Problem 1

1.1

Consider the following function of n:

$$n^2 \quad \sum_{i=0}^{n} 5 \cdot i \quad n^3 \cdot \sqrt{\frac{1}{n^3}} \quad n^2 + 2^n \quad (\Pi_{i=1}^{9} i) \quad (\sum_{i=0}^{\log_2(n)} 2^i) + 1 \quad 7^{\ln(n)}$$

$$-\ln(\frac{1}{n}) \quad \ln(2^n) \quad 10 \quad n \log_2(n^7) \quad \sqrt{n^4} \quad n^n \quad 5n$$

Group the above functions that have identical growth rate and order these groups on increasing growth. Hence:

- If you place functions $f_1(n)$ and $f_2(n)$ in the same group, then we must have $f_1(n) = \Theta(f_2(n))$;
- If you place function $f_1(n)$ in a group ordered before the group in which you place function $f_2(n)$, then we must have $f_1(n) = \mathcal{O}(f_2(n)) \wedge f_1(n) \neq \Omega(f_2(n))$.

Solution:

Theorem 3.25 states the following:

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} \text{ is defined and is } \begin{cases} \infty & \text{then } f(n) = \Omega(g(n)); \\ c, \text{ with c} > 0 \text{ a constant} & \text{then } f(n) = \Theta(g(n)); \\ 0 & \text{then } f(n) = \mathcal{O}(g(n)); \end{cases}$$

The following are the rules of thumb to determine the order of growth of functions:

$$
\begin{aligned}
c \cdot f(n) &= \Theta(f(n)) \\
\log_a(n) &= \Theta(\log_b(n)) \\
\lim_{n\to\infty} \frac{n^c}{n^{c+d}} &= \lim_{n\to\infty} \frac{1}{n^d} = 0 \to n^c = \mathcal{O}(n^{c+d}) \\
\log_2(n)^c &= \mathcal{O}(n^d)\ \forall c > 0, d > 0 \\
n^c &= \mathcal{O}(d^n)\ \forall c > 0, d > 1 \\
d^{\frac{n}{u}} &= \mathcal{O}(c^{\frac{n}{v}})\ \forall c \geq d \geq 1, u \geq v \geq 1 \\
\sum_{i=1}^{m} c_i \cdot n^{d_i} &= \mathcal{O}(n^{d_i}) \\
f(n) + g(n) &= \Theta(g(n)) \\
h(n) \cdot n(n) &= \mathcal{O}(h(n) \cdot g(n))
\end{aligned}
$$

Let's start with analysing the functions that have the same growth rate:

- n^2: polynomial growth, $n^2 = \Theta(n^2)$ based on rule 3.
- $\sum_{i=0}^{n} 5 \cdot i$: polynomial growth, $\sum_{i=0}^{n} 5 \cdot i = 5 \cdot \frac{n(n+1)}{2} = \frac{5}{2}n^2 + \frac{5}{2}n$, which is $\Theta(n^2)$ based on rule 7.
- $n^3 \cdot \sqrt{\frac{1}{n^3}}$: polynomial growth, $= n^3 \cdot \frac{1}{n^{\frac{3}{2}}} \to \Theta(n^{\frac{3}{2}})$ based on rule $9, 3$
- $n^2 + 2^n$: exponential growth, $= \Theta(2^n)$ based on rule 7
- $(\Pi_{i=1}^{9} i)$: constant, $= 1 \cdot 2 \ldots 9$ based on rule 9
- $\sum_{i=0}^{\log_2(n)} 2^i + 1$: linear, since the term is a geometrics series, such that $\sum_{i=0}^{\log_2(n)} 2^i + 1 = \frac{1 \cdot (2^{\log_2(n)+1} - 1)}{2-1} + 1 = 2 \cdot 2^{\log_2(n)} = 2n$. Therefore based on rule 1 we have $\Theta(n)$
- $7^{\ln(n)}$: polynomial, since $7^{\ln(n)} = n^{\ln(7)}$. Apply rule 3 we have $7^{\ln(n)} = \Theta(n^{\ln 7}) \to \mathcal{O}(n^2)$
- $-\ln(\frac{1}{n})$: logarithmic, since $-\ln(\frac{1}{n}) = \ln(n)$. Apply rule 1 yield $\Theta(\ln n)$.
- $\ln(2^n)$: linear, since $\ln(2^n) = n \cdot \ln_2$ and rule 1 yield $\Theta(n)$
- 10: constant
- $n \log_2(n^7)$: polynomial, since $n \log_2(n^7) = n \cdot 7 \log_2(n) = 7n \log_2(n)$. Based on rule 9 and 1 yield $\mathcal{O}(n^2)$
- $\sqrt{n^4}$: polynomial growth, $= n^2$ based on rule 3.
- n^n: super-exponential.
- $5n$: linear based on rule 1, gives $\Theta(n)$

Thus the order from increasing rate:

- $10 \quad (\Pi_{i=1}^{9} i)$ (*constant*)
- $-\ln(\frac{1}{n})$ (*logarithmic*)
- $\ln(2^n) \quad 5n \quad \sum_{i=0}^{\log_2(n)} 2^i + 1$ (*linear*)
- $n^2 \quad \sum_{i=0}^{n} 5 \cdot i \quad n^3 \cdot \sqrt{\frac{1}{n^3}} \quad \sqrt{n^4} \quad n\log_2(n^7) \quad 7^{\ln(n)}$ (*polynomial*)
- $n^2 + 2^n$ (*exponential*)
- n^n (*super-exponential*)

1.2

Consider the following recurrence

$$T(n) = \begin{cases} 7 & \text{if } n \leq 1; \\ 3T(n-2) & \text{if } n > 1. \end{cases}$$

Use induction to prove that $T(n) = f(n)$ with $f(n) = 7 \cdot 3^{[\frac{n}{2}]}$

Solution:

base case:

Given $T(n) = f(n) = 7 \cdot 3^{[\frac{n}{2}]}$ apply for $n = 0$ and $n = 1$:

- $n = 1 \rightarrow T(n) = 7 \cdot 3^{[\frac{1}{2}]} = 7 \cdot 3^0 = 7$
- $n = 0 \rightarrow T(n) = 7 \cdot 3^{[\frac{0}{2}]} = 7 \cdot 3^0 = 7$

Thus the base case holds.

induction hypothesis:

Assume $T(n) = f(n)$ holds for $n > 1$. We will proves that $f(n+2) = T(n+2)$ also holds.

$$f(n) = T(n)$$
$$T(n) = f(n) = 3T(n-2) = 3f(n-2) = 3 \cdot 7 \cdot 3^{[\frac{n-2}{2}]}$$
$$T(n+2) = 3T(n)$$

Therefore:

$$T(n+2) = 3 \cdot 3 \cdot 7 \cdot 3^{[\frac{n-2}{2}]} = 7 \cdot 3^{[\frac{n-2}{2}+2]}$$
$$T(n+2) = 7 \cdot 3^{[\frac{n+2}{2}]} = f(n+2)$$

Therefore the induction hypothesis holds.

conclusion:

> $T(n) = f(n) = 7 \cdot 3^{[\frac{n}{2}]}$ is true for all $n \geq 0$.

Problem 2

Consider the following `Count` algorithm

```
Algorithm Count(L, v):
Pre: L is an array, v is a value
i, c := 0, 0
while i neq |L| do
  if L[i] = v then
    c := c + 1
 end if
 i := i + 1
end while
return c.
Post: return the number of copies of v in L
```

2.1

Provide an invariant for the while loop at Line 2

Solution:

$$0 \leq i \leq |L|, c = \sum_{j=0}^{i-1}[L[j] = v]$$

2.2

Provide a bound function for the while loop at Line 2

Solution:

$$f(i) = |L| - i$$

2.3

Prove that `Count` algorithm is correct.

Solution:

Line 1: `i, c := 0, 0`

- $L[0, i)$ with $i = 0$ is $L[0, 0)$
- $L[0, 0)$ is empty, hence $c = \sum_{j=0}^{i-1}[L[j] = v] = 0$
- bound function $f(i) = |L| - i$ starts at $|L|, |L| \geq 0$

Line 2: `while i neq |L| do`

- bound function $f(i)$ stops at 0
- invariant still holds, with $i \neq |L|$

Now prove invariant holds again til reach the end of the `m-th` loop:

Line 3-5: `if L[i] = v then` case:

- $L[i] = v$ hence $v \in L[0, i]$

- invariant still holds, with $i \neq |L|$ and $L[v] = v$
- $i_{\text{new}} = i + 1$ hence $0 < i_{\text{new}} \leq |L|$ implies $ 0 \leq i{\text{new}} \leq |L|$ and $f(i{\text{new}}) = f(i) - 1$
- $c_{\text{new}} = c + 1 = \sum_{j=0}^{i-1} [L[j] = v] + 1 = \sum_{j=0}^{i_{\text{new}}} [L[j] = v]$
- $f(i)$ strictly decreases after each iteration, $i_{\text{new}} := i + 1$

Therefore the invariant still holds within the if statement.

L7: `end while`

- $i = |L|$ hence $f(i) = 0$, the loop stops
- $c = \sum_{j=0}^{i-1} [L[j] = v] = \sum_{j=0}^{|L|-1} [L[j] = v]$

Therefore the invariant still holds at the end of the loop.

2.4

What is the runtime and memory complexity of `Count` algorithm?

Solution:

- L1 implies 2 instructions
- L2 implies 2 instructions $|L| + 1$ times,
- L3-5 (if loop) implies 4 instructions $|L|$ times
- L6 implies 2 instructions $|L|$ times

Therefore number of work is $5 + 8N$, thus runtime complexity would be $\Theta(5 + 8N)$.

Memory complexity is $\Theta(1)$, since only 2 variables are used.

2.5

Provide an algorithm `FastCount(L, v)` operates on ordered lists `L` and computes the same results as `Count(L, v)` but with a $\mathcal{O}(\log_2(|L|))$

Solution:

```
Algorithm FastCount(L, v):
Pre: L is an ordered array, v is a value
function binarySearchFirst(L, v)
  low, high := 0, |L| - 1
  results := -1
  while low ⩽ high do
    mid := (low + high) / 2
    if L[mid] < v then
      low := mid + 1
    else if L[mid] > v then
      high := mid - 1
    else
      results := mid
      high := mid - 1
  end while
  return results
end function

function binarySearchLast(L, v)
  low, high := 0, |L| - 1
  results := -1
  while low ⩽ high do
    mid := (low + high) / 2
    if L[mid] < v then
      low := mid + 1
    else if L[mid] > v then
      high := mid - 1
    else
      results := mid
      low := mid + 1
  end while
  return result
end function
```

```
firstIndex := binarySearchFirst(L, v)
if firstIndex = -1 then
  return 0
end if
lastIndex := binarySearchLast(L, v)
return lastIndex - firstIndex + 1
Post: return the number of copies of v in L
```