Searching & Sorting

Searching and sorting are fundamental for efficient data processing. It saves time and resources.

Searching

The process of locating an element in a collection of data. Given a dataset and a “key” element, it finds an element and returns the position if it exists and “No ouput” if it does not.

There are two methods of searching:

- Linear search
- Binary search

Linear Search

A simple searching algorithm that checks elements one-by-one until the target is found.

Algorithm

- Start from the first element of the array
- Check if the element matches the key
    - If it matches, return the position of the element
    - If it does not match, go to the next element
- If last element does not match, return “Not Found”

Pseudocode

```
LINEAR(A, x)
    INPUT A of size n, x
    FOR i ← 0 TO n - 1 DO
        IF A[i] = x THEN
            RETURN i
    END FOR
```

```
    RETURN "Not Found"
END
```

Binary Search

Efficient algorithm to find an element in a sorted array. Uses the divide and conquer method.

Algorithm

- Check the middle element of the array
- Check if the element matches the key
    - If it matches, return the position of the element
    - If it does not match, divide the array and check which array to search next
        - If the key is greater than the middle element, go to the right array
        - If the key is lesser than the middle element, go to the left array
        - Repeat the process with the new array
- If last element does not match, return “Not Found”

Pseudocode

```
BINARY(A, x)
    INPUT A of size n, x
    high ← n - 1
    low ← 0
    WHILE low < high DO
        mid ← (high - low) / 2
        IF A[mid] = x THEN
            RETURN mid
        ELSE IF A[mid] > x THEN
            high = mid - 1
        ELSE
            low = mid + 1
    ENDWHILE
```

```
    RETURN "Not Found"
END
```

Finally, lets look at the difference between the two

	Linear Search	Binary Search
Definition	Goes through every element one-by-one	Divides and conquers
Array type	Sorted or unsorted	Sorted only
Time complexity (Best case)	O(1)	O(1)
Time complexity (Worst case)	O(n)	O(log n)
Space complexity	O(1)	O(1)
Best for	Small, unsorted array	Large, sorted array

Sorting

The process of arranging the elements of a dataset in order (ascending, descending, alphabetical).

There are three different sorting algorithms:

- Bubble sort
- Insertion sort
- Selection sort

Bubble Sort

Compares each element with the adjacent elements and swaps them if they're out of order. The largest element bubbes up automatically.

Algorithm

- Start from the first element
- Check if the element is greater than the next element
    - If it is greater, swap the elements

    - Otherwise, go to the next element and start a new pass
- Repeat this process until no swaps are needed

Pseudocode

```
BUBBLE(A)
   INPUT A of size n
   FOR i ← 0 TO n - 1 DO
      FOR j ← 0 TO n - i - 2 DO
         IF A[j + 1] < A[j] THEN
            SWAP(A[j], A[j + 1])
         ENDIF
      ENDFOR
   ENDFOR
END
```

Insertion Sort

This algorithm builds the sorted array one element at a time. It takes the next element and inserts it int the correct position among the already sorted elements.

Algorithm

- Assume the first element is sorted
- Start with the second element (the key)
- Compare with all the sorted elements (previous elements)
    - Shift the elements that are greater than the key by one position to the right
    - Insert the key on the empty spot
- Go to the next element as the key
- Repeat this until the last element is processed as the key

Pseudocode

```
INSERTION(A)
   INPUT A of size n
   FOR i ← 0 TO n - 1 DO
```

```
        key ← A[i]
        j ← i
        WHILE j >= 0 AND A[j] > key DO
            A[j + 1] ← A[j]
            j ← j - 1
        ENDWHILE
        A[j + 1] ← key
    ENDFOR
END
```

Selection Sort

This algorithm picks a “key” element, and the minimum element from the unsorted array, and swaps the elements, until te array gets sorted (the last element becomes the key).

Algorithm

- Select the first element as key
- Get the least element of the unsorted section of the array
    - If both are the same, then do nothing
    - Otherwise, swap both the elements, and the minimum element will be part of the sorted array
- Go to the next element, and repeat the process until the array is sorted

Pseudocode

```
SELECTION(A)
    INPUT A of size n
    FOR i ← 0 TO n - 1 DO
        minIndex = i
        key = A[i]
        FOR j ← i TO n - 1 DO
            IF A[j] < A[minIndex] THEN
                minIndex ← j
            ENDIF
        ENDFOR
        SWAP(A[minIndex], key)
```

```
    ENDFOR
END
```