN ALGORITHM DESIGNER TRY

SORTING ALGORITHMS DESIGN CONSIDERATIONS

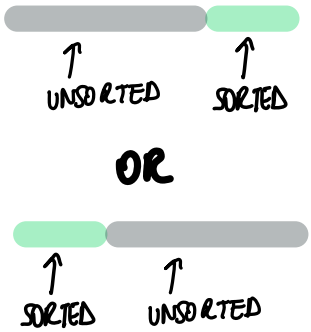
THERE ARE SEVERAL CRITERIA TO BE USED IN EVALUATING A SORTING ALGORITHM

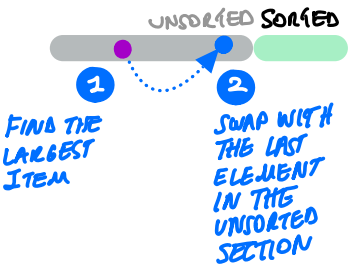
- COST MODEL  → COMPARISON (<, >, ==)
  → EXCHANGE (SWAP)] OR
  → ARRAY ACCESS
- MEMORY USAGE → IN-PLACE  ← REARRANGES ON THE ORIGINAL INPUT
  → EXTRA-MEMORY   ← NEEDS EXTRA STORAGE

ELEMENTARY SORTING ALGORITHMS

SELECTION SORT

THE ALGORITHM DIVIDES THE INPUT LIST INTO TWO PARTS: A SORTED SUBLIST OF ITEMS WHICH IS BUILT UP FROM LEFT TO RIGHT AT THE FRONT OF THE LIST AND A SUBLIST OF THE REMAINING UNSORTED ITEMS THAT OCCUPY THE REST OF THE LIST. INITIALLY, THE SORTED LIST IS EMPTY



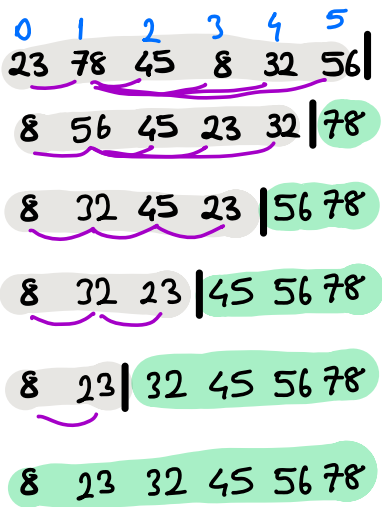


THE ALGORITHM PROCEEDS BY FINDING THE SMALLEST ELEMENT IN THE UNSORTED SUBLIST, EXCHANGING IT WITH THE LEFT MOST UNSORTED ELEMENT, AND MOVING THE SUBLIST BOUNDARIES ONE ELEMENT TO THE RIGHT.

EXAMPLE

FIND THE FIRST MINIMUM, REPLACE WITH THE FIRST ELEMENT
FIND THE SECOND MIN., REPLACE WITH THE SECOND ELEMENT.
⋮

SAMPLE IMPLEMENTATION



	# of comparisons	# of swaps
	5 N-1	1
	4 N-2	1
	3	1
	2	1
	1 N-(N-1)	1
	0 N-N	1
	+	+
	N.N-(1+2+..N)	N
	≈ N(N+1)/2	N

COST MODEL # of COMPARISONS + # OF SWAPS

MEMORY USAGE IN-PLACE

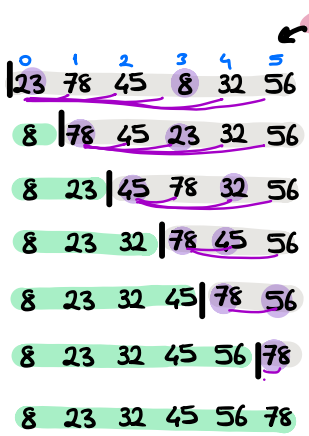
⇒ $O(n^2)$ BEST WORST CASES $O(N^2)$

```
template <class T>
int findIndexOfLargest(T array[], int size){
    int largestIndex = 0;
    for(int i=1; i<size; i++){
        if(array[i] > array[largestIndex])
            largestIndex = i;
    }

    return largestIndex;
}

template <class T>
void selectionSort(T array[], int n){
    for(int last = n-1; last>0; last--){
        int indexOfLargest = findIndexOfLargest(array,
last + 1);
        std::swap(array[indexOfLargest], array[last]);
    }
}
```

ALTERNATIVE IMPLEMENTATION

i=3 MEANS, THE FIRST 3 ITEMS ARE SORTED!

```
for(int i=0; i<n; i++){
    int minIndex=i;
    for(int j=i+1; j<n; j++){
        if(less(a[j],a[minIndex]))
            minIndex=j;
    }
    swap(a, i, minIndex);
}
```



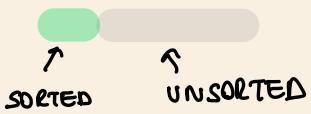
	# of comparisons	# of swaps
	5 N-1	1
	4 N-2	1
	3	1
	2	1
	1 N-(N-1)	1
	0 N-N	1
	+	+
	N.N-(1+2+..N)	N
	$N^2 - \frac{N(N+1)}{2}$	N

$\Rightarrow O(n^2)$ BEST WORST CASES $O(N^2)$

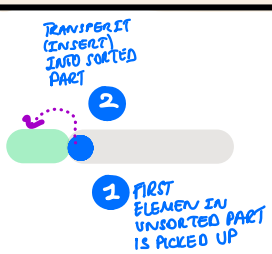
INSERTION SORT

INSERTION SORT IS A SIMPLE SORTING ALGORITHM THAT IS APPRORIATE FOR SMALL INPUTS

THE INPUT LIST IS DIVIDED INTO TWO PARTS: SORTED AND UNSORTED.



EACH PASS, THE FIRST ELEMENT OF UNSORTED PART IS PICKED UP FIRST, TRANSFERRED TO THE SORTED SUBLIST, AND INSERTED AT THE APPROPRIATE PLACE.



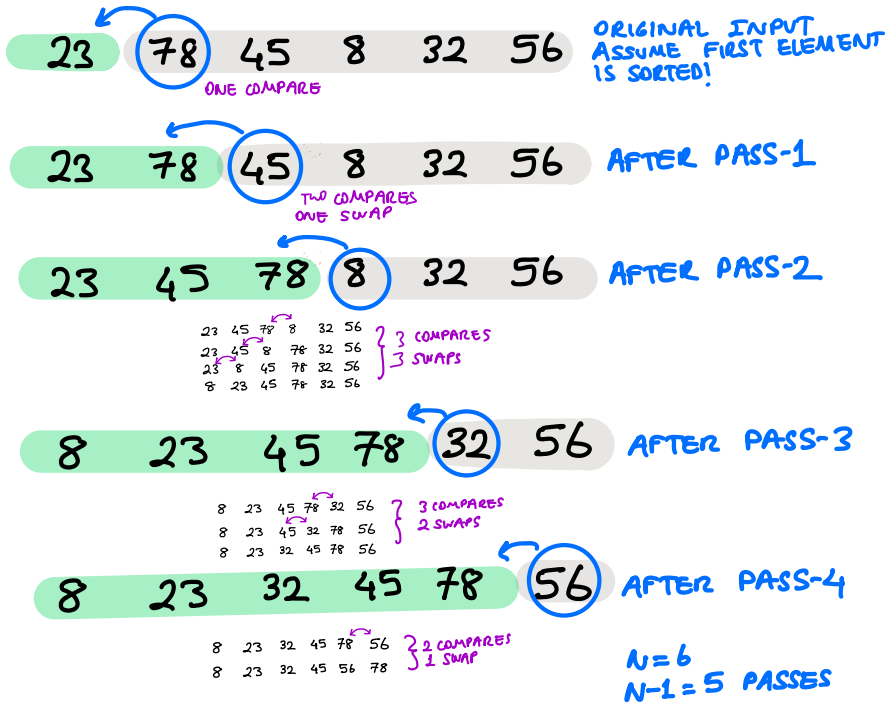
A LIST OF N ELEMENTS WILL TAKE AT MOST N-1 PASSES TO SORT THE DATA.

MOST COMMON SORTING ALGORITHM USED BY CARD PLAYERS

APPROACH:

- CHOOSE THE SECOND ELEMENT IN THE LIST, AND PLACE IT IN ORDER WITH RESPECT TO THE FIRST ELEMENT
- CHOOSE THE THIRD ELEMENT IN THE LIST AND PLACE IN IN ORDER WITH RESPECT TO THE FIRST TWO ELEMENTS
- CONTINUE THIS PROCESS UNTIL DONE

EXAMPLE



SAMPLE IMPLEMENTATION

```
template <class T>
void insertionSort(T array[], int n){
    for(int unsorted = 1; unsorted<n; unsorted++){
        T item = array[unsorted];
        int location = unsorted;
        while (location > 0 && array[location-1] > item){
            array[location] = array[location-1];
            location--;
        }
        array[location] = item;

    }
}
```

AN ALTERNATIVE IMPLEMENTATION

```
for(int i=1; i<n; i++) {
    for(int j=i; j>0 && less(a[j],a[j-1]); j--)
        swap(a, j, j-1);
```

COST MODEL

BEST CASE: THE INPUTS A ARE ALREADY SORTED

$a[0] < a[1] \dots < a[n-1]$ i.e $A = \{1, 2, 3, 4\}$

1 2 3 4

1 2 3 4

1 2 3 4

- ONE COMPARISON PER STAGE $i (i = 1 \dots n-1)$
- NO SWAP/EXCHANGE PER STAGE

$T(n) = \#\text{of COMPARISON} + \#\text{of SWAPS}$

$= (N-1) \cdot 1 + 0$

$T(n) = O(N)$

WORST CASE: THE INPUTS A CONTAIN DISTINCT ITEMS IN REVERSE ORDER

$a[0] < a[1] \dots < a[n-1]$ i.e $A = \{4, 3, 2, 1\}$

STAGE 1 $i=1$

0 1 2 3

4 3 2 1

3 4 2 1

STAGE 2 $i=2$

0 1 2 3

3 4 2 1

3 2 4 1

2 3 4 1

STAGE 3 $i=3$

2 3 4 1

2 3 1 4

2 1 3 4

1 2 3 4

i	#of COMP	#of SWAPS
1	1	1
2	2	2
3	3	3
n-1	n-1	n-1

- i comparisons and i swaps in every stage. $\sum_{i=1}^{n-1} 2i$

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

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

$T(n) = O(N^2)$

MEMORY USAGE IN-PLACE

BUBBLE SORT

COMPARES ADJACENT ITEMS

- EXCHANGES THEM IF IT OUT OF ORDER
- REQUIRES SEVERAL PASSES OVER THE DATA

LARGEST ITEM BUBBLES TO END OF THE ARRAY

PASS-1

29 10 14 37 13
10 29 14 37 13
10 14 29 37 13
10 14 29 37 13
10 14 29 13 37

PASS-2

10 14 29 13 37
10 14 29 13 37
10 14 29 13 37
10 14 13 29 37

PASS-3

10 14 13 29 37
10 14 13 29 37
10 13 14 29 37

PASS-4

10 13 14 29 37
10 13 14 29 37

WORST CASE : $O(N^2)$

BEST CASE : $O(N)$
ALREADY IN ORDER

```
template <class T>
void bubbleSort(T array[], int n){
    int pass = 1;
    bool sorted = false;
    while(!sorted && pass<n){
        sorted = true;
        for(int i=0; i<n-pass; i++){
            if(array[i] > array[i+1])
                std::swap(array[i], array[i+1]);
                sorted = false;
        }

        pass++;
    }

}
```