DSC 40B

Lecture 5-6 : Best, Worst, Average

Agenda

Plan for the lecture

- Best, Worst and Average cases.

The Movie problem

The Movie Problem

The Movie Problem

- **Given**: an array movies of movie durations, and the flight duration t
- **Find**: two movies whose durations add to t
  - If no two movies sum to t, return **None**.

Exercise

- Design a brute force solution to the problem. What is its time complexity?

Brute force

```
def find_movies(movies, t):
    n = len(movies)
    for i in range(n):
        for j in range(i + 1, n):
            if movies[i] + movies[j] == t:
                return (i, j)
    return None
```

Time Complexity

- It looks like there is a **best** case and **worst** case.
- How do we formalize this?

For the future...

- Can you come up with a better algorithm?
- What is the *best possible* time complexity?

Best and Worst Cases

Definition

- Define $T_{\text{best}}(n)$ to be the **least** time taken by the algorithm on **any input of size n.**

- The asymptotic growth of $T_{\text{best}}(n)$ is the algorithm's **best case time complexity.**

Example 1: mean

```
def mean(arr):
    total = 0
    for x in arr:
        total += x
    return total / len(arr)
```

Example 1: mean

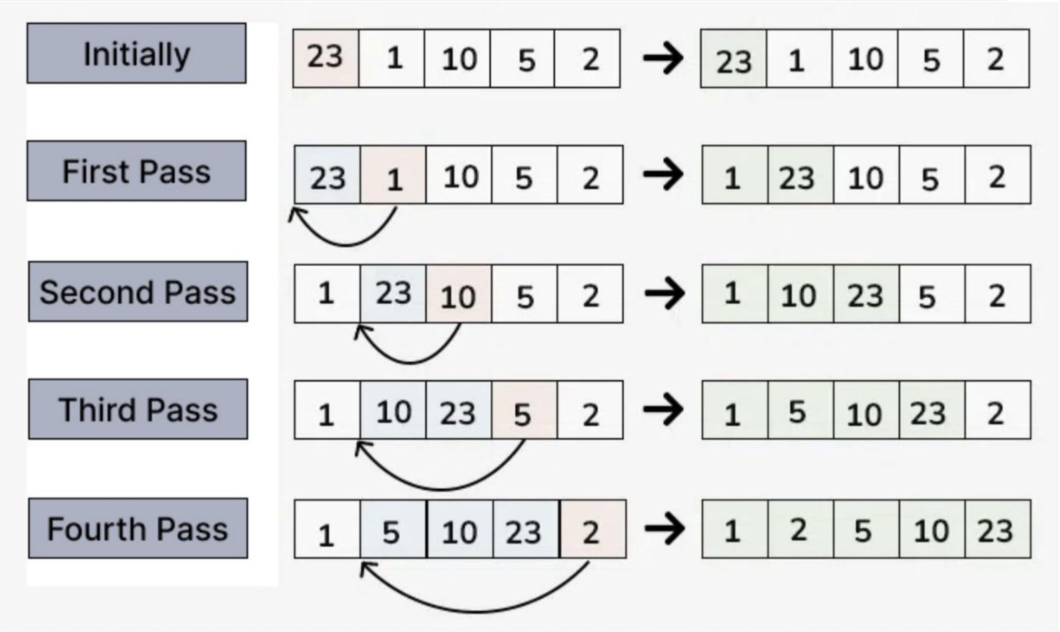
```
def mean(arr):
    total = 0
    for x in arr:
        total += x
    return total / len(arr)
```

$T_{best}(n) = n$

Caution!

- The best case is never: “the input is of size one”.
- The best case is about the **structure** of the input, not its **size**.
- Not always constant time!
    - **Example**: sorting.

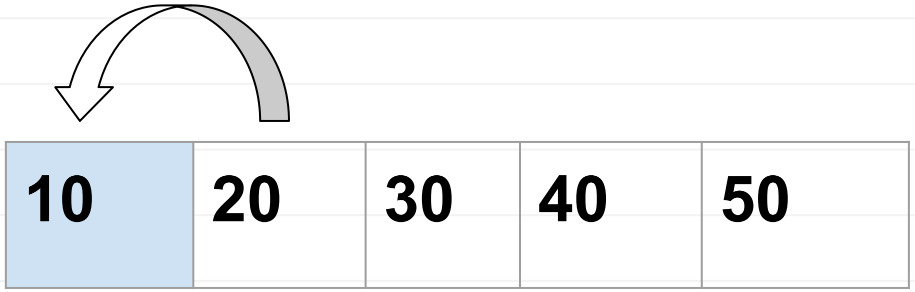
```python
def insertionSort(arr):
    for i in range(1, len(arr)):
        key = arr[i]
        j = i - 1
        while j >= 0 and key < arr[j]:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = key
```



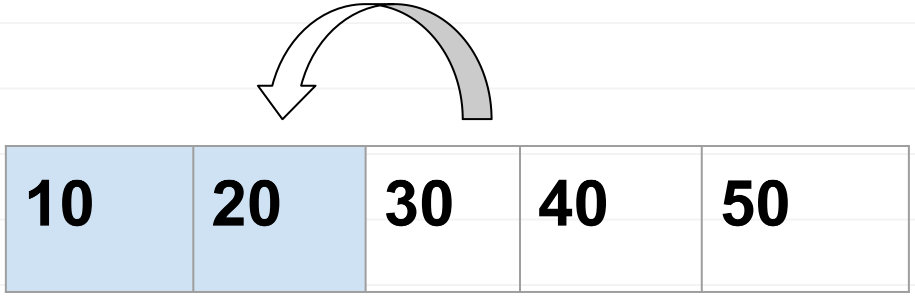
```python
def insertionSort(arr):
    for i in range(1, len(arr)):
        key = arr[i]
        j = i - 1
        while j >= 0 and key < arr[j]:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = key
```

10	20	30	40	50

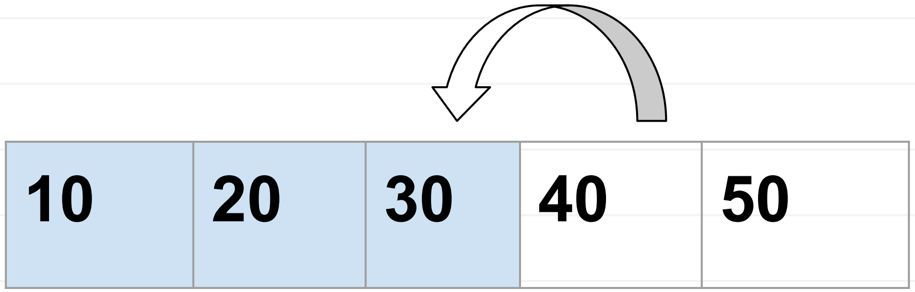
```python
def insertionSort(arr):
    for i in range(1, len(arr)):
        key = arr[i]
        j = i - 1
        while j >= 0 and key < arr[j]:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = key
```



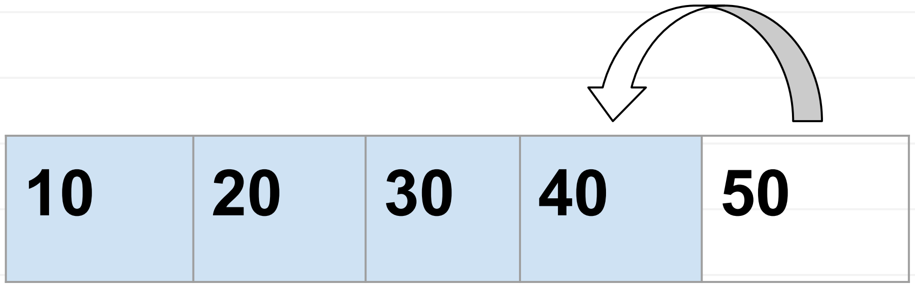
```python
def insertionSort(arr):
    for i in range(1, len(arr)):
        key = arr[i]
        j = i - 1
        while j >= 0 and key < arr[j]:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = key
```



```python
def insertionSort(arr):
    for i in range(1, len(arr)):
        key = arr[i]
        j = i - 1
        while j >= 0 and key < arr[j]:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = key
```



```python
def insertionSort(arr):
    for i in range(1, len(arr)):
        key = arr[i]
        j = i - 1
        while j >= 0 and key < arr[j]:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = key
```



```python
def insertionSort(arr):
    for i in range(1, len(arr)):
        key = arr[i]
        j = i - 1
        while j >= 0 and key < arr[j]:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = key
```

$T_{best}(n) = n$

10	20	30	40	50

Time Complexity of mean

- Linear time, $\Theta(n)$.
- Depends **only** on the array's **size**, n, not on its actual elements.

Example 2: Linear Search

- **Given**: an array arr of numbers and a target t.
- **Find**: the index of t in arr, or None if it is missing.
- Example: arr = [-3, -6, 7, 3, 0, 15, 4]

```python
def linear_search(arr, t):
    for i, x in enumerate(arr):
        if x == t:
            return i
    return None
```

Exercise: Time complexity?

```
def linear_search(arr, t):
    for i, x in enumerate(arr):
        if x == t:
            return i
    return None
```

Observation

- It looks like there are two extreme cases...

The Best Case

- When the target, t, is the very first element.
- The loop exits after one iteration.
- $\Theta(1)$ time?

The Worst Case

- When the target, t, is not in the array at all.
- The loop exits after n iterations.
- $\Theta(n)$ time?

Time Complexity

- `linear_search` can take vastly different amounts of time on two inputs of the same size.
    - Depends on **actual elements** as well as size.
- It has no single, overall time complexity.
- Instead we'll report **best** and **worst** case time complexities.

Best Case Time Complexity

- How does the time taken in the **best case** grow as the input gets larger?

Best Case

- In linear_search's **best case**, $T_{best}(n)$ = $\boldsymbol{c}$, no matter how large the array is.
- The **best case time complexity** is Θ(1).

Worst Case Time Complexity

- How does the time taken in the **worst case** grow as the input gets larger?

Definition

- Define $T_{\text{worst}}(n)$ to be the **most time** taken by the algorithm on any input of size n.

- The asymptotic growth of $T_{\text{worst}}(n)$ is the algorithm's **worst case time complexity.**

Worst Case

- In the worst case, `linear_search` iterates through the entire array.
- The **worst case** time complexity is $\Theta(n)$.

Exercise: times: Best and Worst

```
def func(arr):
   n = len(arr)
   for x in arr:
      for y in arr:
         x + y == 10
            return sum(arr)
```

A: $\Theta(1)$

B: $\Theta(n)$

C: $\Theta(n^2)$

D: $\Theta(n^3)$

Best Case

- Best case occurs when the first element is 5: 5 + 5 = 10
- `sum(arr)` takes $\Theta(n)$ time
- Exists, taking $\Theta(n)$ time in total

Worst Case

- Worst case occurs when no two numbers add to 10.
- Has to loop over all $\Theta(n^2)$ pairs.
- Worst case time complexity: $\Theta(n^2)$.
- Note: Not $\Theta(n^3)$ since the `sum(arr)` only runs once.

Note

- An algorithm like `linear_search` **doesn't** have one single time complexity.
- An algorithm like `mean` **does**, since the best and worst case time complexities coincide.

Main Idea

Reporting **best** and **worst** case time complexities gives us a richer of the performance of the algorithm.

Average Case

Time Taken, Typically

- Best case and worst case can be **misleading**.
    - Depend on a single **good**/**bad** input.
- How much time is taken, typically?
- **Idea**: compute the average time taken over *all possible inputs.*

Recall: The Expectation

- The **expected value** of a random variable X is:

$$\sum_{x} x \cdot P(X = x)$$

winnings	probability
$ 0	50%
$ 1	30%
$ 10	18%
$ 50	2%

Expected winnings:

Recall: The Expectation

- The **expected value** of a random variable X is:

$$\sum_x x \cdot P(X = x)$$

winnings	probability
$ 0	50%
$ 1	30%
$ 10	18%
$ 50	2%

$0 x .5 +

Expected winnings:

Recall: The Expectation

- The **expected value** of a random variable X is:

$$\sum_x x \cdot P(X = x)$$

winnings	probability
$ 0	50%
$ 1	30%
$ 10	18%
$ 50	2%

Expected winnings:

$0 x .5 + $1 x .3

Recall: The Expectation

- The **expected value** of a random variable X is:

$$\sum_x x \cdot P(X = x)$$

winnings	probability
$ 0	50%
$ 1	30%
$ 10	18%
$ 50	2%

Expected winnings:

$0 x .5 + $1 x .3 + $10 x .18 +

Recall: The Expectation

- The **expected value** of a random variable X is:

$$\sum_x x \cdot P(X = x)$$

winnings	probability
\$ 0	50%
\$ 1	30%
\$ 10	18%
\$ 50	2%

Expected winnings:

\$0 x .5 + \$1 x .3 + \$10 x .18 + \$50 x .02

Recall: The Expectation

- The **expected value** of a random variable X is:

$$\sum_x x \cdot P(X = x)$$

winnings	probability
$ 0	50%
$ 1	30%
$ 10	18%
$ 50	2%

Expected winnings:

$0 x .5 + $1 x .3 + $10 x .18 + $50 x .02 = $3.10

Average Case

- We'll compute the expected time over all cases:

$$T_{avg}(n) = \sum_{case \in \text{all cases}} P(case) \cdot T(case)$$

- Called the **average case time complexity.**

Strategy for Finding Average Case

- **Step 0**: Make assumption about distribution of inputs.
- **Step 1**: Determine the possible cases.
- **Step 2**: Determine the probability of each case.
- **Step 3**: Determine the time taken for each case.
- **Step 4**: Compute the expected time (average).

Example: Linear Search

- **Best? Worst?**

```
def linear_search(arr, t):
    for i, x in enumerate(arr):
        if x == t:
            return i
    return None
```

Example: Linear Search

- What is the average case time complexity of linear search?

Step 0: Assume input distribution

- We must assume something about the input.
- **Example**: Target must be in array, equally-likely to be any element, no duplicates.
- This is *usually* given to you.

Step 1: Determine the Cases

Example: *linear search.*

- **Case 1**: target is first element
- **Case 2**: target is second element

  ⋮

- **Case n**: target is nth element
- ***Case n + 1***: *target is not in array (but not needed due to assumptions)*

Step 2: Case Probabilities

- What is the probability that we see each case?
  - **Example**: what is the probability that the target is the kth element?
- This is where we use assumptions from **Step 0**.

Example

- **Assume**: target is in the array exactly once, equally-likely to be any element.
- Each case has probability $1/n$.

Step 3: Case Times

- Determine time taken in each case.
- **Example**: linear search.
  - Let's say it takes time c per iteration.

  Case 1: time c

  Case 2: time $2c$

  $\vdots$

  Case i: time $c \cdot i$

  $\vdots$

  Case n: time $c \cdot$ n

Step 4: Compute Expectation

$$T_{avg}(n) = \sum_{i=1}^{n} P(\text{case } i) \cdot T(\text{case } i)$$

Average Case Time Complexity

- The **average case** time complexity of **linear search** is $\Theta(n)$.
  - Under these assumptions on the input!

Note

- **Worst case** time complexity is still useful.
- Easier to calculate.
- Often same as average case (but not always!)
- Sometimes worst case is very important.
  - Real time applications, time complexity attacks

Note

- **Hard** to make realistic assumptions on input distribution.
- **Example**: linear search.
  - Is it realistic to assume t is in array?
  - If not, what is the probability that it *is* in the array?

Exercise

- Suppose we *change* our assumptions:
  - The target has a 50% chance of being in the array.
- If it is in the array, it is equally-likely to be any element.
- What is the average case complexity now?

Average Case in Movie Problem

Recall: The Movie Problem

- **Given**: an array movies of movie durations, and the flight duration t
- **Find**: two movies whose durations add to t.
  - If no two movies sum to t, return None.

```python
def find_movies(movies, t):
    n = len(movies)
    for i in range(n):
        for j in range(i + 1, n):
            if movies[i] + movies[j] == t:
                return (i, j)
    return None
```

Time Complexity

- **Best case**: $\Theta(1)$
  - When the first pair of movies checked equals target.
- **Worst case**: $\Theta(n^2)$
  - When no pair of movies equals target.

"Average" Case?

- The best and worst cases are extremes.
- How much time is taken, *typically*?
  - That is, when the target pair is not the first checked nor the last, but somewhere in the middle.

Exercise

- How much time do you expect *find_movies* to take on a typical input?

A: $\Theta(1)$

B: $\Theta(n^2)$

C: Something in between, like $\Theta(n)$

The Movie Problem

```
def find_movies(movies, t):
    n = len(movies)
    for i in range(n):
        for j in range(i + 1, n):
            if movies[i] + movies[j] == t:
                return (i, j)
    return None
```

Step 0: Assume input distribution

- Suppose we are told that:
  - There is a **unique** pair of movies that add to t.
  - All pairs are **equally likely**.

Step 1: Determine the Cases

- Case α: the αth pair checked sums to t.
- Each pair of movies is a case.
- There are $\binom{n}{2}$ cases (pairs of movies)

Step 2: Case Probabilities

- **Assume**: there is a unique pair that adds to t.
- **Assume**: all pairs are equally likely.
- Probability of any case: $\frac{1}{\binom{n}{2}} = \frac{2}{n(n-1)}$

Step 3: Case Time

- How much time is taken for a particular case?
- Example, suppose the movies a and b sum to the target.
- How long does it take to find this pair?

Exercise

Roughly how much time is taken (how many times does line 5 run) if the αth pair checked sums to the target?

```
1 def find_movies(movies, t):
2     n = len(movies)
3     for i in range(n):
4         for j in range(i + 1, n):
5             if movies[i] + movies[j] == t:
6                 return (i, j)
7     return None
```

(m_1, m_2)

Exercise

```
def find_movies(movies, t):
    n = len(movies)
    for i in range(n):
        for j in range(i + 1, n):
            if movies[i] + movies[j] == t:
                return (i, j)
    return None
```

α	T

(m_1, m_2)

Exercise

```
def find_movies(movies, t):
    n = len(movies)
    for i in range(n):
        for j in range(i + 1, n):
            if movies[i] + movies[j] == t:
                return (i, j)
    return None
```

α	T
1	1

(m_1, m_3)

Exercise

```python
def find_movies(movies, t):
    n = len(movies)
    for i in range(n):
        for j in range(i + 1, n):
            if movies[i] + movies[j] == t:
                return (i, j)
    return None
```

α	T
1	1
2	2

(m_1, m_4)

Exercise

```python
def find_movies(movies, t):
    n = len(movies)
    for i in range(n):
        for j in range(i + 1, n):
            if movies[i] + movies[j] == t:
                return (i, j)
    return None
```

α	T
1	1
2	2
3	3

Exercise

```python
def find_movies(movies, t):
    n = len(movies)
    for i in range(n):
        for j in range(i + 1, n):
            if movies[i] + movies[j] == t:
                return (i, j)
    return None
```

α	T
1	1
2	2
3	3
4	4

Exercise

```
1 def find_movies(movies, t):
2     n = len(movies)
3     for i in range(n):
4         for j in range(i + 1, n):
5             if movies[i] + movies[j] == t:
6                 return (i, j)
7     return None
```

α	T
1	1
2	2
3	3
4	4

Roughly how much time is taken (how many times does line 5 run) if the αth pair checked sums to the target? **T(case α) = α**

Step 4: Compute Expectation

$$T_{avg} =$$

Step 4: Compute Expectation

$$T_{avg} = \sum_{\alpha = 1}$$

Step 4: Compute Expectation

$$T_{avg} = \sum_{\alpha = 1}^{\binom{n}{2}}$$

Step 4: Compute Expectation

$$T_{avg} = \sum_{\alpha=1}^{\binom{n}{2}} P(case\ \alpha)$$

Step 4: Compute Expectation

$$T_{avg} = \sum_{\alpha = 1}^{\binom{n}{2}} P(case\ \alpha) \cdot T(case\ \alpha)$$

$$T_{avg} = \sum_{\alpha=1}^{\binom{n}{2}} P(case\ \alpha) \cdot T(case\ \alpha)$$

$$\sum_{\alpha=1}^{\binom{n}{2}} \cdot \frac{1}{\binom{n}{2}} \cdot T(case\ \alpha)$$

$$T_{avg} = \sum_{\alpha=1}^{\binom{n}{2}} P(case\ \alpha) \cdot T(case\ \alpha)$$

$$\sum_{\alpha=1}^{\binom{n}{2}} \cdot \frac{1}{\binom{n}{2}} \cdot T(case\ \alpha) = \sum_{\alpha=1}^{\binom{n}{2}} \cdot \frac{1}{\binom{n}{2}} \cdot \alpha$$

$$\sum_{\alpha=1}^{\binom{n}{2}} \cdot \frac{1}{\binom{n}{2}} \cdot \alpha = \frac{1}{\binom{n}{2}} \cdot \sum_{\alpha=1}^{\binom{n}{2}} \alpha$$

$$\sum_{\alpha=1}^{\binom{n}{2}} \cdot \frac{1}{\binom{n}{2}} \cdot \alpha = \frac{1}{\binom{n}{2}} \cdot \sum_{\alpha=1}^{\binom{n}{2}} \alpha$$

$$\sum_{\alpha=1}^{\binom{n}{2}} \alpha = \sum_{\alpha=1}^{t} \alpha = \frac{t(t+1)}{2}$$

$$\sum_{\alpha=1}^{\binom{n}{2}} \cdot \frac{1}{\binom{n}{2}} \cdot \alpha = \frac{1}{\binom{n}{2}} \cdot \sum_{\alpha=1}^{\binom{n}{2}} \alpha$$

$$\sum_{\alpha=1}^{\binom{n}{2}} \alpha = \sum_{\alpha=1}^{t} \alpha = \frac{t(t+1)}{2} = \frac{\binom{n}{2}\left[\binom{n}{2}+1\right]}{2}$$

$$\sum_{\alpha=1}^{\binom{n}{2}} \cdot \frac{1}{\binom{n}{2}} \cdot \alpha = \frac{1}{\binom{n}{2}} \cdot \sum_{\alpha=1}^{\binom{n}{2}} \alpha$$

$$\sum_{\alpha=1}^{\binom{n}{2}} \alpha = \sum_{\alpha=1}^{t} \alpha = \frac{t(t+1)}{2} = \frac{\binom{n}{2}\left[\binom{n}{2}+1\right]}{2}$$

$$\binom{n}{2} = \frac{n!}{2!\,(n-2)!} = \frac{n(n-1)}{2} = \Theta(n^2)$$

$$\binom{n}{2} = \frac{n!}{2!\,(n-2)!} = \frac{n(n-1)}{2} = \Theta(n^2)$$

$$\frac{\binom{n}{2}\left[\binom{n}{2}+1\right]}{2} = \Theta(?)$$

$$\binom{n}{2} = \frac{n!}{2!\,(n-2)!} = \frac{n(n-1)}{2} = \Theta(n^2)$$

$$\frac{\binom{n}{2}\left[\binom{n}{2}+1\right]}{2} = \Theta(n^4)$$

$$\frac{1}{\binom{n}{2}} \cdot \sum_{\alpha = 1}^{\binom{n}{2}} \alpha$$

$$\frac{1}{\binom{n}{2}} \cdot \sum_{\alpha = 1}^{\binom{n}{2}} \alpha = \frac{1}{\binom{n}{2}} \cdot \frac{\binom{n}{2}\left[\binom{n}{2}+1\right]}{2}$$

$$\frac{1}{\binom{n}{2}} \cdot \sum_{\alpha=1}^{\binom{n}{2}} \alpha = \frac{1}{\binom{n}{2}} \cdot \frac{\binom{n}{2}\left[\binom{n}{2}+1\right]}{2}$$

$$= \frac{1}{\binom{n}{2}} \cdot \Theta(n^4)$$

$$\frac{1}{\binom{n}{2}} \cdot \sum_{\alpha=1}^{\binom{n}{2}} \alpha = \frac{1}{\binom{n}{2}} \cdot \frac{\binom{n}{2}\left[\binom{n}{2}+1\right]}{2}$$

$$= \frac{1}{\binom{n}{2}} \cdot \Theta(n^4)$$

$$= \frac{1}{\Theta(n^2)} \cdot \Theta(n^4)$$

$$\frac{1}{\binom{n}{2}} \cdot \sum_{\alpha=1}^{\binom{n}{2}} \alpha = \frac{1}{\binom{n}{2}} \cdot \frac{\binom{n}{2}\left[\binom{n}{2}+1\right]}{2}$$

$$= \frac{1}{\binom{n}{2}} \cdot \Theta(n^4)$$

$$= \frac{1}{\Theta(n^2)} \cdot \Theta(n^4)$$

$$= \Theta(n^2)$$

Average Case

- The average case time complexity of find_movies is $\Theta(n^2)$.
- Same as the **worst** case!

Note

- We've seen two algorithms where the average case = the worst case.
- Not *always* the case!
- Interpretation: the worst case is not too extreme.

Thank you!

Do you have any questions?

CampusWire!