LEARN CODING

ale66

github.com/ale66/learn-coding

SORTING

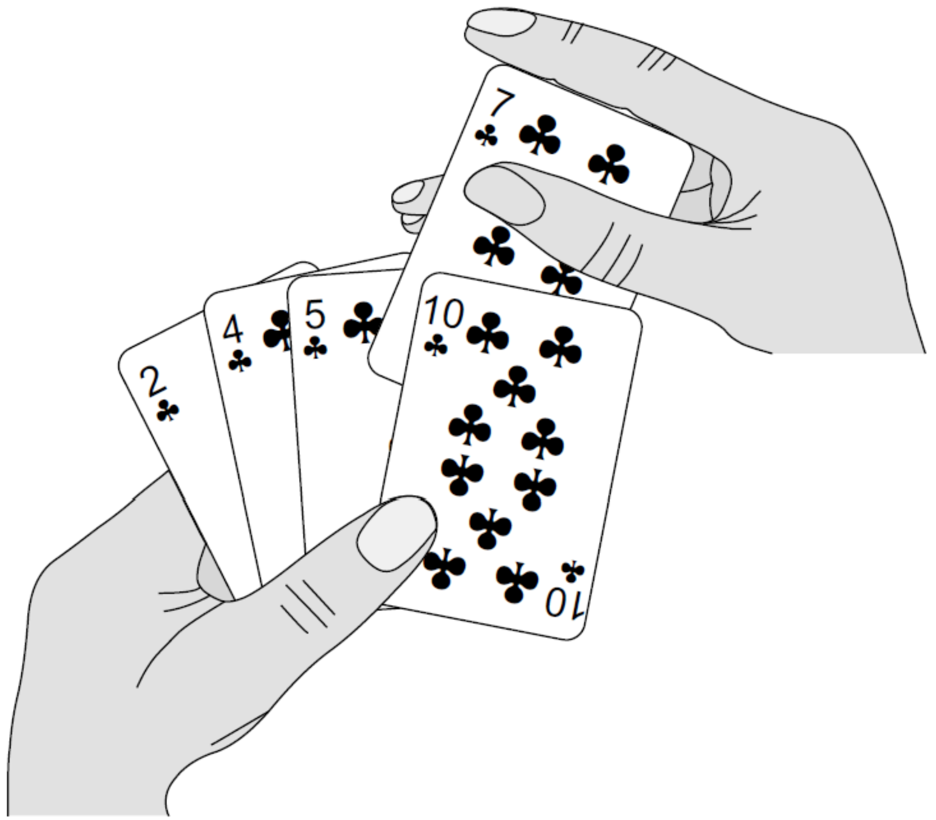


Figure 2.1 Sorting a hand of cards using insertion sort.

STATEMENT

Instance:

a sequence of *n* integers: $A = a_1, a_2 \dots a_n$

Solution:

A permutation of the values $\pi : [1..n] \to [1..n]$

Constraint

values never decrease: $a_1 \leq a_2 \dots a_n$

Let's assume that there are no repeated values and Python notation (first elem. is in pos. 0)

```
A = [11, 6, 8, 2, 22, 16, 25]

sort(A) = [?, ?, ?, ?, ?, ?]
```

$$\pi(0) = ?$$

$$\pi(1) = ?$$

$$\pi(2) = ?$$

$$\pi(3) = ?$$

...

Let's assume that there are no repeated values

```
A = [11, 6, 8, 2, 22, 16, 25]

sort(A) = [2, 6, 8, 11, 16, 22, 25]
```

$\pi(0) = 3$: the elem. in position 1 now goes to pos. 3

$\pi(1) = 1$: the elem. in pos. 1 remains there

$\pi(2) = 2$: so does the elem. in pos. 2

$\pi(3) = 0$: the elem. in pos. 3 now goes to pos. 0

GOOD NEWS ABOUT SORTING

Solvable within $Kn \log_2 n$ steps

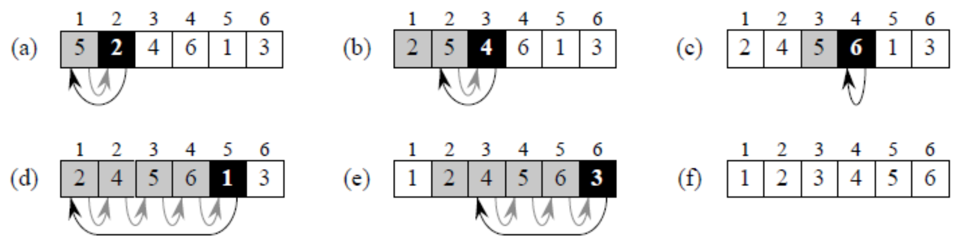


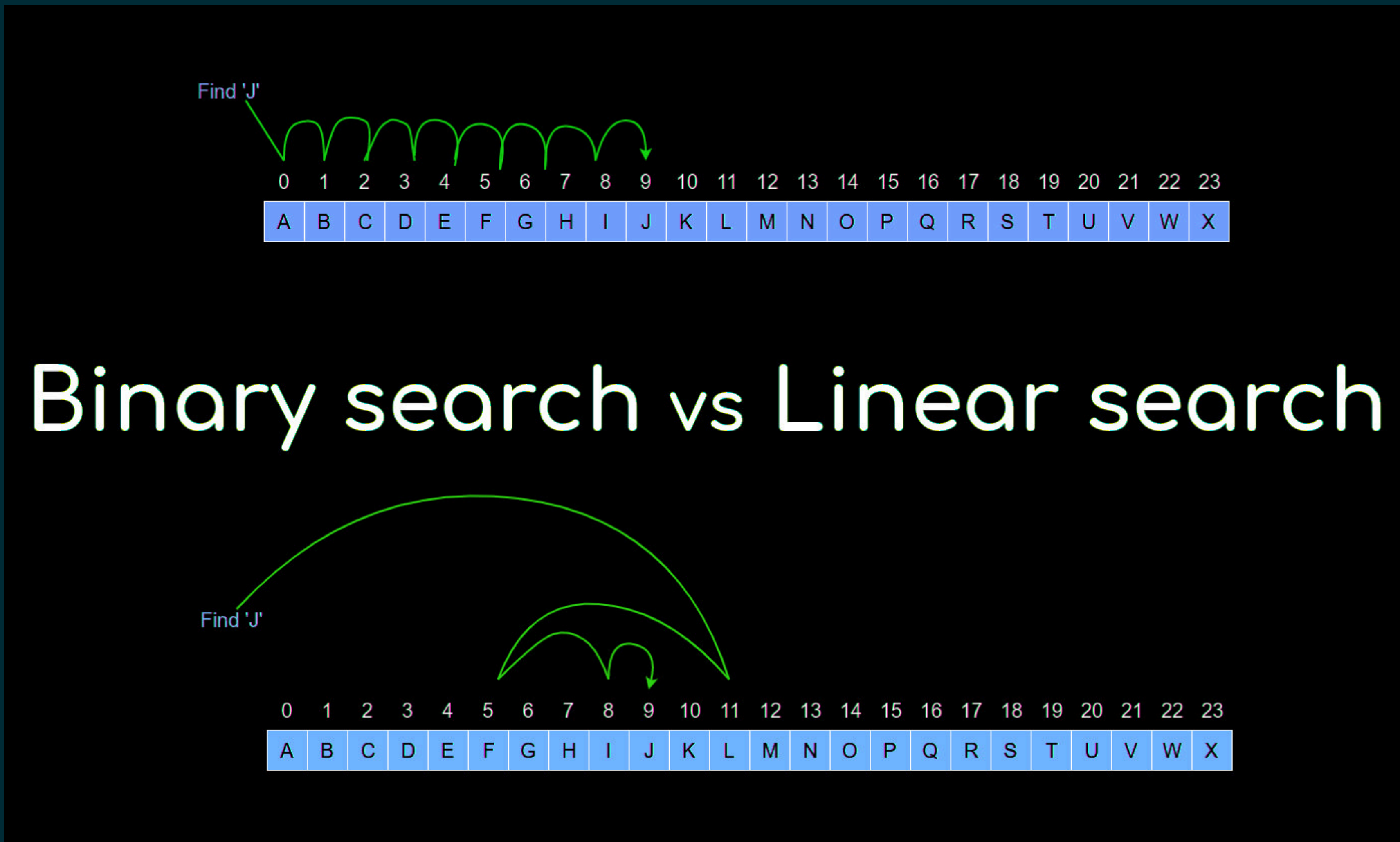
Figure 2.2 The operation of INSERTION-SORT on the array $A = \langle 5, 2, 4, 6, 1, 3 \rangle$. Array indices appear above the rectangles, and values stored in the array positions appear within the rectangles. **(a)–(e)** The iterations of the **for** loop of lines 1–8. In each iteration, the black rectangle holds the key taken from $A[j]$, which is compared with the values in shaded rectangles to its left in the test of line 5. Shaded arrows show array values moved one position to the right in line 6, and black arrows indicate where the key moves to in line 8. **(f)** The final sorted array.

Even quicker when A is already half-sorted

Python runs `powersort()`, an optimised version of TimSort

```
a = [11, 6, 8, 2, 22, 16, 25]

a.sort()
```

SEARCHING



Find 'J'
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
A B C D E F G H I J K L M N O P Q R S T U V W X
Binary search vs Linear search
Find 'J'
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
A B C D E F G H I J K L M N O P Q R S T U V W X

Istance:

- a collection of n integers $A = a_1, a_2, \ldots a_n$
- an integer k

Question: does k belong to the collection?

OBSERVATIONS

It can be generalised to data types that are ordered (strings have alphanumerical ordering)

There is a simple algorithm that will answer after at most n comparisons

this is a very basic problem which is used elsewhere: it is important that the implementation is quick and well-tested

CASE STUDY: EGO NETWORKS

- while less than N profiles collected
  - generate random FB ids (a fixed-lenght, 32-digits integer)
  - test the random id: does it land on an open FB profile?
    - yes: expand the visit to the neighborood
    - no: go back to generating random ids

Cost: up to n comparisons

SIMPLE SOLUTION

```python
def search(a, k):
  '''Linear search'''

  found = False
  n = len(a)

  for elem in a:
    if k == elem:
      found = True

  return found
```

EXERCISE

Apply `while` instead of `for` to stop operations as soon as the key value is found

Return the position at which the key was found

ORDERED SEQUENCES

Special case: the input sequence A is already sorted, either in increasing or decreasing

a *much more efficient* algorithm is available

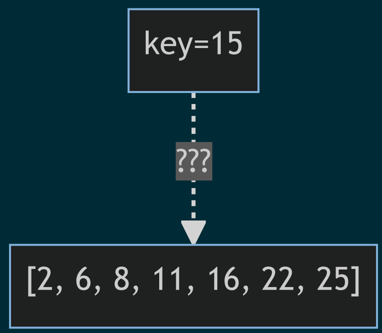
Binary_search, which is correct only for sorted sequences, will take *at most* $\log_2 n$ comparisons before we stop

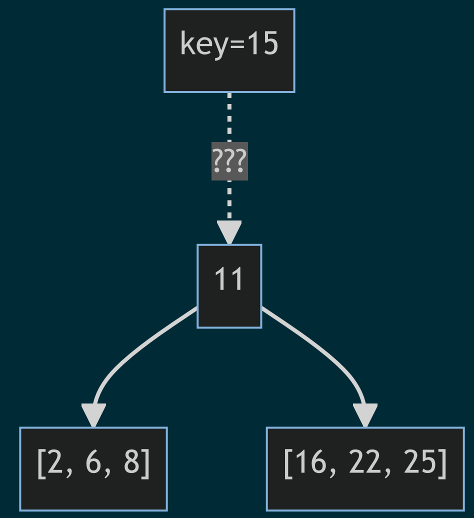
items	Linear	BS
1,000	1,000	10
1,000,000	1,000,000	20
1,000,000,000	1,000,000,000	30

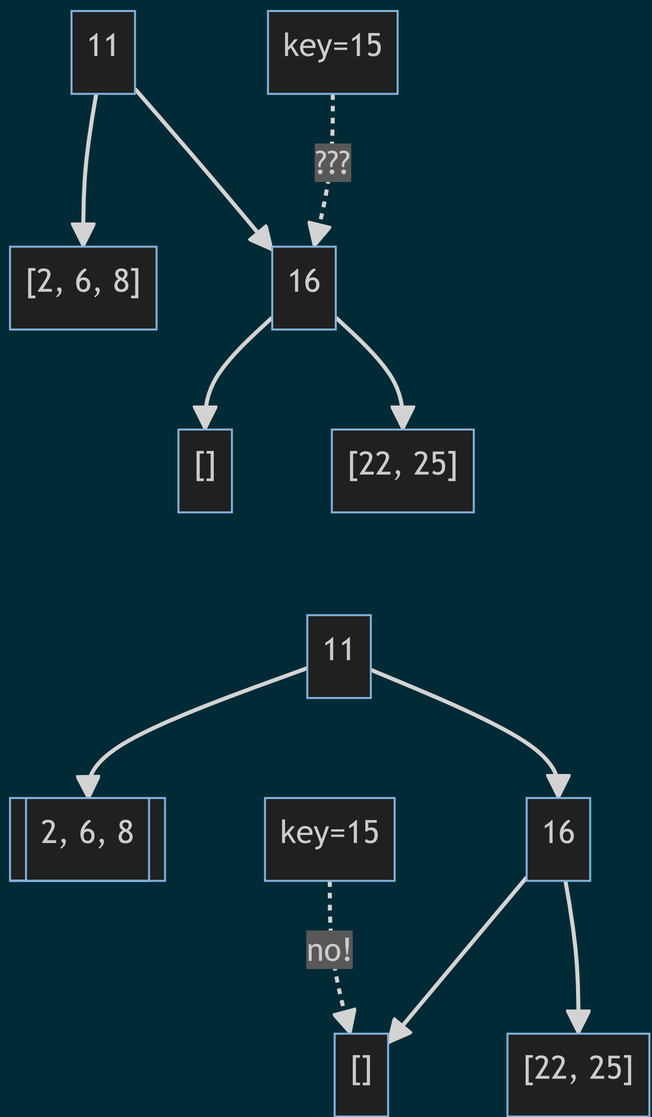
IDEA

Data are sorted: exploit this property to cut down the size of the list *segment* to be checked

```
my_sorted_list = [2, 6, 8, 11, 16, 22, 25]

key = 15
```







We checked only two values (11 and 16) but we can stop already and answer ‘no’

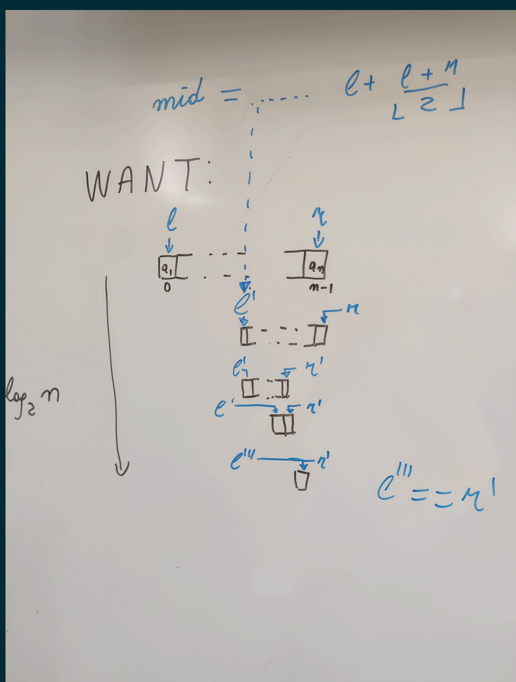
STEPS

- input the ordered list and the key value to be searched
- find the *median* value (here, it's right in the middle!)
- if the median $==$ the key value then stop and say 'yes'
- but if the search key $>$ median then the value, if it exists, can only be in second half of the list
- otherwise, the key value, if it exists, can only be in the first half of the list
- depending on the result of the comparison, continue searching on the 'right' half of the list.

BS halves the searched data at each iteration

Soon, the halving will shrink the list down to just one value, so we check and finish

How soon? It will take at most $\log_2 n$ 'cuts' to shrink the list down to 1

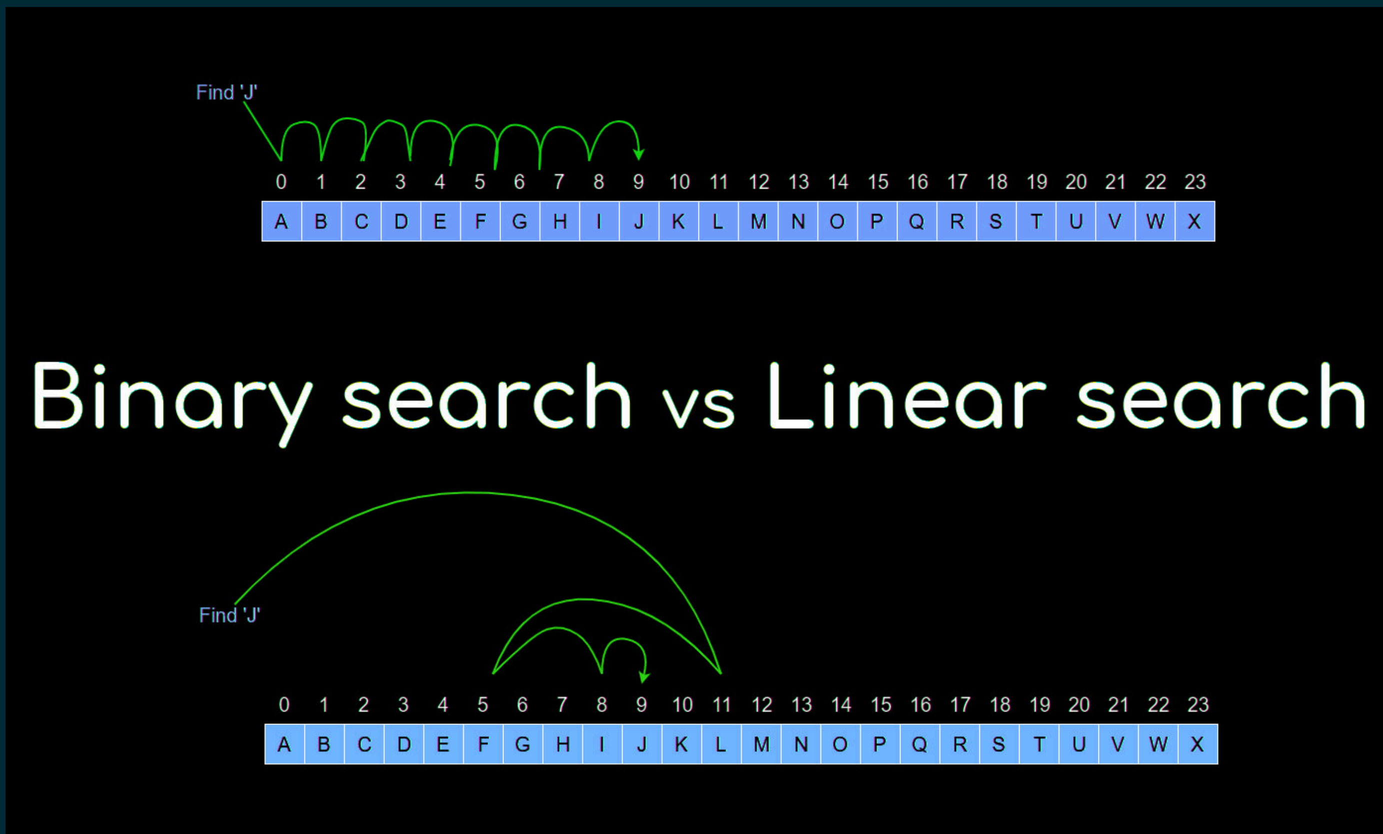


```python
def bs(a, k):
    '''Simple Binary search implementation: a is a list of integers, k is a

    found = False
    n = len(a)

    # the boundaries of our search
    l = 0
    r = n
```

```python
    while ((found == False) and (l < r)):

        mid = l + int((r-l)/2)

        if k < a[mid]:
            r = mid

        elif k > a[mid]:
            l = mid + 1

        else:
            found = True
            print('found in position ', mid)

    return found
```

VISUALISATION

Instance: an ordered list of 24 uppercase chars, `key = 'J'`



$$\lceil \log_2 24 \rceil = \lceil 4.5849 \rceil = 5 \text{ comparisons}$$