1. Prove $n^2 \in \mathcal{O}(n^3)$:
   $\lim_{n\to\infty} \frac{n^2}{n^3} = 0$
   $0 < \infty$ therefore by limit definition of asymptotic relations $n^2 \in \mathcal{O}(n^3)$

   Prove $n^3 \notin \mathcal{O}(n^2)$:
   $\lim_{n\to\infty} \frac{n^3}{n^2} = \infty$
   $\infty \not< \infty$ therefore by limit definition of asymptotic relations $n^3 \notin \mathcal{O}(n^2)$

2. Find the time complexities of the following algorithms

   (a) LinearSearch-1
   Undefined. Program never terminates.

   (b) LinearSearch-2

$$\begin{aligned}\text{Time} &= \sum_{i=1}^{n-1}(\text{time}(\textbf{return } i) + \text{time}(i \leftarrow i+1) + \text{time}(\textbf{return } -1)) \\ &= \sum_{i=1}^{n-1}(c_1 + c_2 + c_3) \\ &= (c_1 + c_2 + c_3)\sum_{i}^{n-1} 1 \\ &\in \Theta(n)\end{aligned}$$

   (c) Factorial
   Undefined. No base case for $n = 1$.

3. (i)

```
FindRepeatedNumberNaive(A[1...n])
for i ← 1 to n do
    for j ← i to n do
        if A[i] = A[j] then
            return A[i]
return −1
```

   (ii)

```
FindRepeatedNumberEfficient(A[1...n])
Create an array found[1...(n − 1)]
for i ← 1 to n do
    value ← A[i]
    if found[value] = True then
        return value
    found[value] ← True
return −1
```

4.

(i)

```
GroupingNaive(A[1...n])
remove_index ← 1
for i ← 1 to n do
    element ← A[i]
    if IsPerfectSquare(element) then
        remove_index ← i
        for j ← remove − 1 down to 0 do
            A[j + 1] ← A[j]
        A[1] ← element
```

(ii)

```
GroupingBest(A[1...n])
left ← 1; right ← n
while left < right do
    a ← A[left]; b ← A[right]
    asq ← IsPerfectSquare(a); bsq ← IsPerfectSquare(b)
    if not asq and bsq then
        Swap(a, b)
    if asq then
        left ← left + 1
    if not bsq then
        right ← right − 1
```

5.

(i)

```
JosephusProblemArray(n, k, j)
Create an array people[i...n] ← [i...n]
capped ← (n % k = 0) ? j : Min(j, n)
index ← 0
visited_since_last_kill ← k
killed ← 0
last_killed ← 0
while killed < capped do
    i ← index % n
    person ← people[i]
    if person = 0 then
        index ← index + 1
    else if visited_since_last_kill = k then
        last_killed ← person
        people[i] ← 0
        killed ← killed + 1
        visited_since_last_kill ← 0
    else
        visited_since_last_kill ← visited_since_last_kill + 1
    index ← index + 1
return last_killed
```