# Computational complexity

## Breakout task: data generation

Generate a `list` of 30 random chromosomes, with replacement

hint: use the standard library: `import random`

# Data generation: solution

```
import random
NUM_CHROMOSOMES = 23
num_samples = 30
chromosomes = random.choices(range(1, NUM_CHROMOSOMES + 1), k=num_samples)
```

## Algorithm 1

Write a function to determine if a specific integer is included. Don't use `in`.

# Algorithm 1: solution

```
def is_in(things: list[int], query: int) -> bool:
    for thing in things:
        if thing == query:
            return True
    return False
```

# Algorithm 1: analysis

`N` is the length of the list (`num_samples`).

```python
def is_in(things: list[int], query: int) -> bool:
    for thing in things:  # N times
        if thing == query:  # O(1)
            return True  # O(1)
    return False  # O(1)
```

`N * (1 + 1) + 1 → O(N)`

## Algorithm 2

Write a function to de-duplicate. Don't use a `set` or `dict`.

# Algorithm 2: solution

```
def dedup(things):
    unique_things = []
    for thing in things:
        if not is_in(unique_things, thing):
            unique_things.append(thing)
    return unique_things
```

# Algorithm 2: analysis

```
def dedup(things):
    unique_things = []  # O(1)
    for thing in things:  # N times
        if not is_in(unique_things, thing):  # O(N)
            unique_things.append(thing)  # O(1)
    return unique_things  # O(1)
```

`1 + N * (N * (1 + 1)) → O(N**2)`

## Algorithm 3

Write a function to determine if a subset exists that sums to 20.

# Algorithm 3: solution

```
def subset_sum(samples: list[int], target: int) -> bool:
    num_samples = len(samples)
    num_permutations = 2**num_samples
    for permutation in range(num_permutations):
        total = 0
        for idx in reversed(range(num_samples)):
            if permutation > 2**idx:
                permutation -= 2**idx
                total += samples[idx]
        if total == target:
            return True
    return False
```

# Algorithm 3: analysis

```python
def subset_sum(samples: list[int], target: int) -> bool:
    num_samples = len(samples)  # O(1)
    num_permutations = 2**num_samples  # O(1)
    for permutation in range(num_permutations):  # 2**N times
        total = 0  # O(1)
        for idx in reversed(range(num_samples)):  # N times
            if permutation > 2**idx:
                permutation -= 2**idx  # O(1)
                total += chromosomes[idx]  # O(1)
        if total == target:  # O(1)
            return True  # O(1)
    return False  # O(1)
```

```
1 + 1 + 2**N * (1 + N * (1 + 1) + 1) + 1 → O(N * 2**N)
```