Encoding (Representation)

Cards in a deck are numbered **0 – 51**

Card Suit	
0	**Clubs**
1	**Diamonds**
2	**Hearts**
3	**Spades**

Card Rank	
0	**Deuce**
1	**Three**
2	**Four**
3	**Five**
4	**Six**
5	**Seven**
6	**Eight**
7	**Nine**
8	**Ten**
9	**Jack**
10	**Queen**
11	**King**
12	**Ace**

How can we convert *to* & *from* a **card#** and a specific **suit** and **rank**?

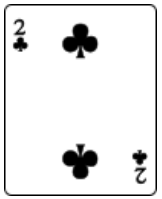
Encoding (Representation)

Cards in a deck are numbered **0 – 51**

Card Suit	
0	**Clubs**
1	**Diamonds**
2	**Hearts**
3	**Spades**

Card Rank	
0	**Deuce**
1	**Three**
2	**Four**
3	**Five**
4	**Six**
5	**Seven**
6	**Eight**
7	**Nine**
8	**Ten**
9	**Jack**
10	**Queen**
11	**King**
12	**Ace**

Card# = Suit * 13 + Rank



Suit = **0**
Rank = **0**
0 * 13 + 0 = **0**

Suit = **2**
Rank = **10**
2 * 13 + 10 = **36**

Suit = **3**
Rank = **12**
3 * 13 + 12 = **51**

Decoding (Representation)

Cards in a deck are numbered **0 – 51**

Card Suit	
0	**Clubs**
1	**Diamonds**
2	**Hearts**
3	**Spades**

Card Rank	
0	**Deuce**
1	**Three**
2	**Four**
3	**Five**
4	**Six**
5	**Seven**
6	**Eight**
7	**Nine**
8	**Ten**
9	**Jack**
10	**Queen**
11	**King**
12	**Ace**

Suit = Card# // 13
Rank = Card# % 13

// returns the integer quotient
39 // 7 = 5

% is the modulus (remainder)
39 % 7 = 4

Decoding (Representation)

Cards in a deck are numbered **0 – 51**

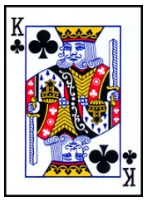
Card Suit	
0	**Clubs**
1	**Diamonds**
2	**Hearts**
3	**Spades**

Card Rank	
0	**Deuce**
1	**Three**
2	**Four**
3	**Five**
4	**Six**
5	**Seven**
6	**Eight**
7	**Nine**
8	**Ten**
9	**Jack**
10	**Queen**
11	**King**
12	**Ace**

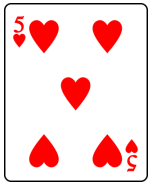
Suit = Card# // 13

Rank = Card# % 13

Card # = **11**
Suit = 11 **//** 13 = **0**
Rank = 11 **%** 13 = **11**



Card # = **29**
Suit = 29 **//** 13 = **2**
Rank = 29 **%** 13 = **3**



Card # = **48**
Suit = 48 **//** 13 = **3**
Rank = 48 **%** 13 = **9**

Run list_cards.ipynb – **Cells 1…2**

Declare two string arrays: `suits` and `ranks` to store human-readable card identifiers ①

The index number of each element matches the encoding table in the slide deck ②

```
[1]  # Cell 1

     import numpy as np

     # fmt: off
     suits = ["Clubs", "Diamonds", "Hearts", "Spades"]

     ranks = ["Deuce", "Three", "Four", "Five", "Six", "Seven",
              "Eight", "Nine", "Ten", "Jack", "Queen", "King", "Ace"]
     # fmt: on
```

③ ④

Note: You should not edit this file!

Define a function to initialize a deck (put the cards in order) ⑤

Each element in the deck array is set to its index number: position # = card # ⑥

```
[2]  # Cell 2
     def init_deck():
         return np.arange(0, 52)

     init_deck()
```

⑦ ⑧ ⑨

Position Number	0	1	2	3	4
Card Number	0	1	2	3	4

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
       34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50,
       51])
```

⑩

Run list_cards.ipynb – **Cell 3**

Define a function to print a deck of cards ← ①

We must convert a `card number` to the specific suit # and rank # for that card number ← ②

```
[3]  # Cell 3
     def print_deck(deck):                                          ← ③
         for card_pos in range(52):                                 ← ④
             card_num = deck[card_pos]                              ← ⑤
             suit_num = card_num // 13                              ← ⑥
             rank_num = card_num % 13                               ← ⑦
             card_name = f"{ranks[rank_num]} of {suits[suit_num]}"  ← ⑧
             print(f"The card in position {card_pos:2} is the {card_name}")  ← ⑨
```

Create a deck and then print out that deck

```
[4]  # Cell 4
     deck = init_deck()
     print_deck(deck)
```

```
The card in position  0 is the Deuce of Clubs
The card in position  1 is the Three of Clubs
The card in position  2 is the Four of Clubs
The card in position  3 is the Five of Clubs
The card in position  4 is the Six of Clubs
The card in position  5 is the Seven of Clubs
The card in position  6 is the Eight of Clubs
```

Run list_cards.ipynb – **Cell 4**

Define a function to print a deck of cards

We must convert a `card number` to the specific suit # and rank # for that card number

```
[3]  # Cell 3
     def print_deck(deck):
         for card_pos in range(52):
             card_num = deck[card_pos]
             suit_num = card_num // 13
             rank_num = card_num % 13
             card_name = f"{ranks[rank_num]} of {suits[suit_num]}"
             print(f"The card in position {card_pos:2} is the {card_name}")
```

Create a deck and then print out that deck ← ①

```
[4]  # Cell 4
     deck = init_deck()
     print_deck(deck)
```

② ③

```
The card in position  0 is the Deuce of Clubs
The card in position  1 is the Three of Clubs
The card in position  2 is the Four of Clubs
The card in position  3 is the Five of Clubs
The card in position  4 is the Six of Clubs
The card in position  5 is the Seven of Clubs
The card in position  6 is the Eight of Clubs
```

④

Initializing a Deck of Cards

```
The card in position 0 is the Deuce of Clubs
The card in position 1 is the Three of Clubs
The card in position 2 is the Four of Clubs
The card in position 3 is the Five of Clubs
The card in position 4 is the Six of Clubs
The card in position 5 is the Seven of Clubs
The card in position 6 is the Eight of Clubs
The card in position 7 is the Nine of Clubs
The card in position 8 is the Ten of Clubs
The card in position 9 is the Jack of Clubs
The card in position 10 is the Queen of Clubs
The card in position 11 is the King of Clubs
The card in position 12 is the Ace of Clubs
The card in position 13 is the Deuce of Diamonds
The card in position 14 is the Three of Diamonds
The card in position 15 is the Four of Diamonds
The card in position 16 is the Five of Diamonds
The card in position 17 is the Six of Diamonds
The card in position 18 is the Seven of Diamonds
The card in position 19 is the Eight of Diamonds
The card in position 20 is the Nine of Diamonds
The card in position 21 is the Ten of Diamonds
The card in position 22 is the Jack of Diamonds
The card in position 23 is the Queen of Diamonds
The card in position 24 is the King of Diamonds
The card in position 25 is the Ace of Diamonds
The card in position 26 is the Deuce of Hearts
The card in position 27 is the Three of Hearts
```

Position Number	0	1	2	3	4
Card Number	0	1	2	3	4

Card Suit	
0	Clubs
1	Diamonds
2	Hearts
3	Spades

Card Rank	
0	Deuce
1	Three
2	Four
3	Five
4	Six
5	Seven
6	Eight
7	Nine
8	Ten
9	Jack
10	Queen
11	King
12	Ace

How can we randomly deal a deck of cards?

Run dealer_bogus.ipynb – **Cells 1…2**

Declare two string arrays: `suits` and `ranks` to store human-readable card identifiers

The index number of each element matches the encoding table in the slide deck

```
[1]  # Cell 1                                                    ①

     import numpy as np

     # fmt: off
     suits = ["Clubs", "Diamonds", "Hearts", "Spades"]

     ranks = ["Deuce", "Three", "Four", "Five", "Six", "Seven",
              "Eight", "Nine", "Ten", "Jack", "Queen", "King", "Ace"]
     # fmt: on
```

Define a function to randomly initialize a deck ②

```
[2]  # Cell 2
     def init_deck():                                           ③
         deck = np.arange(52)                                   ④
         for card_pos in range(52):                             ⑤
             card_num = np.random.randint(52)                   ⑥
             deck[card_pos] = card_num                          ⑦
         return deck                                            ⑧
```

Note: You should not edit this file!

Run dealer_bogus.ipynb – **Cells 3…4**

```
[3]  # Cell 3                                                    ①
     def print_deck(deck):
         for card_pos in range(52):
             card_num = deck[card_pos]
             suit_num = card_num // 13
             rank_num = card_num % 13
             card_name = f"{ranks[rank_num]} of {suits[suit_num]}"
             print(f"The card in position {card_pos:2} is the {card_name}")
```

Create a deck and then print out that deck ②

Initialize the numpy random number generator so we get the same deal ③

```
[4]  # Cell 4                                                    ④
     np.random.seed(2016)                                       ⑤
     deck = init_deck()                                         ⑥
     print_deck(deck)                                           ⑦
```

```
The card in position  0 is the Six of Spades
The card in position  1 is the Queen of Clubs
The card in position  2 is the Eight of Diamonds
The card in position  3 is the Deuce of Diamonds                ⑧
The card in position  4 is the Deuce of Spades
The card in position  5 is the Ten of Hearts
The card in position  6 is the King of Clubs
```

Randomizing a Deck of Cards?

```
The card in position 0 is the Six of Spades
The card in position 1 is the Queen of Clubs
The card in position 2 is the Eight of Diamonds
The card in position 3 is the Deuce of Diamonds
The card in position 4 is the Deuce of Spades
The card in position 5 is the Ten of Hearts
The card in position 6 is the King of Clubs
The card in position 7 is the Jack of Hearts
The card in position 8 is the Three of Spades
The card in position 9 is the Six of Spades
The card in position 10 is the Seven of Spades
The card in position 11 is the King of Clubs
The card in position 12 is the Three of Spades
The card in position 13 is the Deuce of Spades
The card in position 14 is the Four of Spades
The card in position 15 is the Ten of Hearts
The card in position 16 is the Jack of Diamonds
The card in position 17 is the Ace of Spades
The card in position 18 is the Queen of Hearts
The card in position 19 is the Nine of Spades
The card in position 20 is the Nine of Hearts
The card in position 21 is the Jack of Clubs
The card in position 22 is the Nine of Hearts
The card in position 23 is the Seven of Diamonds
The card in position 24 is the Five of Hearts
The card in position 25 is the Ten of Hearts
The card in position 26 is the Ten of Clubs
The card in position 27 is the Three of Hearts
```

```
The card in position 8 is the Three of Spades
The card in position 12 is the Three of Spades
The card in position 35 is the Three of Spades
```

Card Pos	0	1	...	3	4	...	9	...	13
Card Num	43	10		13	39		43		39
Card	6S	QC		2D	2S		6S		2S

```
def init_deck():
    deck = np.arange(52)
    for card_pos in range(52):
        card_num = np.random.randint(52)
        deck[card_pos] = card_num
    return deck
```

Random… but no repeats?

- How can we get a set of random numbers where no number is repeated until ***all*** numbers are picked at least once?
- Can we flag that a particular card # has already been dealt, and therefore not deal that card again?

Random… but no repeats?

- We need a ***helper*** array to store a **True** or **False** flag to record if a random trial card number has already been dealt
- Then we need to keep picking random trial card numbers until a card number is found that **has yet to be dealt**
- When that card is found, we **update the helper array** to record the fact that the card number has been dealt so it cannot be picked again
- Essentially, we equip the algorithm with a “memory” so it can avoid picking **duplicate** random card numbers

Instrumenting Your Code

- Instrumenting code is the process of taking accurate **timings** of the runtime performance of key algorithms within the program
- Python provides a **time** object that can measure the current CPU time of a running process to the nearest millisecond ($1/1000^{th}$ of a second) which is sufficient in most situations
- We bracket the code under analysis by measuring the clock immediately ***before*** the start and again ***after*** the end of the algorithm to calculate the **elapsed** time
- Careful tracking of code timings will provide objective empirical evidence if changes to algorithms and/or data structures are indeed making the program more efficient

Run dealer_slow.ipynb – **Cell 1**

Note: You should not edit this file!

Declare two string arrays: `suits` and `ranks` to store human-readable card identifiers

The index number of each element matches the encoding table in the slide deck

```
[1]  # Cell 1  ← ①

     import numpy as np

     # fmt: off
     suits = ["Clubs", "Diamonds", "Hearts", "Spades"]

     ranks = ["Deuce", "Three", "Four", "Five", "Six", "Seven",
              "Eight", "Nine", "Ten", "Jack", "Queen", "King", "Ace"]
     # fmt: on
```

Run dealer_slow.ipynb – **Cell 2**

Define a function to randomly initialize a deck ← ①

Prevent duplicate cards by maintaining a helper `Boolean` array ← ②

This array can track if a specific card number has already been dealt ← ③

```
[2]  # Cell 2
     def init_deck():                                    ← ④
         deck = np.arange(52)
         already_dealt = np.zeros(52, dtype=bool)        ← ⑤
         for card_pos in range(52):
             card_num = np.random.randint(52)
             while already_dealt[card_num]:              ← ⑥
                 card_num = np.random.randint(52)        ← ⑦
             deck[card_pos] = card_num
             already_dealt[card_num] = True              ← ⑧
         return deck
```

Run dealer_slow.ipynb – **Cell 3**

Define a function to print a deck of cards

We must convert a `card number` to the specific suit # and rank # for that card number

```
[3]  # Cell 3
     def print_deck(deck):
         for card_pos in range(52):
             card_num = deck[card_pos]
             suit_num = card_num // 13
             rank_num = card_num % 13
             card_name = f"{ranks[rank_num]} of {suits[suit_num]}"
             print(f"The card in position {card_pos:2} is the {card_name}")
```

①

Run dealer_slow.ipynb – **Cell 4**

Time how long it takes to deal 10,000 random decks ①

Initialize the numpy random number generator so we get the same deal

Print the last deal to visually confirm there are no duplicates

```
[4]  # Cell 4
     import time                                          ②

     np.random.seed(2016)                                 ③
     total_deals = 10_000

     start_time = time.perf_counter()                     ④
     for _ in range(total_deals):                         ⑤
         deck = init_deck()
     elapsed_time = time.perf_counter() - start_time      ⑥

     print(f"Total deals: {total_deals:,}")               ⑦
     print(f"Total run time (sec): {elapsed_time:.3f}")
     print()

     print_deck(deck)                                     ⑧

     Total deals: 10,000
     Total run time (sec): 16.889                         ⑨

     The card in position  0 is the Five of Clubs
     The card in position  1 is the Ace of Hearts
     The card in position  2 is the Three of Spades
     The card in position  3 is the Jack of Spades
     The card in position  4 is the King of Clubs
     The card in position  5 is the Five of Diamonds
     The card in position  6 is the Nine of Hearts
```

No repeated cards!

Correct but inefficient…

dealer_bogus.ipynb

```
def init_deck():
    deck = np.arange(52)
    for card_pos in range(52):
        card_num = np.random.randint(52)
        deck[card_pos] = card_num
    return deck
```

dealer_slow.ipynb

```
def init_deck():
    deck = np.arange(52)
    already_dealt = np.zeros(52, dtype=bool)
    for card_pos in range(52):
        card_num = np.random.randint(52)
        while already_dealt[card_num]:
            card_num = np.random.randint(52)
        deck[card_pos] = card_num
        already_dealt[card_num] = True
    return deck
```

A Fast Dealer

- There is an inherent inefficiency in the naïve algorithm employed in the current **init_deck**() function
- It takes **longer and longer**, as more cards are dealt, to randomly pick (**find**) a card that has not yet been dealt
- We need to discover an algorithm that, while ensuring every card is dealt only once, doesn't lose time at the end of the deal searching for ***that one remaining card*** that has not yet been dealt
- The improved algorithm doesn’t need an **already_dealt** helper **array**, and a **7th grader discovered it!**

Run dealer_fast.ipynb – **Cells 1…2**

Declare two string arrays: `suits` **and** `ranks` **to store human-readable card identifiers**

The index number of each element matches the encoding table in the slide deck

```
[1]  # Cell 1

     import numpy as np

     # fmt: off
     suits = ["Clubs", "Diamonds", "Hearts", "Spades"]

     ranks = ["Deuce", "Three", "Four", "Five", "Six", "Seven",
              "Eight", "Nine", "Ten", "Jack", "Queen", "King", "Ace"]
     # fmt: on
```

① ←

Define a function to randomly initialize a deck ← ②

How does this approach avoid duplicates without a helper array? ← ③

```
[2]  # Cell 2
     def init_deck():
         deck = np.arange(52)
         for card_pos in range(52):
             new_card_pos = np.random.randint(52)
             swap_card = deck[card_pos]
             deck[card_pos] = deck[new_card_pos]
             deck[new_card_pos] = swap_card
         return deck
```

④ ← ⑤ ← ⑥ ← ⑦ ← ⑧ ← ⑨

Note: You should not edit this file!

Run dealer_fast.ipynb – **Cell 3**

Define a function to print a deck of cards

We must convert a `card number` to the specific suit # and rank # for that card number

```
[3]  # Cell 3
     def print_deck(deck):
         for card_pos in range(52):
             card_num = deck[card_pos]
             suit_num = card_num // 13
             rank_num = card_num % 13
             card_name = f"{ranks[rank_num]} of {suits[suit_num]}"
             print(f"The card in position {card_pos:2} is the {card_name}")
```

①

Run dealer_fast.ipynb – **Cell 4**

Time how long it takes to deal 10,000 random decks

Initialize the numpy random number generator so we get the same deal

Print the last deal to visually confirm there are no duplicates

```
[8]  # Cell 4
     import time

     np.random.seed(2016)
     total_deals = 10_000

     start_time = time.perf_counter()
     for _ in range(total_deals):
         deck = init_deck()
     elapsed_time = time.perf_counter() - start_time

     print(f"Total deals: {total_deals:,}")
     print(f"Total run time (sec): {elapsed_time:.3f}")
     print()

     print_deck(deck)
```

① ②

```
Total deals: 10,000
Total run time (sec): 2.223

The card in position  0 is the Jack of Clubs
The card in position  1 is the Deuce of Hearts
The card in position  2 is the Nine of Hearts
The card in position  3 is the Four of Diamonds
The card in position  4 is the Nine of Clubs
The card in position  5 is the Eight of Hearts
The card in position  6 is the Seven of Spades
```

☺

No repeated cards!

A Fast Dealer

- Consider the revised **init_deck()** function:

```
def init_deck():
    deck = np.arange(52)
    for card_pos in range(52):
        new_card_pos = np.random.randint(52)
        swap_card = deck[card_pos]
        deck[card_pos] = deck[new_card_pos]
        deck[new_card_pos] = swap_card
    return deck
```

- What is going on in this function that ensures no duplicate cards are dealt **and** doesn't waste time trying to find the cards at the end that have not yet been dealt?

Slow vs. Fast Dealer

dealer_slow.ipynb

```
def init_deck():
    deck = np.arange(52)
    already_dealt = np.zeros(52, dtype=bool)
    for card_pos in range(52):
        card_num = np.random.randint(52)
        while already_dealt[card_num]:
            card_num = np.random.randint(52)
        deck[card_pos] = card_num
        already_dealt[card_num] = True
    return deck
```

```
Total deals: 10,000
Total run time (sec): 16.889
```

dealer_fast.ipynb

```
def init_deck():
    deck = np.arange(52)
    for card_pos in range(52):
        new_card_pos = np.random.randint(52)
        swap_card = deck[card_pos]
        deck[card_pos] = deck[new_card_pos]
        deck[new_card_pos] = swap_card
    return deck
```

```
Total deals: 10,000
Total run time (sec): 2.223
```

- Fewer lines of code
- No helper list needed
- ~ **600%** faster
- Discovered by a 7th grader

Computing is a **New** Science

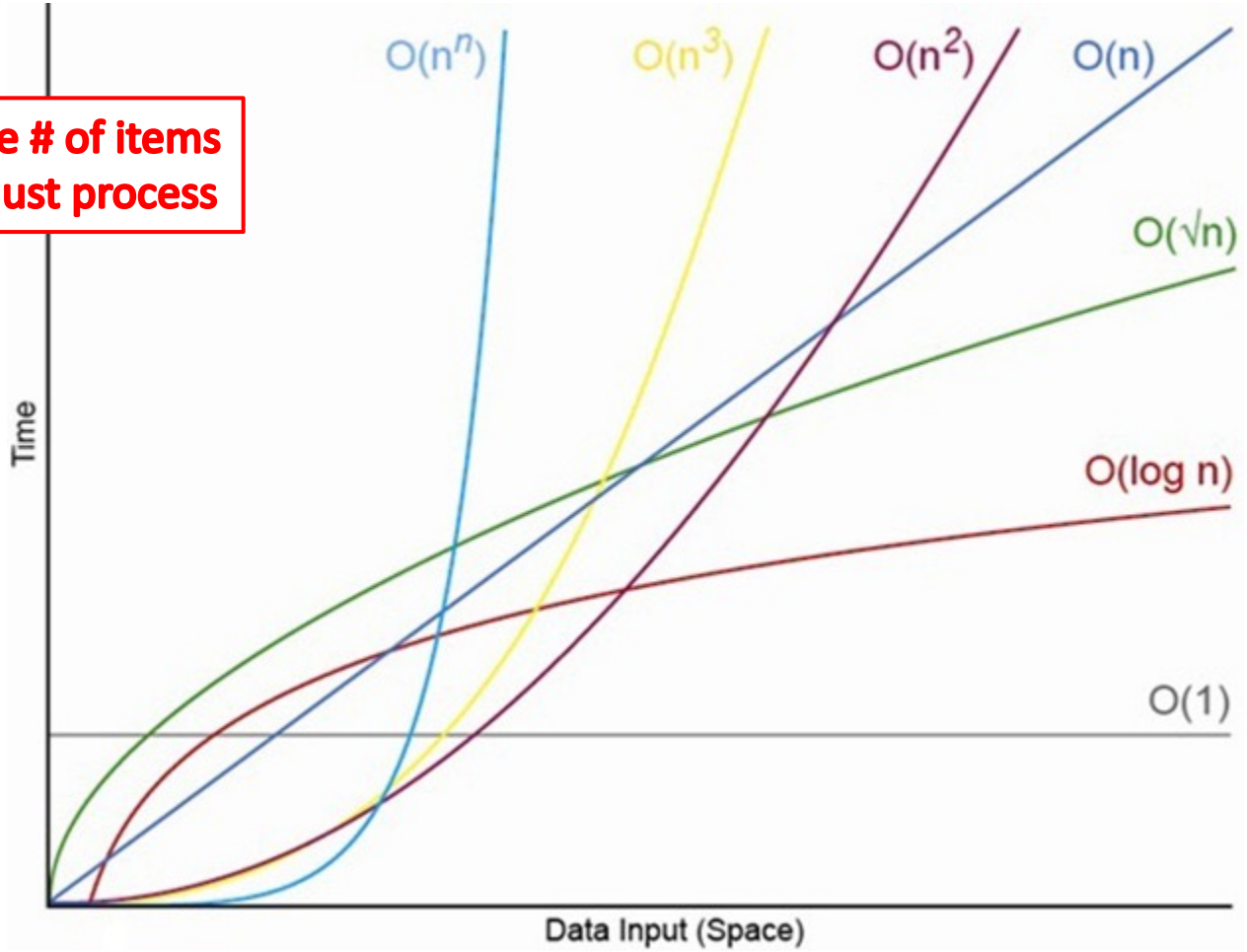
- The best algorithms are the ones that leave you scratching your head thinking "**…that was so obvious – why didn't I think of that?**"
  - They are often **the shortest** algorithms in terms of source code length (but not always)
  - They are also normally **the fastest** algorithms to execute
- For many algorithms, we have yet to discover the provably optimal approach – there is still so much unknown
- Even students taking an initial computer science course can get a flash of inspiration and see something in a new way!

Algorithmic Efficiency

- Scientific computing often involves analyzing large data sets or running large-scale simulations
- It is essential to have code that runs as fast as possible while returning the correct results
- We measure algorithm efficiency by estimating the impact on the **total run time** as the **size of the input data increases**
- We are only interested in the principal (highest exponent) term, which describes the overall "**order**" of the algorithm, as we are not trying to calculate the *exact* run time
- The order of an algorithm is expressed in "**Big O**" notation
- The optimal algorithms have the smallest possible order

Algorithmic Efficiency

n is the # of items you must process



increasing n # of items →

Session **06** – Now You Know…

- The art of computer *science* is finding an efficient way to represent **everything as a number**
- An encoding must be **unambiguous** to support proper decoding
- Decoding multiple things from a single number often requires the use of the **modulus** **%** operator
- We can **instrument** (time) code by measuring the **elapsed** time between when it starts and ends
- Algorithm *design* remains an area of very active research and even new programmers can make a **novel** contribution

Capstone Project 1

Create your own "game" in Python. The game can be about anything, but it should utilize a minimum of two of the concepts we've covered so far.

The "main character" in the game can either choose their actions based on user input, or can follow some "strategy" (for instance, the character might be a gambler, and chooses random actions, or they might be more conservative, and choose safe actions 90% of the time).

For example, your game might include many if/else statements, and also write a log of the "character's" actions to disk in a log.txt file or something along those lines.