0.1.1. 7.1 Answer each part TRUE or FALSE

0.1.1.1. a. $2n = O(n)$
by the definition of $O(n)$ we have

$\boldsymbol{L \in O(n) \Leftrightarrow cL \leq n}$

set c to 2 , then we have $\boldsymbol{2n}$, so it must be the case that $2n \in O(n)$

0.1.1.2. b. $n^2 = O(n)$
by the definition, again

$L \in O(n) \Leftrightarrow cL \leq n$

but note that c must be n, thus

$nL \leq n^2$

thus it must be that this case is false

0.1.1.3. c. $n^2 = O(n \log^2 n)$
by the definition of $O(n \log^2 n)$

remember that we can multiply the outside factor for the inside one, thus

$L \in O(\log^2 n^2) \Leftrightarrow L \leq c \log^2 n^2$

but, then, $2^L \leq 2^c n^2$, but it's not the same language anymore, thus it must be the case that n^2 does not belong to $O(\log^2 n^2)$, false

0.1.1.4. d. $n \log n = O(n^2)$
by the definition, $n \log n \in O(n^2) \Leftrightarrow n \log n \leq cn^2$, simplifying

$\log n^2 \leq cn^2$

$n^2 \leq 2^c 2^{n^2}$ again not a constant, as such is false.

0.1.1.5. e . $3^n = 2^{O(n)}$
$3^n \in 2^{O(n)} \Leftrightarrow 3^n \leq 2^{cn}$

0.1.1.6. f . $2^{2^n} = O(2^{2^n})$
again, by definition

$2^{2^n} \in O(2^{2^n}) \Leftrightarrow 2^{2^n} \leq c2^{2^n}$

keep applying log

$2^n \leq \log c + 2^n$

repeat, one more time

$2^n - 2^n \leq \log c$

$0 \leq \log c$

so there's a constant ≥ 1 so TRUE.

0.1.2. 7.2 Answer each part TRUE or FALSE

0.1.2.1. a. $n = o(2n)$
based on the definition of $o(n)$

$L \in o(n) \Leftrightarrow n < cn$

if $c = 1$ then it must be the case that $n \in O(n)$, but then we can always find a constant that would make this false, they have the same growth rate, thus it must be false.

0.1.2.2. b . $2n = o(n^2)$
it must be the case that $n \in o(n^2) \Leftrightarrow n < cn^2$ thus it must also be the case that $2n < n^2$ for any c, thus $2n \in o(n^2)$

0.1.2.3. c. $2^n = o(3^n)$
again, based on the definition

$2^n \in o(3^n) \Leftrightarrow 2^n < c3^n$

it must be the case as well, thus TRUE.

0.1.2.4. d. $1 = o(n)$
by the definition $1 \in O(n)$,that is a linear function always grow faster than a constant, TRUE.

0.1.2.5. e n = o(log n)
by the definition

$\boldsymbol{n \in o(\log n) \Leftrightarrow n < c \log n}$

but $\boldsymbol{\log n \leq n}$, thus it must be false.

0.1.2.6. f. $\boldsymbol{1 = o(\frac{1}{n})}$
this one must be false, due to $\boldsymbol{0 < n < 1}$ being the only case for this to be true.

0.1.2.7. 7.4 Fill out the table described in the polynomial time algorithm for context free language
recognition from Theorem 7.16 for string w = baba and CFG G:

$S \to RT$
$R \to TR \mid a$
$T \to TR \mid b$

	1	**2**	**3**	**4**	**4**
1	T				
2		R			
3			T		
4				R	

7.6 Show that P is closed under union, concatenation, and complement

Consider the following TM

```
$M =$ "On input $\langle M_1, M_2, a\rangle$ where $M_1$is a TM and $M_2$ is a TM and a is a string
    1. Run $M_1$ on input a
    2. Run $M_2$ on input b
    3. If any of them accept, accept, else reject
```

Due to each machine taking at most polynomial time due to $L \in P$ it must be the case that step 1 takes $O\left(t^{k_i}\right)$ and step 2 $O\left(t^{k_j}\right)$ for some $\left(k_j, k_i\right) \in N \times N$ and the sum of two polynomials is a polynomial, thus it must be the case that the union of two P languages is also in P.

```
$M =$ "On input $\langle M_1, M_2, a\rangle$ where $M_1$ is a TM and $M_2$ and is a TM, a is a string
    1. For $k = 0$ to $|a|$
    2. Partition a as $(l, r)$ where $l$ start at position 0 of a and end at $k - 1$ and r start at $k$ and end at |a| position
    3. Run $M_1$ in $l$ and $M_2$ in $r$
    4. If $M_1, M_2$ accepts then accept
    5. Reject
```

Note that we can partition a in a^2 splits, each partition can have at most size $|a|$ and two exists, each partition takes $O\left(t^{k_j}\right)$ for M_1 and $O\left(t^{k_i}\right)$ for M_2 for $(t_k, t_i) \in N \times N$, so as seen before the sum of polynomials is a polynomial, and the multiplication of a constant by a polynomial is a polynomail, a^2, in this case.

As such it must be the case that concatenation of languages is in P

```
$M =$ "On input $\langle M_1, a\rangle$ where $M_1$ is a TM and a a string
    1. Run $M_1$ on a
    2. Reject if accepts and accept if rejects
```

Note that it must be the case that the time it takes is $O\left(t^k\right)$ where $k \in N$, as such it take polynomial time, consequently the extra step is constant time, inverting the answer, thus negation of a language in P belongs to P as well

0.1.3. 7.7 Show that NP is closed under union and concatenation

```
$M =$ "On input $\langle M_1, M_2, a\rangle$ where $M_1$ is a TM, $M_2$ is a TM and a is a string
    1. Run $M_1$ with a as input
    2. Run $M_2$ with a as input
    3. Accept if one of them accept, otherwise reject"
```

Note that takes polynomial time for each machine, and the sum of polynomials is a poylnomial consequently it must be the case that the union of two languages in NP is in NP

```
$M =$ "On input $\langle M_1, M_2, a\rangle$ where $M_1, M_2, a$ is a TM, TM and string respectively
    1. Partition a in a non deterministic manner
    2. For each partition run $M_1$ on the left substring and $M_2$ on the right substring
    3. Accept if both accept
    4. Reject
```

0.1.4. 7.8 Let CONNECTED = $\{< G > \mid$ G is a connected undirected graph$\}$. Analyze the algorithm given on page 185 to show that this language is in P.

```
$M =$ "On input $\langle G \rangle$ where G is the encoding of a Graph
  1. Select the first node of G and mark it
  2. Repeat the following stage until no new nodes are marked:
  3.  For each node in G, mark it if its attached by an edge to a node that is already marked
  4. Scan all the nodes of G to determine
     whether they all are marked. If they are, accept; otherwise, reject"
```

- the first step takes $O(1)$ so constant time
- the second step takes $O(G^3)$ so linear time
- the last step takes $O(G)$ so linear time

so $O(G^2)$ is the final time the second step has the following time because in the worst case a node might compare with $n - 1$ nodes and n nodes exist, so $O(n^3)$

0.1.5. 7.9 A triangle in an undirected graph is a 3-clique. Show that TRIANGLE ∈ P, where TRIANGLE = $\{\langle G \rangle \mid G \text{ contains a triangle}\}$

```
$M =$ "On input $\langle G \rangle$ where $G$ is a graph
  1. For i = 1 to $\binom{G}{3}$ where the latter is the size of all combinations of size 3 in G of nodes, do:
  2. Test whether a combination that has not being seen is a 3-clique
   2.1. If  it is, then accept
  4. reject"
```

Note that step 1 takes $\frac{G!}{(G-3)!3!}$ and that number is $\frac{G(G-1)(G-2)}{6}$ that give us $\boldsymbol{O(n^3)}$ time complexity for this operation, the step two takes $\boldsymbol{O(1)}$ time complexity, cause a check can be done in constant time, step 4 is $\boldsymbol{O(1)}$ as well, thus it takes polynomial time to find a 3-clique. ☐

0.1.6. 7.10 Show that $\mathbf{ALL_{DFA}}$ is in P.

```
$M =$ "On input $\langle A \rangle$ where A is a DFA
  1. Mark the starting state
  2. Repeat until no achievable state can be marked, meaning available by starting at the start state
   2.1. For each marked state, mark every state that has not been marked
    and is accesible by the application of a single transition function that has the current state as state argument
   2.2 If a non accepting state has been marked, reject

  3. Accept"
```

Step 1 takes $\boldsymbol{O(1)}$ time, step 2 takes, in the worst case, $\boldsymbol{O(|\text{states}| + |\delta_{\text{achievable}}|)}$ thus a polynomial ,the other steps are $\boldsymbol{O(1)}$, thus the total runtime is

$$O(|\text{states}| + |\delta_{\text{achiveable}}|) \cdot (O(1) + O(1)) + O(1)$$

thus a polynomial time complexity algorithm

0.1.7. 7.11 In both parts, provide an anylisis of the time complexity of your algorithm.

0.1.7.1. a. Show that $\mathbf{EQ_{DFA}} \in \boldsymbol{P}$

```
$M =$ On input $\langle D_1, D_2 \rangle$ where both are dfa's
   1. Execute these for each dfa $D_1, D_2$
    1.1. Invert every state of $D_i$ and create a new DFA encoded with the modifications, call it $D_3$
    1.2. Create a new dfa $D_4$ that is the union of $D_3, D_j$
    1.3. Run $\mathrm{ALL}_{\mathrm{DFA}}$ with $D_4$ as input,
   4. If in both runs the TM $\mathrm{ALL}_{\mathrm{DFA}}$ accepts then accept, otherwise reject"
```

Note that DFA's are closed for complement and union, as seen in previous chapters. If indeed D_2 is equal to D_1 it must be the case that the complement of $D_2 \cup D_1$ is Σ^*, but it could also be that $D_1 \vee D_2$ are Σ^* , thus we must deal with that case, and indeed if one of them is Σ^* the complement will be the empty language, thus not satisfying Σ^* after their union if L_2 is not Σ^*, but in the case that both are them the construction would always return true from $\mathrm{ALL}_{\mathrm{DFA}}$.

Now the time complexity analysis, step 0 takes $O(1)$, setp 1 takes $O(|\text{States of dfa}_i|)$, step 2 takes $O\big(|\text{states of dfa}_3| \times |\text{state of dfa}_j\ |\big)$ and step 3 takes same time as previous step state 4 takes $O(1)$ time.

Then the total runtime is

$$O(1) \cdot \big(O(\text{States of dfa}_i) + 2O\big(|\text{states of dfa}_3| \times |\text{state of dfa}_j\ |\big)\big) + O(1)$$

thus a polynomial and as such it take polynomial time to decide the language.

0.1.7.2. b. Say that a language A is star closed if $A = A^*$. Give a polynomial time algorithm to test whether a DFA recognizes a star-closed language (Note that $\mathrm{EQ}_{\mathrm{NFA}}$ is not known to be in P)

```
$M =$ "On input $\langle D, a \rangle$ where D is a DFA and a is a string
   1. For k equal to the increasing sequence of numbers that $\bmod a = 0$
    1.1. Split a in $\frac{a}{k}$ sequences, each of size k
    1.2 For j = 0 to $j = \frac{a}{k}$
     1.2.1 Concatenate the first j sequences and run in D
     1.2.2 If it rejects, then go back to step 1 and try the next possible value of k
    1.3 Accept
   2. Reject"
```

Step 1 takes $\boldsymbol{O(1)}$ time , step 1.1 takes $\boldsymbol{O\left(\frac{a}{k}\right)}$ and step 1.2.1 takes $\boldsymbol{jO(k)}$ time and step 1.2.2 takes $\boldsymbol{O(1)}$ time and last steps take $\boldsymbol{O(1)}$ time as well.

it can be seem that we have multiplication of polynomials by constant factor, thus it must be the case that is decidable in polynomial time. ⬚

0.1.8. 7.12 Call graphs G and H isomorphic if the nodes of G may be reordered so that it is identical

to H. Let $\mathbf{ISO} = \{\langle G, H \rangle \mid G \text{ and } H \text{ are isomorphic graphs}\}$. Show that $\mathbf{ISO} \in \mathbf{NP}$

```
$M =$ "On input $\langle G, H \rangle$ where G,H are graphs
    1. Non deterministically select a permutation of G, call it p
    2. Exchange H nodes with the permutation nodes
    3. Check if edges are the same in both
    4. Accept if they are, reject otherwise
```

step 1 takes $O(1)$ time, for each branch, step 2 takes $O(|G_{\text{nodes}}|)$, step 3 takes $\boldsymbol{O}(|\boldsymbol{G}_{\mathbf{nodes}}| \cdot |\boldsymbol{H}_{\mathbf{nodes}}|)$ and last step takes $\boldsymbol{O}(\mathbf{1})$

thus final time complexity is polynomial and decided by a NDTM, thus ISO is an NP problem.

0.1.9. 7.13 Let $\mathbf{MODEXP} = \{\langle \boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}, \boldsymbol{p} \rangle \mid \boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}, \textbf{and } \boldsymbol{p} \textbf{ are positive binary integers such that } \boldsymbol{a}^{\boldsymbol{b}} \equiv \boldsymbol{c} (\mathbf{mod}\, \boldsymbol{p})\}$.

Show that MODEXP $\in P$. (Note that the most obvious algorithm doesn't run in poly time. Hint: try it first where b is power of 2.)

```
$M =$ On input $\langle a, b, c, p \rangle$ where a,b,c,p each is a positive binary number
    1. Compute $a \bmod p$, store it as x
    2. For i=0 to $|b|$ in binary
     2.1. If $b_i$ is 1
      2.1.2. update value of y as $yx \bmod p$
     2.2. update x as $x \equiv x^2 \bmod p$
    3. if $x = c$ accept, otherwise reject"
```

Step 1 takes $\boldsymbol{O}(\mathbf{1})$, step 2 takes $\boldsymbol{O}(|\boldsymbol{b}|)$, step 2.1 takes $\boldsymbol{O}(\mathbf{1})$, step 2.1.2 takes $\boldsymbol{O}(\mathbf{1})$ and last step takes $\boldsymbol{O}(\mathbf{1})$ thus is a polynomial time algorithm.

0.1.10. 7.14 Show that if $P = \text{NP}$, then every language $A \in P$, except $A = \text{empty}$ and $A = \Sigma^*$, is NP-complete.

1. Suppose that $P = \text{NP}$.
2. We now prove that $\mathbf{SAT} \underset{\boldsymbol{p}}{\leq} \mathbf{empty}$ is a false statement and due to that $\mathbf{empty}$ can't be NP-complete
3. Note that $\mathbf{empty}$ has 0 cardinality there's no mapping possible that allows us to partition the accepting strings and rejected strings, as such this reduction is impossible
4. We now prove that $\mathbf{SAT}^* \underset{\boldsymbol{p}}{\leq} \mathbf{\Sigma}^*$ is a false statement