COMP 110

CL13: Big-O Notation

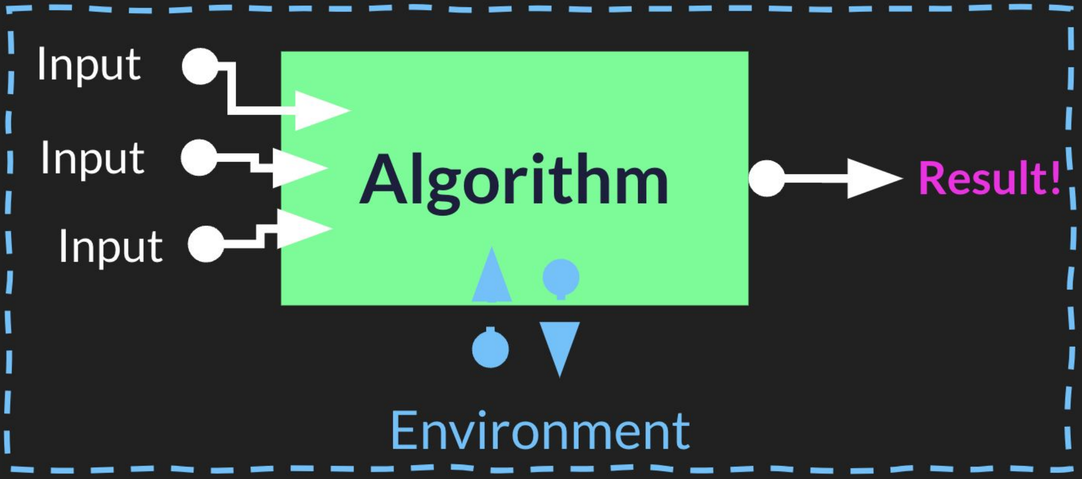
Recall: Algorithms

Input is data given to an algorithm

An **algorithm** is a series of steps

An algorithm **returns** some **result**

An algorithm *may* be influenced by its **environment** and it *may* produce side-effects which influence its environment.



What is an algorithm?

- A set of steps to solve a general problem
- Finite
- Can handle a problem of arbitrary size

How do we measure how “good” an algorithm is?

- Is it correct?
- How long does it take to implement?
- How much computer memory does it take?

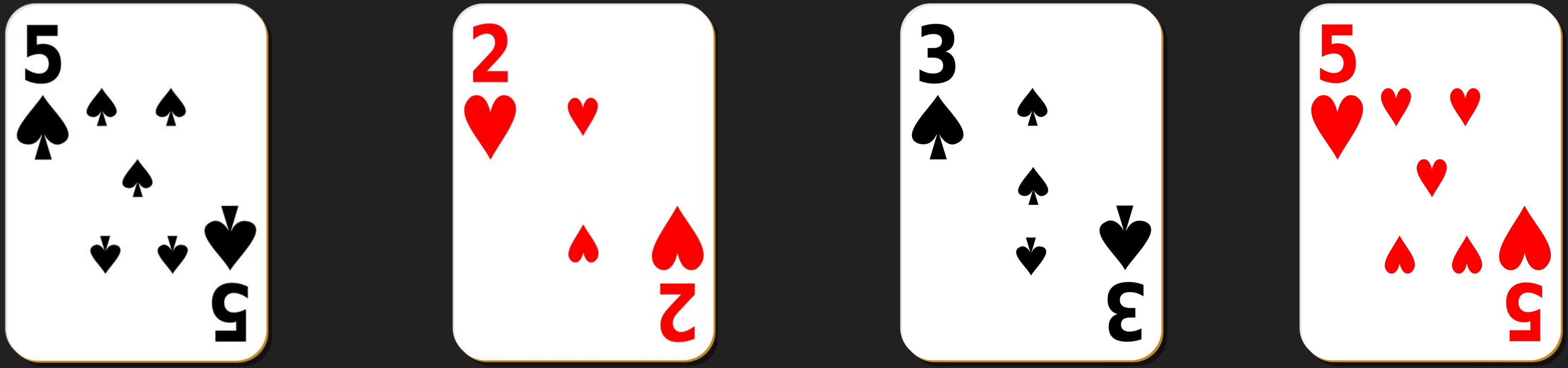
Why do we care about computation speed?

- Security: Cryptography works because encrypted information takes *too long* to decipher!
- User Experience: Users don’t want to work with a slow application!
- Big Data: We want to be able to feed as much data as possible into our systems, but we need a way to *efficiently* do that!

Measurements We Use

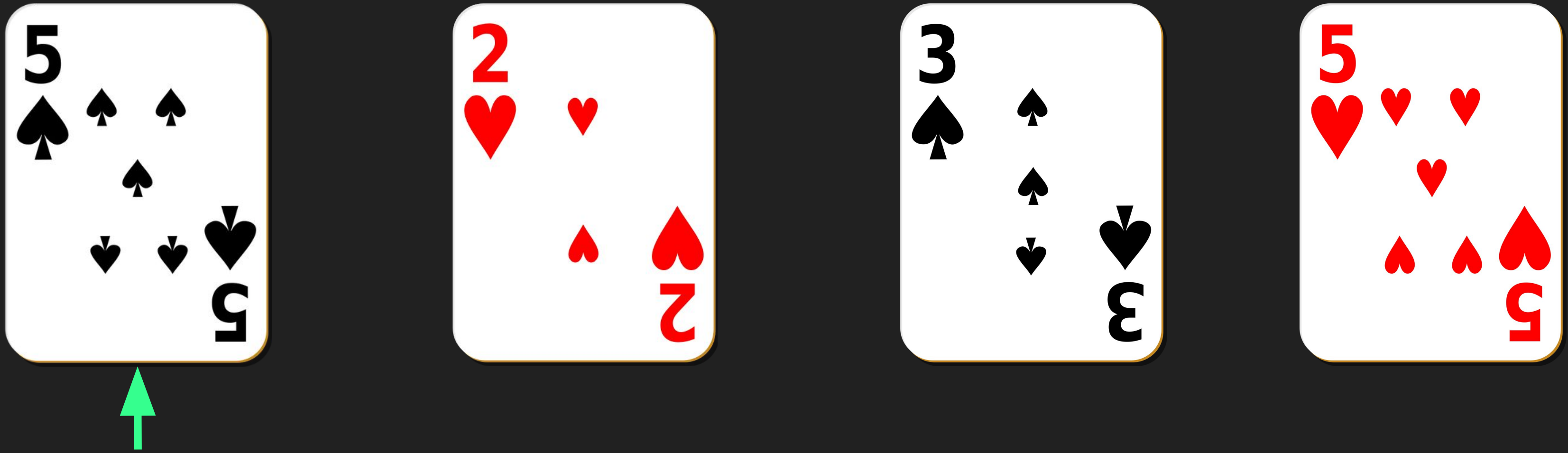
- O “Big O”: upper bound (worst case runtime)
- Ω “Big Omega”: lower bound (best case runtime)
- Θ “Big Theta”: average runtime

Returning to Finding the Lowest Card in a Deck

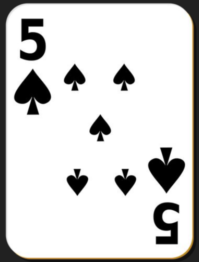


- Go from left to right
- Remember the lowest card you've seen *so far* and compare it to the next cards

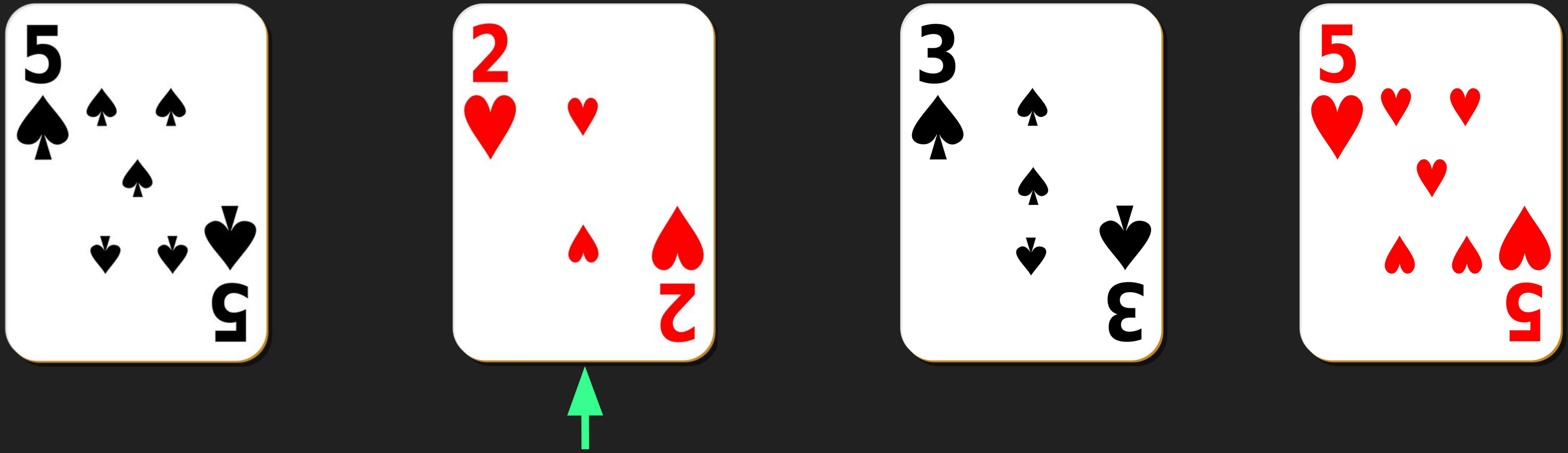
Finding the Lowest Card



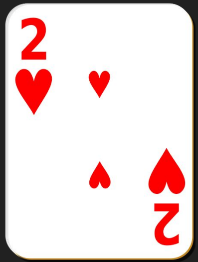
Low card:



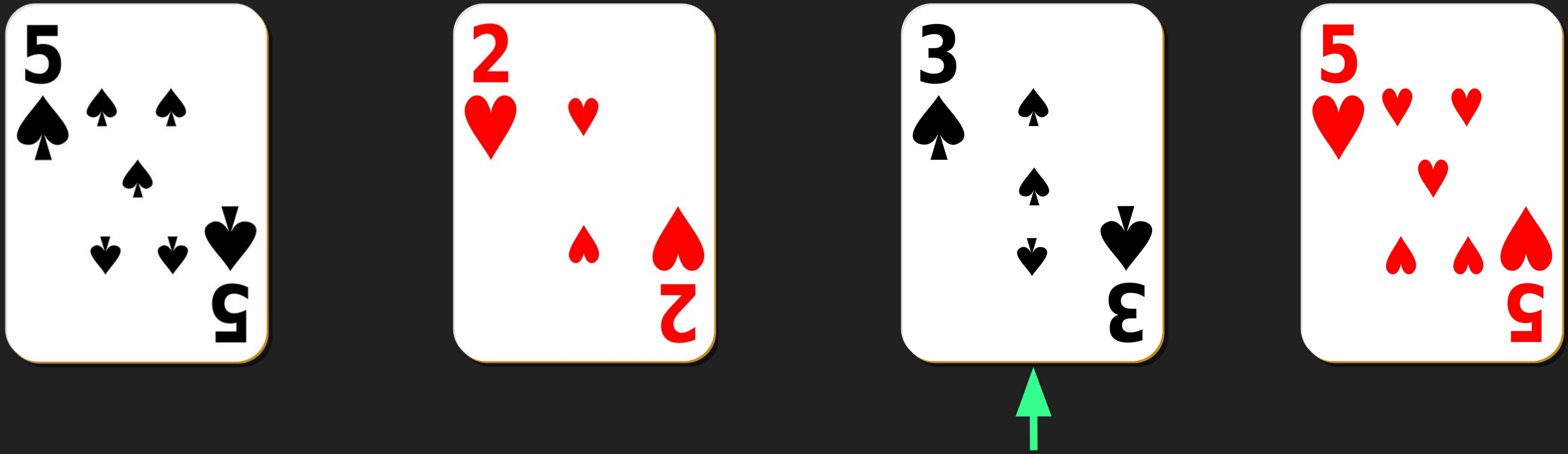
Finding the Lowest Card



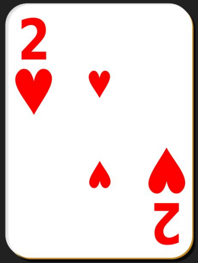
Low card:



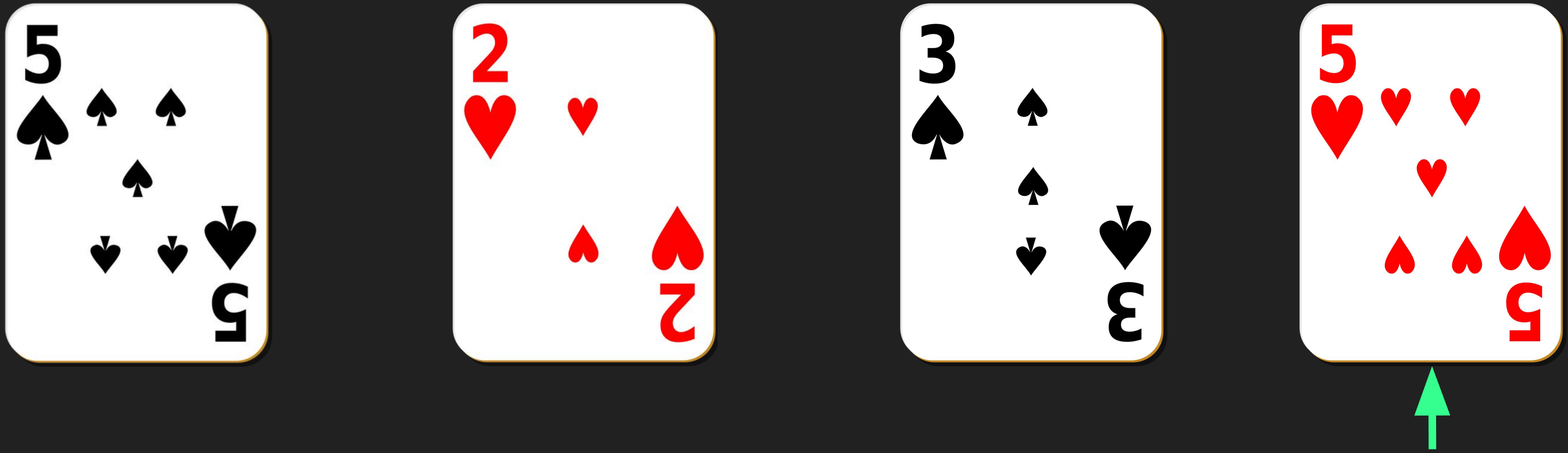
Finding the Lowest Card



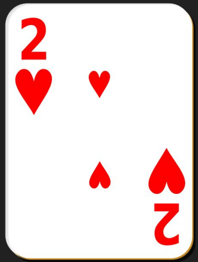
Low card:



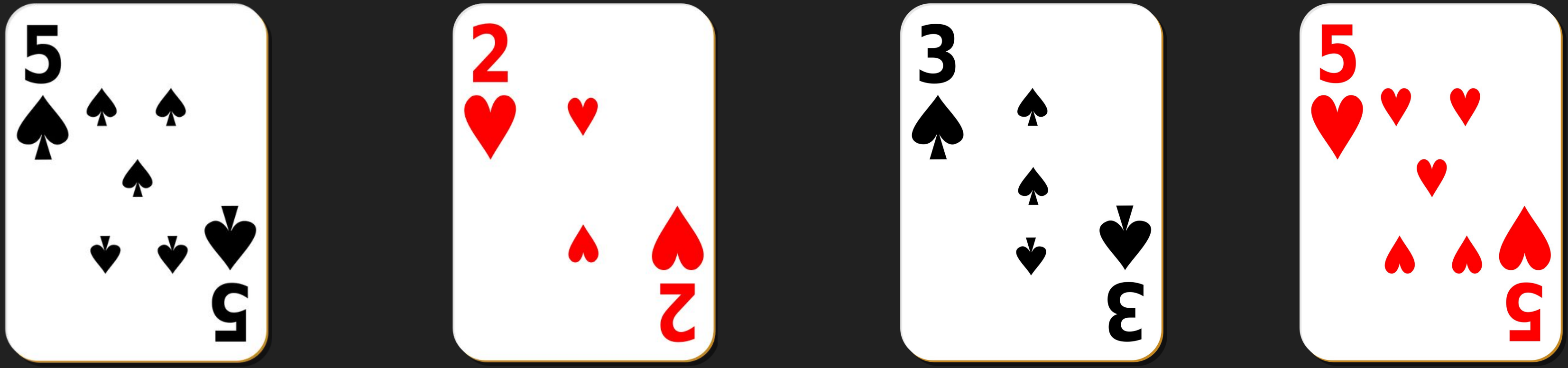
Finding the Lowest Card



Low card:

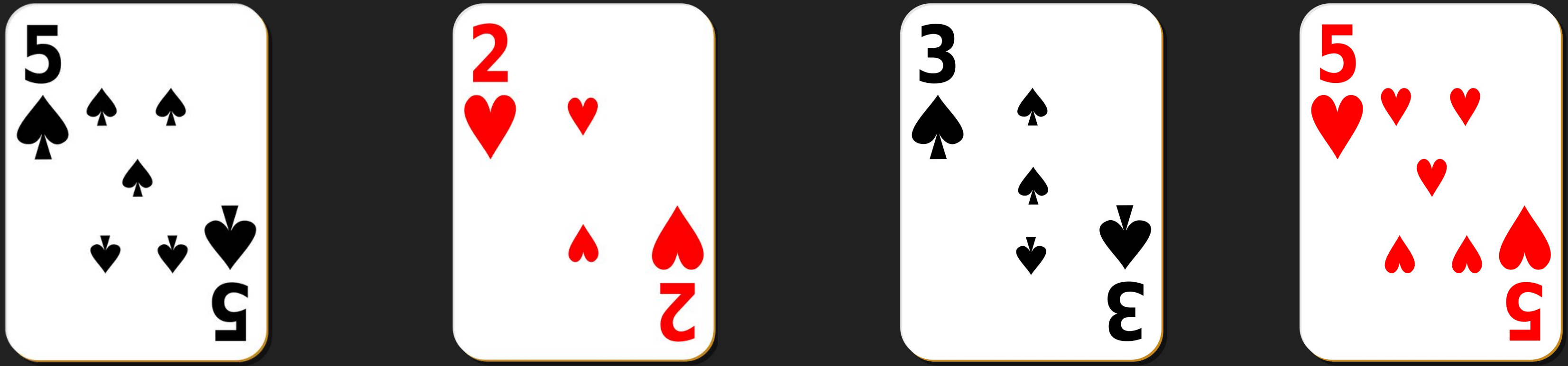


Finding the Lowest Card

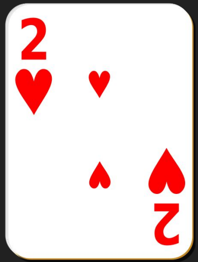


Low card:

Finding the Lowest Card

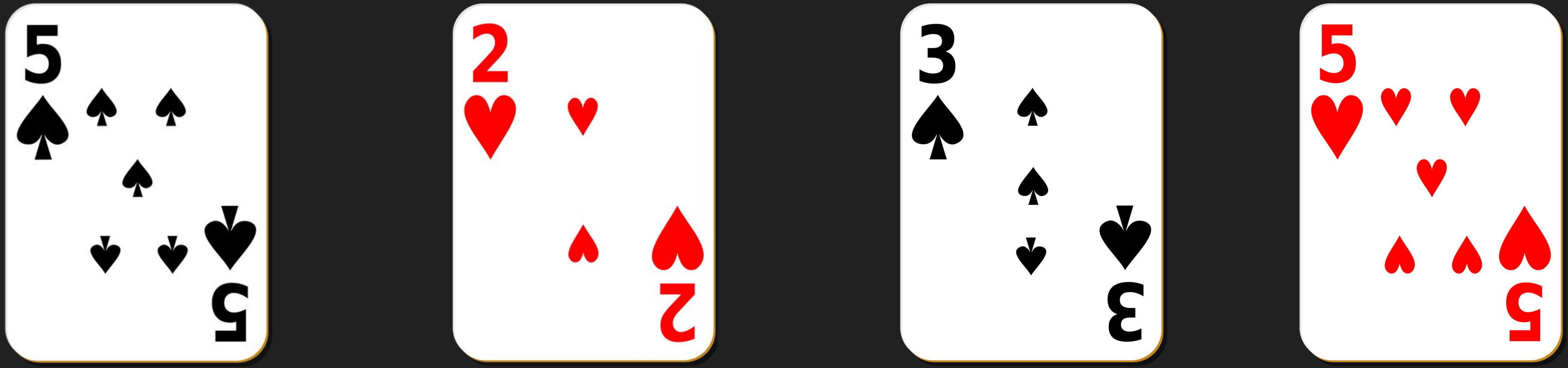


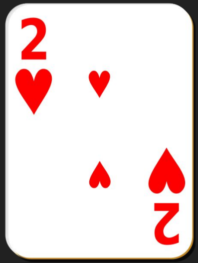
Low card:



4 actions for input of 4 cards.

Finding the Lowest Card



Low card: 

4 actions for
input of 4 cards.
→
n actions for
input of size n.

Finding the Lowest card

- In this approach, we always have to check every card in the deck, so our runtime will always be approximately *n* where *n* is the size of the deck.

$$\text{Finding the minimum} \in O(n)$$

$$\text{Finding the minimum} \in \Omega(n)$$

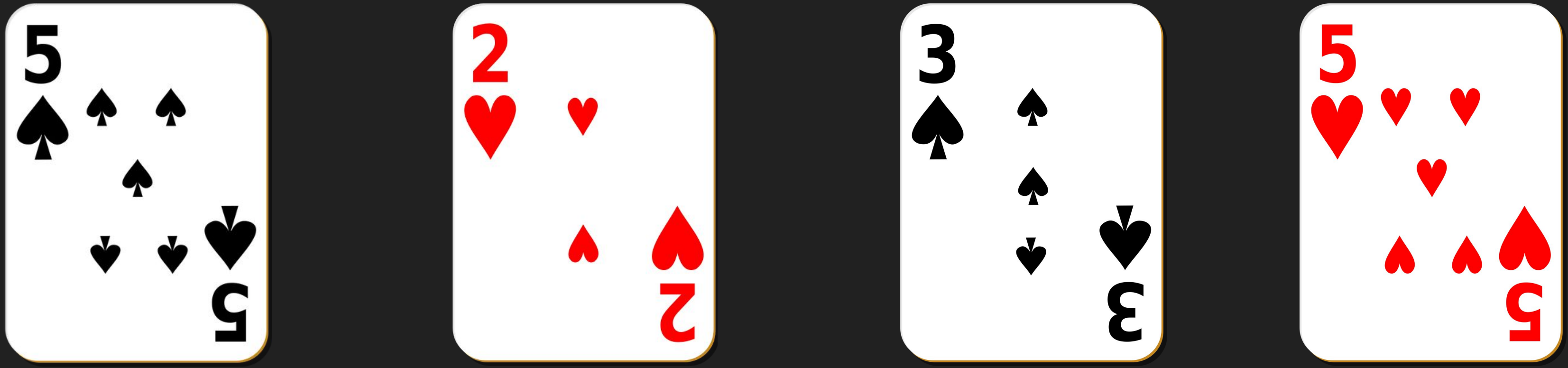
$$\text{Finding the minimum} \in \Theta(n)$$

Speed vs. Memory

- Sometimes you can make a tradeoff between speed and memory.
- E.g. storing a value rather than computing it repeatedly.

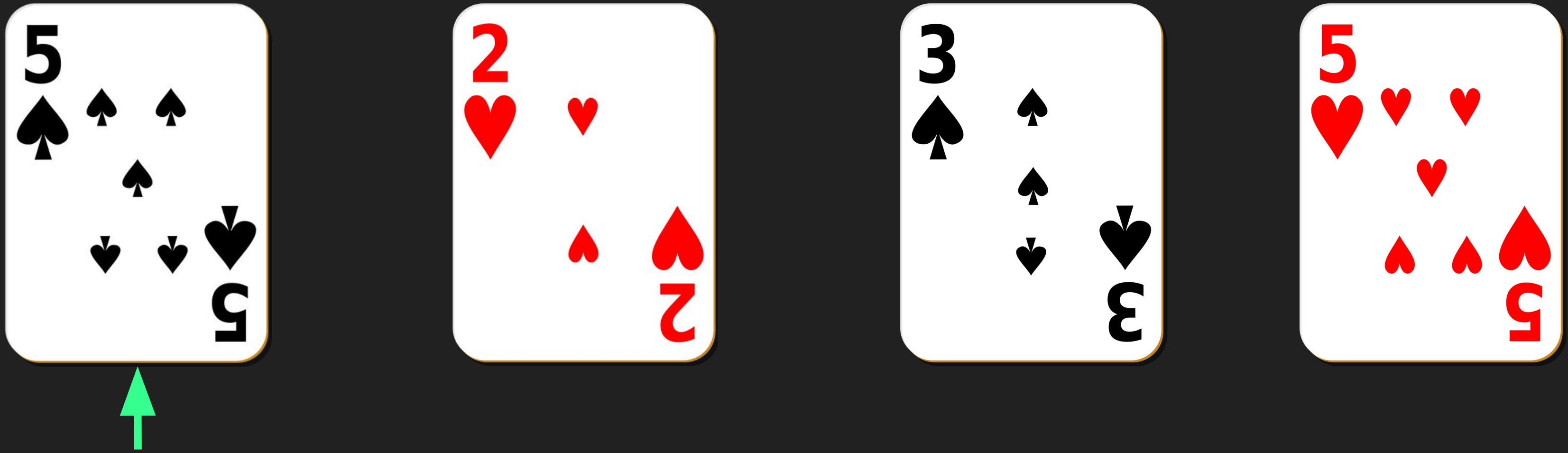
```python
def find_min1(nums: list[int]) -> int:
    min_idx: int = 0
    idx: int = 0
    while idx < len(nums):
        if nums[idx] < nums[min_idx]:
            min_idx = idx
        idx += 1
    return min_idx

def find_min2(nums: list[int]) -> int:
    min_idx: int = 0
    min_val: int = nums[min_idx]
    idx: int = 0
    while idx < len(nums):
        if nums[idx] < min_val:
            min_idx = idx
            min_val = nums[idx]
        idx += 1
    return min_idx

search_vals: list[int] = [10, 9, 8]
find_min1(search_vals)
find_min2(search_vals)
```

New Example: Finding a specific card.

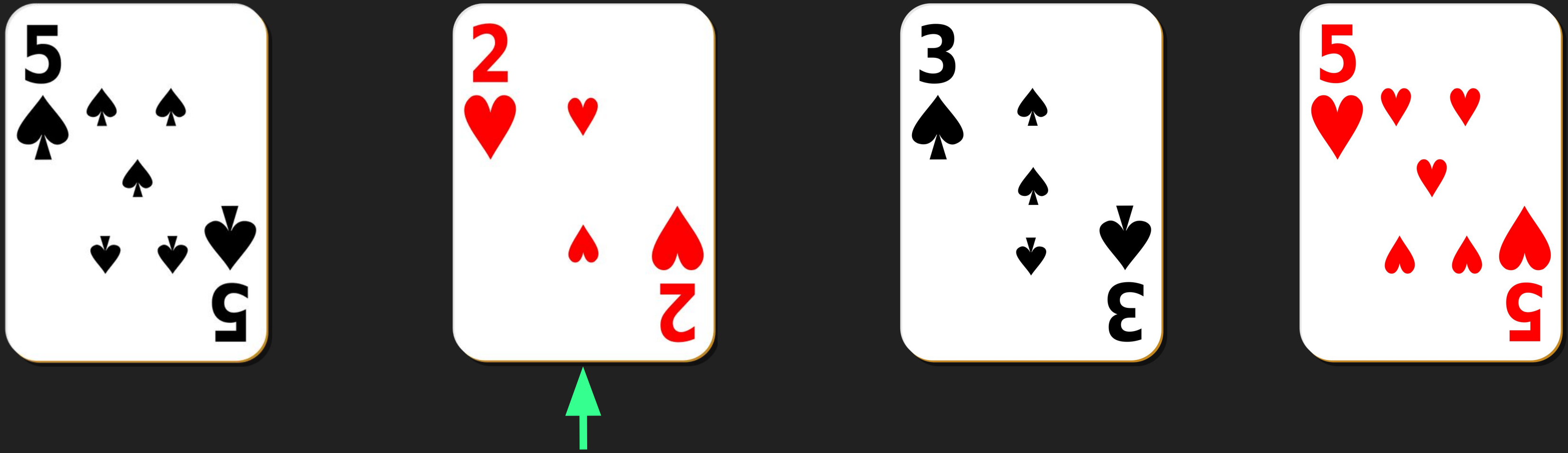


- Go from left to right
- The first time you see your card, exit!

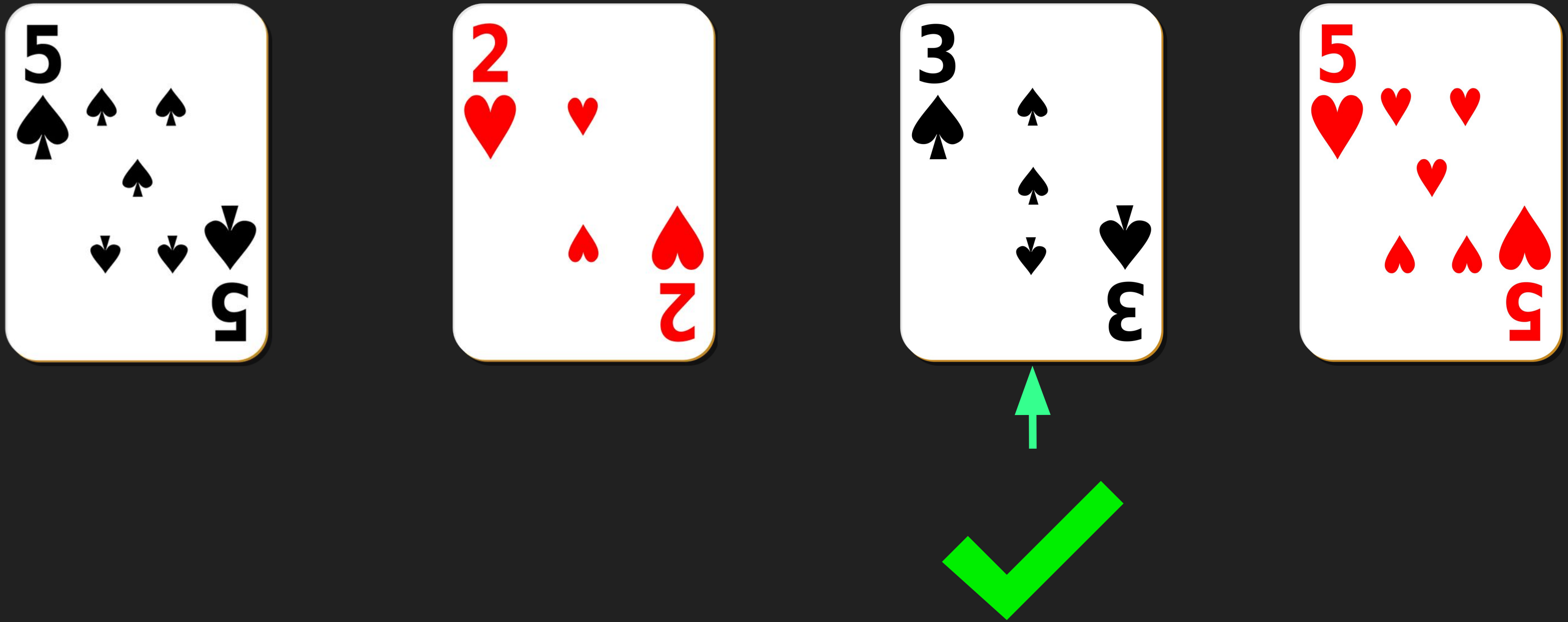
Finding 3



Finding 3



Finding 3



Worst Case

What is the worst case input for this algorithm? (What will make us look at the *most* cards before exiting?)

What is the Big-O (worst case) runtime in terms of deck size n?