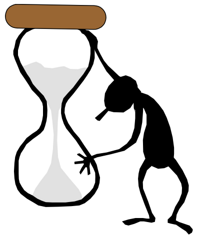
Fundamentals of Data Structures

Mufleh Al-Shatnawi, Ph.D., P.Eng.,

Algorithms Analysis: Computational complexity

- Computational complexity is concerned with describing the amount of **resources** needed to run an algorithm
  - For our purposes, the resource is **time**
  - Time: How much longer does it run?



- Complexity is usually expressed as a function of $\boldsymbol{n}$, where n is the size of the problem
  - The size of the problem is always a **`non-negative`** integer value (i.e., a natural number)

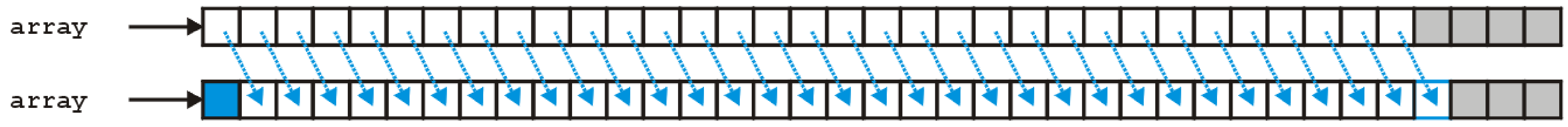
In the real world, we do want to know whether *Algorithm A* runs faster than *Algorithm B* on data set C

Algorithms and Data Structures are important

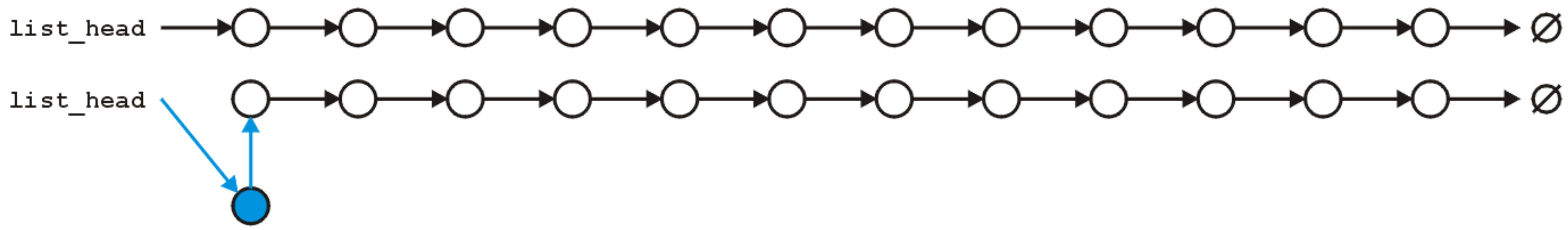
- Consider accessing the k^{th} entry in an **array** or ***linked list***
  - In an array, we can access it using an index `array[k]`
  - We must step through the first $k - 1$ nodes in a linked list
- Consider searching for an entry in a sorted **array** or ***linked list***
  - In a sorted array, we use a fast **binary search**
    - Very fast
  - We must step through all entries less than the entry we're looking for
    - Slow

Algorithms and Data Structures are important

- However, consider inserting a new entry to the start of an **array** or a **linked list**
  - An array requires that you copy all the elements in the array over
    - ***Slow for large arrays***



  - A linked list allows you to make the insertion very quickly
    - ***Very fast regardless of size***



Complexity

- **Time Complexity:** Number of **steps** ("runtime") executed by an algorithm
- **Space Complexity:** Number of units of space required by an algorithm
  - e.g., number of elements in a list
  - number of nodes in a tree or graph

Note

- **Running time** could be measured by counting the number of times **all lines** are executed or the number of times **some lines** are executed.
- **Counting the Number of Times All Lines are Executed**: This approach involves analyzing every line of code in an algorithm and determining how many times each is executed.
  - This comprehensive approach gives a detailed view of the algorithm's performance but can be complex for more extensive algorithms.
- **Counting the Number of Times Some Lines are Executed**: In many cases, it's more practical to focus on specific lines of code most critical to the algorithm's performance. **These are typically the lines inside loops or recursive calls.**
- It’s up to the problem or what the question asks, so always read the question carefully.

Example: Linear Search

Algorithm (Pseudocode):

```
function linearSearch(array, element):
    for i = 0 to length(array) - 1:
        if array[i] == element:
            return i
    return -1
```

Counting All Lines:

- The for loop runs **n** times, where **n** is the array's length.
- The if statement is checked **n** times.
- The return **i** or return **-1** line is executed once, but only after the for loop completes or the element is found.
- Total count: **2n + 1** (approximately, assuming the element is not found early).

Example: Linear Search

Algorithm (Pseudocode):

```
function linearSearch(array, element):
    for i = 0 to length(array) - 1:
        if array[i] == element:
            return i
    return -1
```

Counting Some Lines:

- Here, we might focus on the comparison **array[i] == element**, as it is key to finding the element
- This comparison is executed **n** times
- This gives us a simplified view: **The running time is proportional to n**.