EXAM 4 cs334 automata

1. **∅ ∈ S for every set S.**	False
2. **∅ ∩ S = S for every set S.**	False
3. **∅ ∪ S = S for every set S.**	True
4. **∅ ⊆ S for every set S.**	True
5. **3-Sat is reducible to**	Independent-set
6. **{a^i b^j c^k d^l e^m f^n} is recursive**	True
7. **acceptance problem**	the problem of testing whether a Turing machine accepts a given string
8. **accepting configuration**	a Turing machine configuration where the current state is qaccept
9. **add an element to the end of an array (average case)**	O(1)
10. **add to the beginning of a link based list**	O(1)
11. **Add to the beginning of array based list**	O(n)
12. **add to the end of a Link based list**	O(1)
13. **Add to the middle of a link based list**	O(n)
14. **Algorithm**	Collection of simple instructions for carrying out some task.
15. **All recursive languages are recursively enumerable**	True
16. **All Turing Machines eventually halt on input (lambda)**	False
17. **All Turing Machines have some input on which they run forever**	False
18. **Alphabet**	A finite set of objects called the symbols of the alpabet
19. **Alphabet**	Any nonempty, finite set
20. **alphabet**	any finite set (for our purposes)
21. **Ambiguity**	A grammar is ambiguous iff there is at least one string in L(G) for which G produces more than one parse tree.
22. **ambiguous grammar**	a grammar that derives some string ambiguously
23. **ambiguous grammars**	if a word has two derivation trees or more
24. **ambiguously**	A string w is derived _____ in a context-free grammar G if it has two or more leftmost derivations.
25. **Are any of the group problems impossible?**	P and NP cannot be disjoint. Any NP problem can be solved to be a P problem ??
26. **arity of f**	k, where f is a function with k arguments
27. **Array (average)**	Access: O(1) Search: O(n) Insertion: O(n) Deletion: O(n)
28. **Array (worst)**	Access: O(1) Search: O(n) Insertion: O(n) Deletion: O(n)
29. **Automata theory**	study of abstract machines and problems they are able to solve
30. **A - B**	Difference - Elements in A but not B
31. **Backus Naur Form: a notation for writing practical context-free grammars**	True
32. **BFS and DFS**	O(|V|+|E|)

33. **Binary Function**	When K = 2
34. **binary search**	logn
35. **Binary Search Tree (average)**	Access: O(log n) Search: O(log n) Insertion: O(log n) Deletion: O(log n)
36. **Binary search tree delete average case**	O(logn)
37. **Binary search tree delete best case**	O(1)
38. **Binary search tree delete worst case**	O(n)
39. **binary search tree find average case**	O(logn)
40. **binary search tree find best case**	O(1)
41. **Binary Search tree find min/max average**	O(logn)
42. **Binary Search tree find min/max best case**	O(1)
43. **Binary search tree find min/max worst case**	O(n)
44. **binary search tree find worst case**	O(n)
45. **Binary search tree insert average case**	O(logn)
46. **Binary search tree insert best case**	O(1)
47. **Binary search tree insert worst case**	O(n)
48. **Binary Search Tree (worst)**	Access: O(n) Search: O(n) Insertion: O(n) Deletion: O(n)
49. **BINSEARCH contained in NP**	yes, a better bound is O(nlogn)
50. **BNF (Backus Naur form)**	meta syntax used to express CFGS, grammar -> syntax
51. **bubble sort**	n^2
52. **CFG Ambiguity**	A string with 2 or more left-most derivations
53. **Check if a binary search tree is empty**	O(1)
54. **Chomsky Normal Form**	the Productions are in the following forms - A → a A → BC S → ε where A, B, and C are non-terminals and a is terminal.
55. **Chomsky Normal Form**	A -> BC A -> a exists for any CFG
56. **Chomsky normal form**	If every rule of a context-free grammar is of the form: A → BC A → a where A is any variable, B and C are any non-start variables, and a is any terminal, the grammar is in _____. (S → ε, where S is the start variable, is also allowed)
57. **The Church-Turing Thesis proves that HALTING is decidable**	True
58. **CLIQUE**	NP-complete
59. **complementation proof**	flip final states with non-finals
60. **Complexity**	What makes some problems computationally hard and others easy? (easy vs hard)
61. **Computatbility**	What are the fundamental capabilities and limitations of computers? (Solvable and Unsolvable)

62. **computation**	any sequence that leads to a halt state
63. **computation history**	the sequence of configurations a Turing machine goes through on a given input
64. **configuration**	for a Turing machine, uqv where: uv is the current tape contents (with only blanks after v), q is the current state, and the first symbol of v is the current head location
65. **connected graph**	a graph such that every two nodes have a path between them
66. **Consider for a moment.....**	● You've just discovered a new computational problem ○ But you cannot find a polynomial time solution ■ If you can show that the problem is NP-Complete, you know that finding a polynomial time solution boils down to solving P vs NP
67. **Context-Free Grammar**	no restriction grammar essentially
68. **Context Free Grammar**	A 4-tuple with finite set of grammar rules. Includes a set of non-terminal symbols, a set of terminal symbols, a set of rules, and the start symbol
69. **context-free grammar**	a 4-tuple (V, Σ, R, S), where: 1. V is a finite set called the variables, 2. Σ is a finite set, disjoint from V, called the terminals, 3. R is a finite set called the rules, with each rule being a variable and a string of variables and terminals, and 4. S ∈ V is the start variable.
70. **Context Free Language Closure Properties**	Union, Concatenation, Kleene star Reverse , Letter Substitution
71. **Context Free Languages can be recognized by**	Pushdown Automata
72. **convert fa to tm**	start with fa change each label x to (x, x, r) add accept state
73. **Cook Levin Theorem**	SAT is NP-complete so if SAT has a polynomial algorithm then all of those in NP-complete are also polynomial
74. **Co-Turing Recognizable**	Complement of a Turing Recognizable Language
75. **co-Turing-recognizable language**	a langugae that is the complement of a Turing-recognizable language
76. **cycle**	a path that begins and ends on the same node
77. **Decidable/Recursive Language**	A language is decidable if some TM decides it. AKA Recursive Language
78. **decider**	a Turing machine that halts on all inputs
79. **Decision problem is**	simply a problem for which answer is yes or no
80. **Decision procedure**	answers a decision problem

81. **Define a mapping reduction of the acceptance problem Atm to the halting problem HALT TM**

For every TM M, let M* be the following TM:
M* = "On input x:
1. Run M on x.
2. If M accepts, accept
3. If M rejects, enter an infinite loop."
Thus:
If M accepts input x, then M* accepts x
If M explicitly rejects x, then M* never halts on x
If M never halts on x, then M*never halts on x
To summarize, M accepts x iff M* halts on x

Let then f be the function defined by f(<M,w>)=<M*,w>.

Is f computable? Yes

Obviously <M,w> is in ATM iff f(<M,w>) is in HALT TM i.e. f is a mapping reduction of Atm to HALT TM
So, since ATM is undecidable, HALT TM is undecidable as well.

82. **Define Mapping Reduction and Mapping Reducibility**

Let A and B be languages over an alphabet Sigma.
We say that A is mapping reducible to B, written A<=mB, if there is a computable
function f: Sigma**->Sigma** such that, for every w is in Sigma*,
w is in A iff f(w) is in B.
The function f is called a mapping reduction of A to B.

83. **Definition of Accept**

A TM accepts an input string iff, for this input, sooner or later it enters the accept state.
Otherwise the string is considered rejected.

84. **Definition of acceptance for a Finite State Automata M**

M accepts the string u1 u2 ... un
iff there is a sequence r1, r2, ..., rn, rn+1
of states such that:
r1=s
ri+1 = delta(ri,ui), for each i with 1<= i <= n
rn+1 is in F

85. **Definition of a CFG**

A 4-tuple (V,Sigma,R,S)
1. V is a finite set called the variables;
2. Sigma is a finite set, disjoint from V, called the terminals;
3. R is a finite set of rules, with each rule being a pair of a variable and a string of variables and terminals;
4. S is an element of V called the start variable.

86. **Definition of a regular expression**

R is a Regular Expression (RE) iff R is one of the following:
1. a, where a is a symbol of the alphabet

2. The Empty Set

3. The Null Set

4. (R1)U(R2), where R1 and R2 are RE

5. (R1) o (R2), where R1 and R2 are RE

6. (R1)*, where R1 is a RE

87.	**Definition of Asymptotic Upper Bound**	Let f and g be functions f,g: N->R+. Say that f(n) = O(g(n)) iff positive integers c and n0 exists such that for every integer n>=n0, f(n) Less than or equal to cg(n). When f(n) = O(g(n)), we say that g(n) is an asymptotic upper bound for f(n).
88.	**Definition of Cartesian Product for sets S and T**	S X T={(s,t) | s is in S and t is in T}
89.	**Definition of Concatenation**	L1 o L2 = {xy | x is in L1 and y is in L2}
90.	**Definition of Decidable Relative**	Language A is decidable relative to language B iff there is an OTM with an oracle for B that decides A.
91.	**Definition of Decider**	A Turing machine is said to be a decider iff it halts for every input.
92.	**Definition of Halting Problem**	Let HALT TM = {<M,w> | M is a TM and M halts on input w}
93.	**Definition of Oracle Turing Machine**	An oracle Turing machine (OTM) is a modified Turing machine that has the additional capability of querying an oracle. An oracle for a language B is an external device that is capable of reporting whether any string w is a member of B.
94.	**Definition of Star**	L* = {x1...xk | k >= 0 and each xi is in L}
95.	**Definition of the class of languages P**	P is the class of languages that are decidable in polynomial time on a deterministic (single-tape) TM. In other words, P = TIME(n^1) U TIME(n^2) U TIME(n^3) U TIME(n^4) U ...
96.	**Definition of the class P**	the class of languages that are decidable in polynomial time on a deterministic single-tape TM. the union over k of TIME(n^k), problems that are reasonably solvable
97.	**Definition of time complexity for nondeterministic machines**	Let M be a nondeterministic TM that is a decider (meaning that, on every input, each branch of computation halts). The running time or time complexity of M is the function f: N->N, where f(n) is the maximum number of steps that M uses on any branch of its computation on any input of length n.
98.	**Definition of Turing Reducibility**	Language A is Turing reducible to language B, written A<=TB, iff A is decidable relative to B.
99.	**Definition of Union**	L1 U L2 = {x | x is in L1 or x is in L2}
100.	**Definitions of computing, computability, and the graph of a function**	A function g: Sigma**-> Sigma** is said to be computable, iff there is a TMO M such that for every input w is in Sigma* M returns the output u with u=g(w). In this case we say that M computes g. The graph of such a function is the language {(w,u) | w is in Sigma*, u=g(w)}
101.	**Definitions of NP, coNP, EXPTIME**	NP is the class of languages decided by some nondeterministic polynomial time Turing machine. coNP = {L | L is the complement of some language in NP} EXPTIME = TIME(2^(n^(1))) U TIME(2^(n^(2))) U TIME(2^(n^(3))) U ...
102.	**Definitions of Running Time and Time Complexity**	Let M be a deterministic TM that halts for every input. The running time or time complexity of M is the function f: N->N, where f(n) is the maximum number of steps that M uses on any input of length n. If f(n) is the time complexity of M, we say that M runs in time f(n), or that M is an f(n) time machine.
103.	**degree of A**	the number of edges at a node A
104.	**delete all n elements from an array by repeatedly removing the first element (at index 0)**	n^2
105.	**delete an item from a binary search tree (average case)**	O(logn)

106. **Delete the first item from a singly-linked list (assume a head reference but no tail)**	O(1)
107. **Delete the last item from a singly linked list (assume a head reference but no tail)**	O(n)
108. **Delete the last (tail) node from a doubly linked list with a dummy node implementation**	O(1)
109. **dequeue from a circular array-based queue (worst case)**	1
110. **Dequeue the maximum item from a maximum binary heap**	logn
111. **Deterministic**	○ At any point during the run of the program, given the current instruction and input, we can predict the next instruction ○ Running the same program on the same input produces the same sequence of executed instructions
112. **Deterministic Finite Automata (DFA)**	A 5-tuple represented by the set of states, the input alphabet, transition function, the start state and the set of accepting states
113. **DFA**	M = (Q, Sigma, Delta, q_0, F) -accepter, accepts a certain language -can be represented by graph, table, or functions -no lambda transitions -every letter in alphabet must have an outgoing arrow from each state, ie if {a, b} then each state must have exactly one a arrow coming out and exactly one b arrow coming out -right hand side of delta can only be one state per rule not a set
114. **DFA & NFA**	5-tuple(Q, ∑, δ, q0, F) Q - states ∑ - alphabet δ - Q x ∑ -> Q is the transition function q0 - start state F - set of accept states
115. **DFA & NFA Equivalence**	Every DFA is an NFA. Every NFA has an equivalent DFA by Construction
116. **DFSM = NDFSM**	Can be proven in Formal Theory
117. **Diagonalilzation**	1.shows the correspondence b/w 2 finite sets represented as functions 2.. Helps determine how to find the relative sizes of finite sets 3. Shows count-ability
118. **Dictionary get, set, delete, contains**	O(1)
119. **directed graph**	a graph whose edges are arrows instead of lines
120. **directed path**	a path such that all arrows point in the same direction as its steps
121. **Does TM M accept epsilon?**	SD/D
122. **Does TM M accept w?**	SD/D
123. **Does TM M halt on all strings?**	in NOT(SD)
124. **Does TM M halt on the empty tape**	SD/D

125. **Does TM M halt on w?**	SD/D
126. **Does TM M have an even number of states?**	in D
127. **Doubly-Linked List (average)**	Access: O(n) Search: O(n) Insertion: O(1) Deletion: O(1)
128. **Doubly-Linked List (worst)**	Access: O(n) Search: O(n) Insertion: O(1) Deletion: O(1)
129. **DPDA**	AN NPDA that adheres to following: -(terminal symbol, top of stack symbol) combination appears only once per state -for any state that has lambda input, whatever top of stack symbol with it must not be used as top of stack symbol in same state for another rule
130. **Elements/Members**	Objects in a set
131. **The empty set is a regular language**	True
132. **Enqueue 1 item into a binary heap (worst case)**	logn
133. **Enqueue n items into a Binary Heap (average case)**	n
134. **estimate the worst case of adding n items into a priority Queue**	O(nlogn)
135. **estimate the worst case of finding the minimum in a binary search tree**	O(n)
136. **Every Alphabet is finite**	True
137. **Every alphabet is nonempty**	True
138. **Every computable language is partially computable.**	True
139. **Every Context free language is accepted by some Turing Machine**	True
140. **Every DFSM M, on input w, halts in at most |w| steps.**	True (Theorem)
141. **Every infinite language is uncomputable**	False
142. **Every language is accepted by some Turing Machine**	False
143. **Every language is decided by some Turing Machine**	False
144. **Every Language is finite**	False
145. **Every language is nonempty**	False
146. **every multi-tape TM has an equivalent....**	Single tape TM that runs O(t^2(n))
147. **every NTM has an equivalent**	Single tape TM that runs 2^O(t(n))
148. **Every recursively enumerable language is computable.**	False
149. **Every regular language is decided by some Turing Machine**	True
150. **Every Turing Machine accepts some language**	True
151. **Every Turing Machine decides a language**	False

152. **Exhaustive search parsing**	can be derived within 2|w| rounds of derivation
153. **The extensional definition why a problem is a language?**	a problem is a language ender the extensional definition of sets
154. **Find an item in a binary search tree (worst case)**	O(n)
155. **Find an item in an unordered array (average case)**	O(n)
156. **Find the height of a binary search tree (lab 5)**	n
157. **Find the minimum key in a binary search tree (worst case)**	n
158. **finite automaton**	a 5-tuple (Q, Σ, δ, q_0, F), where: 1. Q is a finite set called the states, 2. Σ is a finite set called the alphabet, 3. δ: Q × Σ → Q is the transition function, 4. q_0 is the start state, and 5. F ⊆ Q is the set of accept states (or final states).
159. **Finite Language**	Finite # of things in it
160. **Finite-State Language**	Language that can be solved by a finite-state machine
161. **For each NDFSM, there is an equivalent DFSM.**	True
162. **For every L, L is recursively enumerable iff L is partially Computable**	True
163. **Formal definition of a finite automaton**	a 5-tuple (Q, Sigma, delta, s, F) Q -> finite set called the states Sigma -> finite set called the alphabet delta -> function of the type Q X Sigma -> Q called the transition function s -> an element of Q called the start state F -> a subset of Q called the set of accept states
164. **formal description**	a description of a Turing machine algorithm that spells out in full the machine's states, transition function, etc.
165. **Formal Grammar Defines**	Formal Language
166. **Formal languages**	A set of strings over a given alphabet
167. **Function**	Takes an input and produces an output
168. **Function f is a mapping reduction from A to B provided**	f is computable and for all x, x E A <=> f(x) E B
169. **A function is a special relation**	True
170. **generalized nondeterministic finite automaton**	a 5-tuple (Q, Σ, δ, qstart, qaccept), where: 1. Q is the finite set of states, 2. Σ is the input alphabet, 3. δ: (Q - {qaccept}) × (Q - {qstart}) → R is the transition function, i. R is the set of all regular expressions over Σ, 4. qstart is the start state, and, 5. qaccept is the accept state.
171. **Generalized Transition Graph**	TG with labeled with regular expressions, try to remove one state at a time
172. **get (but dont delete) the minimum item in a minimum heap**	O(1)
173. **Given a key, retrieve the key/value pair in a hash table (average case)**	1

174.	**Given a list of length n, build a binary heap using the bottom up construction method**	n
175.	**Grammar**	G = (V, T, S, P), set of rules to form sentences in a language
176.	**Greibach Normal Form**	single terminal, followed by 0 or more variables exits for any CFG
177.	**The group problems of P and NP**	P and NP (separate entities) P intersect NP NP is a subset of P P is a subset of NP P = NP (they are the same problem)
178.	**halting configurations**	accepting and rejecting configurations of a Turing machine
179.	**HALTING is in NP**	False
180.	**halting problem**	the problem of determining whether a Turing machine halts on a given input
181.	**The Halting Problem can be shown to be uncomputable by a method called diagonalization.**	True
182.	**Hamiltonian cycle problem. What is it? ● Can we determine if a given graph has a Hamiltonian cycle? ● Can we verify a result?**	● A Hamiltonian cycle is a simple cycle through a graph that visits every vertex of the graph ● Can we determine if a given graph has a Hamiltonian cycle? ○ Yes, a brute-force deterministic algorithm would look at every possible cycle of the graph to see if one is Hamiltonian ■ How many possibilities for a complete graph? ● v! ○ A non-deterministic algorithm would simply return a cycle ● Can we verify a result? ○ Yes! simply look through the returned cycle and verify that it visits every vertex ■ How long will this take?
183.	**H(any) ={<M>: there exists at least one string on which TM M halts} is in SD?**	True
184.	**Hash Table (average)**	Access: N/A Search: O(1) Insertion: O(1) Deletion: O(1)
185.	**Hash Table (worst)**	Access: N/A Search: O(n) Insertion: O(n) Deletion: O(n)
186.	**Heap dequeue 1 item (best case)**	O(1)
187.	**heap dequeue 1 item (worst/average case)**	O(logn)
188.	**heap dequeue n items**	O(nlogn)
189.	**Heap enqueue 1 item (best/average case)**	O(1)
190.	**Heapsort (average)**	O(n log n)
191.	**Heapsort (best)**	O(n log n)
192.	**Heapsort Space Complexity (best)**	O(1)
193.	**Heapsort (worst)**	O(n log n)
194.	**high-level description**	a description of a Turing machine algorithm that uses English prose to describe the algorithm, without mentioning how it manages its tape or head

195.	**How can a TM be formally described?**	as a 7-tuple (Q, X, ∑, δ, q0, B, F) where - Q is a finite set of states, X is the tape alphabet, ∑ is the input alphabet, δ is a transition function, q0 is the initial state, B is the blank symbol, F is the set of final states
196.	**How can I convert a CFG into a PDA?**	Convert the productions of the CFG into GNF. The PDA will have only one state {q}. The start symbol of CFG will be the start symbol in the PDA. All non-terminals of the CFG will be the stack symbols of the PDA and all the terminals of the CFG will be the input symbols of the PDA. For each production in the form A → aX where a is terminal and A, X are combination of terminal and non-terminals, make a transition δ (q, a, A).
197.	**How can I convert a CFG into CNF?**	If the start symbol S occurs on some right side, create a new start symbol S' and a new production S' → S. Remove Null productions. (For each production A → a, construct all productions A → x where x is obtained from 'a' by removing one or multiple nullable non-terminals) Remove unit productions. (To remove A → B, add production A → x to the grammar rule whenever B → x occurs in the grammar) Replace each production A → B1...Bn where n > 2 with A → B1C where C → B2 ...Bn. Repeat this step for all productions having two or more symbols in the right side. If the right side of any production is in the form A → aB where a is a terminal and A, B are non-terminal, then the production is replaced by A → XB and X → a. Repeat this step for every production which is in the form A → aB.
198.	**How can I turn a DFA into a Regular Expression?**	Start by adding a new start and final state that point to the original. Then one state at a time, remove it and replace with an equivalent regular expression. Repeat this until only the new start and final state are left.
199.	**How can I use pumping lemma for CFGs?**	Let L be context free. Then, L must satisfy pumping lemma. At first, choose a number n of the pumping lemma. Break z into uvwxy, where $\|vwx\| \leq n$ and $vx \neq \varepsilon$. If L is a context-free language, there is a pumping length p such that any string $w \in L$ of length $\geq p$ can be written as w = uvxyz, where $vy \neq \varepsilon$, $\|vxy\| \leq p$, and for all $i \geq 0$, $uvixyiz \in L$
200.	**How can pumping lemma be used to determine if a language isn't regular?**	At first, we have to assume that L is regular. So, the pumping lemma should hold for L. Use the pumping lemma to obtain a contradiction - Select w such that $\|w\| \geq c$, Select y such that $\|y\| \geq 1$, Select x such that $\|xy\| \leq c$. Assign the remaining string to z. Select k such that the resulting string is not in L.
201.	**How can you construct a grammar that generates a language?**	Start by creating a list of accepted strings, look for the patterns and create production rules that fit those patterns.
202.	**How can you turn an NFA to a DFA?**	If there are e-transitions, You need to use subset construction.
203.	**How does a transition on a PDA look?**	a, b -> c: where a is the input symbol, b is the stack top symbol, and c is the symbol being pushed on the stack
204.	**How do you create a DFA?**	Decide what the machine needs to remember to decide if a string is in the language. At each state determine what happens when each available input symbol is read.
205.	**How do you create an NFA?**	What should you let the machine guess? Then begin developing the transition function based on the language criteria
206.	**How do you design a TM?**	assume that the input string is terminated by a blank symbol, B, at each end of the string. Let q0 be the initial start state. At each state determine what happens when the TM reads in each potential input symbol until B is reached
207.	**How do you verify if a problem is in NP?**	By demonstrating that a candidate solution can be verified in polynomial time

208.	**How is a pushdown automaton different than a finite automaton.**	Basically a "Finite state machine" + "a stack" A pushdown automaton has three components - an input tape, a control unit, and a stack with infinite size. The stack head scans the top symbol of the stack. A PDA may or may not read an input symbol, but it has to read the top of the stack in every transition
209.	**How many Regular Languages**	Countably infinite
210.	**how to convert from right-linear grammar to left-linear grammar**	Create reverse of right-linear grammar, then reverse that
211.	**how to prove two regular languages are equivalent**	find L3 = (L1 intersect L2 complement) U (L1 complement intersect L2), if L3 is empty set then they are equal
212.	**How to show a language is in NP**	find a polynomial verifier
213.	**How to show a language is In P**	DTM with an algorithm that runs on a regular computer
214.	**How to tell if a language is regular**	if it can be expressed by a DFA
215.	**If L1 and L2 are not regular Languages , then L1(intersection)L2 is not regular**	False
216.	**If L1 and L2 are regular Languages and L1 is a subset of L subset of L2 then L must be regular?**	False
217.	**If L1 and L2 are regular languages then**	L1 U L2, L1 intersect L2, L1L2, L1 bar (complement), and L1 complement are also regular
218.	**If L1 intersect L2 is in D then both L1 and L2 must be in D** **If L is in SD and its complement is context-free, then L must be in D** **If L is in SD then its complement must not be in D** **If L1 is in D and L2 is in SD then L1 intersect L2 must be in D**	False True False False

219.	**If L1 is a proper subset of L2 and L2 is CF then L1 must be context-free**	False
	If L is context-free , then L Union (not) L must be regular	True
	If L1 is not in D and L2 is regular , then it is possible the L1 intersect L2 is regular	True
	The union of two context-free languages must be in D	True
220.	**If L1 is in D and L2 is in SD then L1 intersect L2 must be in SD**	True
	If L1 and L2 are in D, then L1 - L2 must be in D	True
	If L1 and L2 are not in D, then L1-L2 cannot be regular	False
	If L1 and L2 are not in D, then L1 union L2 cannot be in D	false
	Every infinite language has a subsset that is not in D	True
	If not H where in D then every SD language would be in D	True
	{<M>:L(M) is context free} is in D	False
	{<M> :L(M) is not context free} is in D	False
	If L1 is reducible to L2 and L2 is in D then L1 is in D	True
	If L1 is reducible to L2 and L2 is in SD then L1 is in SD	True
	If L1 is reducible to L2 and L1 is not in D then L2 is not in D	True
	If L1 is reducible to L2 and L1 is not in SD then L2 is not in SD	True
	If l1 is reducible to L2 and L2 is not in SD then L1 is not in SD	False
	If L1 is reducible to L2 and L2 is not in D then L1 is not in D	M accepts w iff exists some path that accepts it, M rejects w iff all paths reject it
221.	**If L is a CFL then there is a TM which decides L**	True
222.	**if L is regular then LR**	is also regular (reverse)
223.	**If M is a deterministic finite-state machine then L(M) must be a finite set.**	False
224.	**If M is a deterministic finite-state Machine then L(M) must be a regular language**	True
225.	**if the algorithm is O(t(n)) on a nondeterministic TM than what is it on a deterministic TM**	Then it is 2^O(t(n)) on a deterministic TM
226.	**If there is a TM which accepts a language L then there must be a TM which decides L**	False
227.	**If X is a regular language then there must exist a deterministic finite-state machine M where L(M) = X.**	True

228. **implementation description**	a description of a Turing machine algorithm that uses English prose to describe the way the machine moves its head and stores data on its tape
229. **Increase the size of a hash table with n items already in it**	n
230. **indegree of A**	the number of arrows pointing to a node A
231. **inductive definition**	a definition that defines elements of a set in terms of other elements of the set
232. **An inference rule is sound If and only If**	whenever it is applied to a set A of axioms, any conclusion that it produces is entailed by A
233. **inherently ambiguous**	there is no unambiguous grammar for L
234. **inherently ambiguous language**	a context-free language that can only be generated by ambiguous grammars
235. **insert an item at the beginning (index 0) of an array (best case)**	O(n)
236. **Insert an item into a binary search tree (average case)**	logn
237. **Insert an item to a Hash table (average case)**	1
238. **Insertion Sort (average)**	O(n^2)
239. **insertion sort average case**	O(n^2)
240. **Insertion Sort (best)**	O(n)
241. **insertion sort best case**	O(n)
242. **Insertion Sort Space Complexity (worst)**	O(1)
243. **Insertion Sort (worst)**	O(n^2)
244. **insertion sort worst case**	O(n^2)
245. **integral root**	a polynomial root such that integer values are assigned to all variables
246. **The intersection of a regular language and a nonregular Language must be regular**	False
247. **The intersection of a regular language and a nonregular language must NOT be regular**	False
248. **intersection proof**	create transition table for state by state with pairs from each state
249. **Is Accepting Turning machine Turing Decidable**	No: 1. Assume Acc is Turing decidable, let's produce a contradiction. 2. Like diagonalization, we can't enumerate all langs 3. So, there must be a TM ,H, that decides Acc. 4. H can accept if M accepts w, and so on. 5. Consider H', as a representitive. Same thing, but reject if it loops. If H exist, so does H'. 6. Let D reject M if accepts <M> and accept if M rejects/loops 7. If H' exist, so does D 8. If you run D, then D accepts <D> iff D doesn't accept D. It's a contradiction
250. **is Atm decidable**	No - you can't tell id a computer is in an infinite loop neither can the machine
251. **is Atm recognizable**	Yes you know when it stope
252. **Is c++, perl a formal language**	Yes
253. **{} is computable.**	True
254. **Is every finite language computable?**	Yes, every finite language is regular and therefore computable
255. **Is Halting problem Turing Decidable**	No. It's complement is also not Turning recognizable,

256.	**Is it true that an NFA with one final state always exists?**	Yes it is true that an NFA with one final state always exists
257.	**Is it true that the reverse of any regular language is also regular?**	Yes it is true that the reverse of any regular language is also regular
258.	**is NP-complete contained in NP**	yes
259.	**is PRIMEFACTORS contained in NP**	yes, a better bounds for is BQP
260.	**Is SORT contained in NP**	Yes, a better limit for it is O(nlogn)
261.	**Is there any string on which TM M halts?**	SD/D
262.	**Is there any string that TM M accepts ?**	SD/D
263.	**K-ary function**	a function with K arguments is called a k-ary function
264.	**k-ary function**	a function with k arguments
265.	**k-ary relation**	a property whose domain is a set of k-tuples
266.	**k-regular graph**	a graph such that every node has degree k
267.	**L***	Zero or more iterations of L
268.	**L+**	One or more iterations of L
269.	**Language**	Set of strings
270.	**language**	a set of strings
271.	**Language {abcd} is a regular language.**	True
272.	**A language can be both decidable and undecidable**	False
273.	**A language L is in D**	If and only if there exists a TM M that halts on all inputs and accepts all strings in L and rejects all strings not in L
274.	**A language L is in SD**	if and only if there exists a TM M that accepts all strings in L and fails to accept every string not in L
275.	**Language not solvable by DFA**	B = {0^n 1^n | n ≥ 0} It has to keep track of the number of zeroes seen
276.	**language of M (L(M))**	the set A of all strings that machine M accepts (we say L(M) = A, M recognizes A, or M accepts A) (thought 'accepts' can be ambiguous)
277.	**Languages in P**	All regular languages and context-free languages
278.	**Languages that aren't in D**	Does TM M halt on w? Does TM M not halt on w? Does TM M halt on the empty tape? Is there any string on which TM M halts? Does TM M accept all strings? Do TMs M(a) and M(b) accept that same languages? Is the language that TM M accepts regular?
279.	**leaf**	a node of degree 1 in a tree
280.	**Left-Linear Grammar**	a -> Bx, a -> x
281.	**left-most derivation**	left-most variable replaced each time for derivation

282. **leftmost derivation**	a derivation of a string w in a grammar G such that at every step the leftmost remaining variable is the one replaced
283. **L(M), a.k.a., The language recognized by M**	the set all strings that the machine M accepts
284. **loop**	when a Turing machine does not halt on an input, it is said to ____
285. **L(R)**	the language described by a regular expression R
286. **L(r1*)**	(L(r1))*
287. **L((r1))**	L(r1)
288. **L(r1+r2)**	L(r1) U L(r2)
289. **L(r1r2)**	L(r1)L(r2)
290. **M accepts w**	Let M = (Q, Σ, δ, q_0, F) and w = $w_1w_2...w_n$ be a string over Σ. Then if ∃ a sequence of states $r_0,r_1,...,r_n$ in Q such that: 1. $r_0 = q_0$, 2. δ(r_i, w_{i+1}) = r_{i+1} for i = 0,...,n-1, and 3. $r_n \in F$, we say ____.
291. **Merge Sort**	nlogn
292. **merge sort**	O(nlogn)
293. **Mergesort (average)**	O(n log n)
294. **Mergesort (best)**	O(n log n)
295. **Mergesort Space Complexity (worst)**	O(n)
296. **Mergesort (worst)**	O(n log n)
297. **M recognizes language A**	Let A = {w | M accepts w}. Then we say ____.
298. **NDPDA = Context-free language > DPDA**	True
299. **Nearest Neighbor Heuristic**	From each city, visit the nearest city until a circuit of all cities is completed ● Runtime? ○ v2 ● Any other issues? ● How good are solutions generated by this heuristic compared to optimal solutions?
300. **Next year someone might find a TM which decides the Halting Problem**	False
301. **NFA**	M = (Q, Sigma, Delta, q_0, F) but can include lambda and no restrictions , a DFA is a NFA but not the other way around
302. **NFA extended transition function**	delta*(q_i, w) = {set of states to get there}
303. **NFA to DFA proof by construction Specials**	1. any final state from the original, any new set that contains this or these states becomes final as well 2. If original NFA accepts lambda as word then q_0 is also final
304. **Non-deterministic**	○ A conceptual algorithm with more than one allowed step at certain times and which always takes the right or best step ■ Conceptually, could run on a deterministic computer with unlimited parallel processors ● Would be as fast as always choosing the right step

305.	**Non-deterministic algorithms**	Array search: ○ Linear search: ■ Θ(n) ○ Binary search: ■ Θ(lg n) ○ Non-deterministic search algorithm: ■ Θ(1)
306.	**Nondeterministic Finite Automata (NFA)**	A 5 tuple similar to a DFA, allows e-transitions, concurrent moves (two possible transitions) and missing moves
307.	**nondeterministic finite automaton**	a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where: 1. Q is a finite set of states, 2. Σ is a finite alphabet, 3. δ: $Q \times \Sigma\varepsilon \rightarrow \wp(Q)$ is the transition function, 4. $q_0 \in Q$ is the start state, and 5. $F \subseteq Q$ is the set of accept states.
308.	**NonDeterministic Turing accepts/rejects**	Is SemiDecidable NOT DECIDABLE
	Every CF language is in D	The set of context-free languages is a proper subset of D
	The set D is closed under	Union, Intersection, complement, set difference , concatenation, Kleene Star
	SD is closed under	Union , Intersection, Concatenation, Kleene star
	A language is in D	If and only if both it and its complement are in SD
309.	**not H = {<M,w>: TM M does not halt on input string w}**	is NOT in SD
310.	**NP**	contains languages that can be decided by a NDTM, Can be verified in Polynomial time
311.	**NP**	The set of problems that can be solved by non-deterministic algorithms in polynomial time I.e., solution from a **non-deterministic algorithm** can be verified in polynomial time
312.	**NP-complete**	1. B is in NP 2. every A in NP is polynomial time reducible to B (NP hard)
313.	**NP-Complete Languages**	Subset-SUM, Set-Partition, TSP-DECIDE, Hamiltonian Path. Hamiltonian circuit, KnapSack, Bin packing
314.	**NPDA**	M = (Q, Sigma, Gamma, Delta, q0, z, F) *input tape may ignore tape symbol, may not consume input symbol describes context free language
315.	**NP is defined as the problems that cannot be decided in polynomial time on a TM**	False
316.	**One way of characterizing the class NP**	NP is contained in EXTIME(n^k)
317.	**One way of characterizing the class NP**	The Union over k of NTIME(n^k)
318.	**One way of characterizing the class NP**	languages (problems) with P-time verifiers
319.	**One way to prove that a set A is uncomputable is to show that**	H ≤m A where H is the Halting problem

320.	**outdegree of A**	the number of arrows pointing from a node A
321.	**P**	The set of problems that can be solved by **deterministic algorithms** in polynomial time
322.	**path**	a sequence of nodes connected by edges
323.	**A path P can be infinite. For FSM, DPDA, or NDFSM and NDPDA without ε transitions**	P always ends
324.	**P: contains languages that can be decided by a**	TM in polynomial time
325.	**PCP is an example of an undecidable problem**	True
326.	**PCP is in the set NP**	False
327.	**PDA without epsilon-transitions Must Halt**	True
328.	**Perform a heap sort on a random set of data**	nlogn
329.	**Perform a level order traversal of a binary tree**	n
330.	**perform quick sort on a sorted set of data if the first item is always chosen as the pivot**	O(n^2)
331.	**Perform selection sort on a random set of data**	n^2
332.	**popping**	removing a symbol from the stack
333.	**Power Set**	of A is the set of all subsets of A. EX: A = {0, 1} Power Set = {Ø, {0}, {1}, {0,1}}
334.	**Precedence of operators in Regular Expressions**	*, o, U
335.	**predicate / property**	a function whose range is {TRUE, FALSE}
336.	**Proof by construction for grammar**	given fa, states become variables and symbols become terminals
337.	**Proof by construction for reverse**	take NFA of L with one final state, make the initial sate the final state and reverse all direction of arrows
338.	**Prove a language is Turing recognizable iff it is enumerable.**	Proof of if: Suppose E enumerates L. Construct a TM M that works as follows: M = "On input w: 1. Simulate E. Every time E prints a new string, compare it with w. 2. If w is ever printed, accept." Proof of only if: Suppose M recognizes L. Let s1,s2,s3,... be the lexicographic list of all strings over the alphabet of L. Construct an enumerator E that works as follows: E = " 1. Repeat the following for i=1,2,3,... 2. Simulate M for i steps on each of the inputs s1,s2,...,si. 3. If any computations accept, print out the corresponding sj."
339.	**Prove every nondeterministic TM has an equivalent deterministic TM.**	Simulate every possible branch of computation in a breadth-first manner.

340. **Prove every t(n) multitape Turing Machine has an equivalent O(t^2 (n)) time single tape TM**	Convert a multitape TM M into an equivalent single tape TM S that simulates M The simulation of each step of M takes O(k) steps in S, where k is the length of the active content of the tape of S specifically, S makes two passes through its tape; a pass may require shifting, which still takes O(k) steps). How big can k be? Not bigger than the number of steps M takes, multiplied by the (constant) number c of tapes. That is, k <= ct(n)). Thus, S makes O(t(n)) passes through the active part of its tape, and each pass takes (at most) O(t(n)) steps. Hence the complexity of S is O(t(n)) x O(t(n)) = O(t^2 (n)).
341. **Prove HALT TM is undecidable**	Assume, for a contradiction, that HALT TM is decidable. I.e. there is a TM R that decides HALT TM. Construct the following TM S: S = "On input <M,w>, an encoding of a TM M and a string w: 1. Run R on input <M,w>. 2. If R rejects, reject 3. If R accepts, simulate M on w until it halts. 4. If M has accepted, accept; if M has rejected, reject." If M works forever on w, what will S do on <M,w>? Explicitly Reject If M accepts w, what will S do on input <M,w>? Accept If M explicitly rejects w, what will S do on <M,w>? Explicitly Reject Thus, S decides the language, which is impossible
342. **Prove If A<=mB and B is decidable, then A is decidable.**	Let DB be a decider for B and f be a reduction from A to B. We describe a decider DA for A as follows. DA= "On input w: 1. Compute f(w). 2. Run DB on input f(w) and do whatever DB does."
343. **Prove Let t(n) be a function, where t(n)>=n. Then every t(n) time single-tape nondeterministic TM has an equivalent 2^(O(t(n))) time single-tape TM.**	Convert a nondeterministic TM N into an equivalent deterministic TM D that simulates N by searching N's nondeterministic computation tree. Each branch of that tree has length at most t(n), and thus constructing and searching it takes O(t(n)) steps. And the number of branches is b^(O(t(n))), where b is the maximum number of legal choices given by N's transition function. But b<=2c for some constant c. So, the number of branches is in fact 2^(cO(t(n)))=2^(O(t(n))). Thus, the overall number of steps is O(t(n)) x 2^(O(t(n))) = 2^(O(t(n))).
344. **Prove that CLIQUE is in NP**	Here is a polynomial time NTM N deciding CLIQUE: N = "On input <G,k>: 1. Nondeterministically select a subset c of k nodes of G. 2. Test whether G contains all edges connecting nodes n c. 3. If yes, accept; otherwise reject."
345. **Prove that every multitape TM M has an equivalent single-tape TM S.**	Copy the contents of each tape in M onto the single tape of S, marking the boundaries between the tapes with a special character. Modify S so that it can move between the boundaries easily.

346. **Prove that if a language A is mapping reducible to a language B and B is recognizable, then A is recognizable**	DB is a Turing Machine that recognizes B and f is a reduction from A to B. A Turing Machine DA recognizes A DA = "on input w: 1. compute f(w) 2. run DB on input f(w) and do whatever DB does"
347. **Prove that if a language L and its complement NOT L are both recognizable, then L is decidable**	The Turing Machine U recognizes the language L The Turing Machine NOT U recognizes the language NOT L let M = "on input w: 1. run U and NOT U on input w in parallel 2. if U reaches its accept state, accept if NOT U reaches its accept state, reject therefore, M decides L
348. **Prove that SUBSET-SUM is in NP**	Here is a polynomial time NTM N deciding SUBSET-SUM: N = "On input <S,t>: 1. Nondeterministically select a subset c of S. 2. Test whether the elements of c sum up to t. 3. If yes, accept; otherwise reject."
349. **Prove that the Complement of Atm (NOT Atm) is unrecognizable**	Atm = {<M,w> | M is a TM and M accepts string w} NOT Atm = {<M,w> | M is a TM and M does not accept string w} Suppose, for a contradiction, that NOT Atm is Turing-recognizable. That is, there is a TM U that recognizes NOT Atm. Thus: U recognizes Atm (slide 4.2.a) NOT U recognizes NOT Atm Let M = "On input w: 1. Run both U and NOT U on input w in parallel; 2. If U accepts, accept; if NOT U accepts, reject." It can be seen that M decides ATM, which is impossible
350. **Prove that the halting problem is recognizable**	HALT TM = {<M,w>| M is a TM and M halts on input w} the TM U (universal Turing Machine) recognizes HALT TM: U= "On input <M,w>, where M is a TM and w is a string: 1. Simulate M on input w 2. if M enters accept state, accept if M enters reject state, accept
351. **Prove that Th(N,+,x) is unrecognizable**	Suppose a TM M recognizes Th(N,+, x). Construct a TM D: D = "On input A, and arithmetic sentence, 1. Run M on both A and NOT A in parallel. 2. If M accepts A, accept; if M accepts NOT A, reject" Obviously D decides Th(N,+, *) , which is a contradiction

352.	**Prove the Universal Turing Machine is recognizable**	The following TM U, called the universal TM, recognizes ATM: U = "On input <M,w>, where M is a TM and w is a string: 1. Simulate M on input w. 2. If M ever enters its accept state, accept; if M ever enters its reject state, reject."
353.	**Prove the Universal Turing Machine is undecidable**	Suppose, for a contradiction, that ATM is decidable. That is, there is a TM H that decides ATM. Thus, that machine H behaves as follows: H(<M,w>)= accept if M accepts w reject if M does not accept w Using H as a subroutine, we can construct the following TM D: D = "On input <M>, where M is a TM: 1. Run H on input <M,<M>>. 2. Do the opposite of what H does. That is, if H accepts, reject, and if H rejects, accept." Thus, D(<M>)= accept if M does not accept <M> reject if M accepts <M> But then D(<D>) = accept if D does not accept <D> reject if D accepts <D> contradiction! To summarize: H accepts <M,w> exactly when M accepts w. D rejects <M> exactly when M accepts M. D rejects <D> exactly when D accepts <D>.
354.	**Proving NP-Completeness**	● Show the problem is in NP ● Show that your problem is at least as hard as every other problem in NP ○ Typically done by reducing an existing NP-Complete problem to your problem ■ In polynomial time
355.	**PSPACE**	contains languages that can be decided by a machine with polynomial space
356.	**pumping lemma**	used to prove that a language is not regular
357.	**Pumping lemma for Context free grammars**	For every context-free language L, there is a number p, such that for every string s, which is in L, whose length is at least p, there are strings u, v, x, y, z, such that s = uvxyz and the following conditions are satisfied: 1.) for every i >= 0, uv^ixy^iz is in L; 2.) |vy| > 0; 3.) |vxy| <= p.
358.	**pumping lemma for context-free languages**	If A is a context-free language, there exists a number p such that if s is any string in A of length at least p, the s may be divided into five pieces s = uvxyz satisfying the following conditions: 1. For each n≥0, $uv^nxy^nz \in A$, 2. |vy|>0, and 3. |vxy| ≤ p

359.	**Pumping Lemma for Regular Expressions**	For every regular language L, there is a number p, such that for every string s, which is in L, whose length is at least p, there are strings x, y, z, such that s = xyz and the following conditions are satisfied: 1.) for every i >= 0, xy^iz is in L; 2.) |y| > 0; 3.) |xy| <= p.
360.	**Pumping Lemma for Regular Grammars**	For every regular language L, There exists a constant n For every string w in L such that |w| >= n, There exists a way to break up w into three strings w = xyz such that |y| > 0 , |xy| <= n and For every k>=0 , the string xykz is also in L.
361.	**pumping lemma for regular languages**	If A is a regular language, then there is a number p such that if s is any string in A of length at least p, then s may divided into three pieces, s =xyz, satisfying the following conditions: 1. ∀ n≥0, $xy^nz \in A$, 2. |y| > 0, and 3. |xy| ≤ p
362.	**Pushdown Automata**	can be formally described as a 7-tuple (Q, ∑, S, δ, q0, I, F) - Q is the finite number of states, ∑ is input alphabet, S is stack symbols, δ is the transition function, q0 is the initial state (q0 ∈ Q), I is the initial stack top symbol (I ∈ S), F is a set of accepting states (F ∈ Q)
363.	**pushdown automaton**	a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$, where: Q, Σ, Γ, and F are all finite sets, and: 1. Q is the set of states, 2. Σ is the input alphabet, 3. Γ is the stack alphabet, 4. δ: $Q \times \Sigma\varepsilon \times \Gamma\varepsilon \rightarrow \wp(Q \times \Gamma\varepsilon)$ is the transition function, 5. $q_0 \in Q$ is the start state, and 6. $F \subseteq Q$ is the set of accept states.
364.	**pushing**	writing a symbol on the stack
365.	**Queue (average)**	Access: O(n) Search: O(n) Insertion: O(1) Deletion: O(1)
366.	**Queue (worst)**	Access: O(n) Search: O(n) Insertion: O(1) Deletion: O(1)
367.	**Quick sort**	nlogn
368.	**Quicksort (average)**	O(n log n)
369.	**quick sort average case**	O(nlogn)
370.	**Quicksort (best)**	O(n log n)
371.	**Quick sort best case**	O(nlogn) (O(n) at each partition, and O(logn) paritions
372.	**Quicksort Space Complexity (worst)**	O(log n)
373.	**Quicksort (worst)**	O(n^2)
374.	**quick sort worst case**	O(n^2) (O(n) work at each partition and O(n) partitions)
375.	**Recursive Link based Lists calculate number of items if no num_items variable**	O(n)
376.	**Recursive Link based Lists size of list**	O(n)
377.	**Red black tree search, remove, insert**	O(logn)
378.	**Reduction of SAT to independent set**	SAT->3-SAT->INDEPENDENT-SET

379.	**Reduction of SAT to TSP**	SAT->3-SAT->Hamiltonian-circuit->TSP
380.	**REDUCTION of SAT to vertex-cover**	SAT->3-SAT->Vertex-cover
381.	**Regular Expression**	operators = {+, **or next to each other for concat,** , ()} denotes a regular language
382.	**regular expression**	R is a ____ if R is either: 1. a for some a in the alphabet Σ, 2. ε, 3. ∅, 4. ($R_1 \cup R_2$), where R_1 and R_2 are regular expressions, 5. ($R_1 \circ R_2$), where R_1 and R_2 are regular expressions, or 6. R_1^*, where R_1 is a regular expression. (OOO is *, ∘, ∪)
383.	**Regular Expressions**	Can be the empty string, empty set/language, individual input symbols, X + Y = L(X) ∪ L(Y), X. Y = L(X). L(Y), R **= (L(R))**. Ex: (a + b)*abb = Set of strings of a's and b's ending with the string abb, So L = {abb, aabb, babb, aaabb, ababb, ...}
384.	**Regular Grammar**	Have left side non-terminal and right side lots
385.	**Regular Grammar**	G = (V, T, S, P) it is either right-linear or left-linear *not both, if both then linear not regular
386.	**Regular Language**	a language where some finite automaton recognizes it
387.	**regular language**	a language that is recognized by some finite automaton
388.	**Regular Language closure**	Union, Concatenation, Kleene Star, Complement, Intersection, Difference, Reverse, Letter substitution
389.	**Regular Languages Can be recognized**	Finite State Machines
390.	**regular operations**	where A and B are languages, the operations: Union: $A \cup B = \{x \mid x \in A \text{ or } x \in B\}$ Concatenation: $A \circ B = \{xy \mid x \in A \text{ and } y \in B\}$ Star: $A^* = \{x_1x_2...x_k \mid k \geq 0 \text{ and each } x_i \in A\}$
391.	**rejecting configuration**	a Turing machine configuration where the current state is qreject
392.	**A relation is a set of ordered pairs**	True
393.	**remove from the beginning of a link based list**	O(n)
394.	**remove from the beginning of a link based list**	O(1)
395.	**remove from the middle of a link based list**	O(n)
396.	**Rice's Theorem**	contains only even/odd number of strings, contains all strings that start with a, is infinite , is regular
397.	**Right-Linear Grammar**	a -> xB, a -> x
398.	**right-most derivation**	right-most variable replaced each time for derivation
399.	**S − S = ∅ for every set S.**	True

400.	**S ∪ S = S for every set S**	True
401.	**SAT**	NP-complete
402.	**Satisfiable problems**	PRIMEFACTORS, HAMPATH, K-CLIQUE
403.	**SAT is in the set NP**	True
404.	**selection sort**	n^2
405.	**Selection Sort (average)**	O(n^2)
406.	**Selection Sort (best)**	O(n^2)
407.	**Selection sort on already sorted data**	O(n^2)
408.	**Selection Sort Space Complexity (worst)**	O(1)
409.	**Selection Sort (worst)**	O(n^2)
410.	**A sentence w is unsatisfiable**	If and only if not w is valid
411.	**A sentence w is Valid**	IF and only IF it is true in all interpertations
412.	**Set**	Group of objects represented as a unit
413.	**Show that PATH is in P**	A polynomial time algorithm M for PATH works as follows: On input <G,s,t>, where G is a directed graph with nodes s and t, 1. Place a mark on node s 2. Repeat until no additional nodes are marked: 3. Scan all edges of G. If an edge (a,b) is found going from a marked node a to an unmarked node b, mark node b. 4. if t is marked, accept. Otherwise, reject
414.	**Show that RELPRIME is in P**	E = "On input <x,y>, where x and y are natural numbers, x>y: 1. Repeat until y=0. 2. Assign x <- x mod y. 3. Exchange x and y. 4. Output x." R = "On input <x,y>, where x and y are natural numbers: 1. Swap x and y if necessary so that x>y. 2. Run E on <x,y>. 3. If the result is 1, accept. Otherwise reject.
415.	**Show that the Atm is Turing reducible to HALT TM**	For every TM M with input string w, consider the TM M*, which takes input <M,w> M* = "on input <M,w> 1. simulate M on w 2. if M reaches accept, accept 3. if M reaches reject, enter an infinite loop f is the computable function through which f(<M,w>) = <M*,w> Therefore, <M,w> is in Atm iff f(<M,w>) is HALT TM

416.	**Show the nonregularity of B = {0n1n | n>=0}**	Proof by contradiction: Assume B is regular. Let then p be its pumping length. Select w is in B with |w| >= p. By the pumping lemma, w=xyz and y can be pumped, so that we must also have xyyz is in B. Case 1: y only has 0s. But then xyyz has more 0s than 1s and is not in B. Case 2: y only has 1s. But then xyyz has more 1s than 0s and is not in B. Case 3: y has both 0s and 1s. But then xyyz has a 1 followed by a 0 and is not in B. All cases lead to contradiction, therefore B is not regular
417.	**Show the nonregularity of F = { ww | w is in {0,1}* }**	Proof by contradiction: Assume F is regular. Let then p be its pumping length. Observe that 0p10p1 is in F. By the pumping lemma, 0p10p1 =xyz and y can be pumped. By Condition 3, |xy|<=p. Therefore, y is entirely in the first 0p. Pumping y would produce a string that has only 0s is the first half, and two 1s in the second half. Obviously this string cannot be in F, which contradicts with the pumping lemma.
418.	**simple cycle**	a cycle that doesn't repeat any nodes (other than first and last being equal)
419.	**Simple Grammar**	form of A -> ax where a is a terminal and x is zero or more variables any pair (A, a) occurs at most once in P can be parsed in |w|
420.	**simple path**	a path that doesn't repeat any nodes
421.	**Singleton Set**	Set with one member
422.	**Singly-Linked List (average)**	Access: O(n) Search: O(n) Insertion: O(1) Deletion: O(1)
423.	**Singly-Linked List (worst)**	Access: O(n) Search: O(n) Insertion: O(1) Deletion: O(1)
424.	**Some problems cannot be decided on a Turing Machine in polynomial time**	True
425.	**Some Turing Machines don't accept a language**	False
426.	**Splay Tree (average)**	Access: N/A Search: O(log n) Insertion: O(log n) Deletion: O(log n)
427.	**Splay Tree (worst)**	Access: N/A Search: O(log n) Insertion: O(log n) Deletion: O(log n)
428.	**Stack (average)**	Access: O(n) Search: O(n) Insertion: O(1) Deletion: O(1)
429.	**Stack (worst)**	Access: O(n) Search: O(n) Insertion: O(1) Deletion: O(1)
430.	**start configuration**	the Turing machine configuration q_0w, where w is the input string
431.	**string over an alphabet**	a finite sequence of symbols from an alphabet
432.	**String over Sigma**	A finite sequence of symbols from sigma
433.	**strongly connected directed graph**	a directed graph such that a directed path connects every two nodes
434.	**symbol**	an element of an alphabet
435.	**Symbols**	Members of an alphabet
436.	**Theorem: For Each DFSM M**	there is an equivalent NDFSM M'

437.	**Theorem: For each NDFSM**	there there is an equivalent DFSM
438.	**There are uncountably many non-regular languages over (Epsilon)={a,b}**	True
439.	**There is a TM that accepts {a^2^i}**	True
440.	**TM**	has tape instead of stack read/write head M = (Q, Sigma, Gamma, delta, q0, square, F) abstract representation of a simple, modern, digital computer *want it to reach halt state aq3ba signifies that q3 points to b -can be transducer or accepter
441.	**A TM can have an infinite number of states**	False
442.	**TM in complement of SD**	accept all strings? ,Comparison of accepting languages, Not halt on any string, does not halt on its own description,
443.	**TM vs Finite Automaton**	TM: -has unlimited and unrestricted memory -can read and write on tape -head can move to the left and to the right -tape is infinite
444.	**A TM with an even number of final states must accept an even number of words**	False
445.	**tree**	a connected graph with no simple cycles
446.	**Tree Sort (average)**	O(n log n)
447.	**Tree Sort (best)**	O(n log n)
448.	**Tree Sort Space Complexity (worst)**	O(n)
449.	**Tree Sort (worst)**	O(n^2)
450.	**Turing-decidable language**	a language that some Turing machine decides (also called decidable or recursive language)
451.	**Turing machine**	a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ where Q, Σ, Γ are all finite sets and: 1. Q is the set of states, 2. Σ is the input alphabet not containing the special blank symbol $\sqcup$, 3. Γ is the tape alphabet, where $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$, 4. $\delta: Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is the transition function, 5. $q_0 \in Q$ is the start state, 6. $q_{accept} \in Q$ is the accept state, and 7. $q_{reject} \in Q$ is the reject state, where $q_{reject} \neq q_{accept}$
452.	**A Turing Machine can have at most 3 final states**	False
453.	**Turing machines are capable of deciding all languages.**	False

454.	**Turing machines are capable of deciding languages that are not regular.**	True
455.	**A Turing machine with 3 tapes can decide a language that cannot be decided by any Turing machine with 2 tapes.**	False
456.	**Turing-recognizable language**	a language that some Turing machine recognizes (also called recursively enumerable language)
457.	**Turing Recognizable / Recursively Enumerable**	Language is Turing Recognizable if some TM recognizes it AKA Recursively Enumerable Language
458.	**Type 0 Grammar**	generate recursively enumerable languages. The productions have no restrictions. They are any phase structure grammar including all formal grammars. They generate the languages that are recognized by a Turing machine.
459.	**Type 1 Grammar**	generate context-sensitive languages. The productions must be in the form α A β → α γ β, where A ∈ N (Non-terminal) and α, β, γ ∈ (T ∪ N)* (Strings of terminals and non-terminals). The strings α and β may be empty, but γ must be non-empty. The languages generated by these grammars are recognized by a linear bounded automaton.
460.	**Type 2 Grammar**	Generates context-free languages. The productions must be in the form A → γ, where A ∈ N (Non terminal) and γ ∈ (T∪N)* (String of terminals and non-terminals). These languages generated by these grammars are be recognized by a non-deterministic pushdown automaton.
461.	**Type 3 Grammar**	Generates regular languages. The productions must be in the form X → a or X → aY, where X, Y ∈ N (Non terminal) and a ∈ T (Terminal). The rule S → ε is allowed if S does not appear on the right side of any rule.
462.	**Types of derivations of strings**	Leftmost derivation - A leftmost derivation is obtained by applying production to the leftmost variable in each step. The leftmost derivation for the string "a+a*a" may be - X → X+X → a+X → a+ X**X** → **a+a**X → a+a*a Rightmost derivation - A rightmost derivation is obtained by applying production to the rightmost variable in each step. The rightmost derivation for the above string "a+a**a" may be – X** → **X**X → X**a** → **X+X**a →X+a**a** → **a+a**a
463.	**A U B**	Union - All elements in A or B
464.	**unambiguous grammar implies**	unambiguous language
465.	**Unary function**	When k is 1
466.	**Undecidable Languages**	there is no Turing Machine which accepts the language and makes a decision for every input string w. Examples: The halting problem of Turing machine, The mortality problem
467.	**The union of a finite number of regular Languages must be regular**	True
468.	**The union of an infinite number of regular languages must be regular**	False
469.	**The union of an infinite number of regular languages must NOT be regular**	False
470.	**Unordered Pair**	Set with two members

471.	**Using the pumping Theorem**	If L is regular, then every long string in L is pumpable
472.	**The value of a regular expression is**	A Language
473.	**We say that A ≤m B if**	There exists a mapping reduction from A to B
474.	**What are decidable problems**	Adfa, Edfa, EQdfa, Acfg, Ecfg
475.	**What are the 3 problems in Turning machine**	1. Accepting 2. Halting 3. Empty
476.	**What are the different approaches to drawing a derivation tree?**	Top-down approach: Starts with the starting symbol S. Goes down to tree leaves using productions Bottom-up approach: Starts from tree leaves. Proceeds upward to the root which is the starting symbol S
477.	**What does U uses**	Uses a fixed, finite set of states, and set of alphabet symbols, but still stimulates TMs w/ arbitrarily many states and symbols
478.	**What exactly is NP-Complete?**	NP-Complete problems are the "hardest" problems in NP ● They are all equally "hard" ○ So, if we find a polynomial time solution to one of them, we clearly have a polynomial time solution to all problems in NP
479.	**What if P = NP**	● Most widely-used cryptography would break ○ Efficient solutions would exist for: ■ Attacking public key crypto ■ Attacking AES/DES ■ Reversing cryptographic hash functions ● Operations research and management science would be greatly advanced by efficient solutions to the travelling salesman problem and integer programming problems ● Biology research would be sped up with an efficient solution to protein structure prediction ● Mathematics would be drastically transformed by advances in automated theorem proving
480.	**What if P != NP**	meh Mostly assumed to be the case
481.	**What is a derivation tree?**	an ordered rooted tree that graphically represents the semantic information of a string derived from a context-free grammar. Root vertex - Must be labeled by the start symbol. Vertex - Labeled by a non-terminal symbol. Leaves - Labeled by a terminal symbol or ε.
482.	**What is a Grammar?**	It is a 4 tuple where there is a set of non-terminal symbols, a set of terminal symbols, a start symbol, and production rules for the non-terminal and terminal symbols
483.	**What is a multi-tape Turing Machine?**	can be formally described as a 6-tuple (Q, X, B, δ, q0, F) where - Q is a finite set of states, X is the tape alphabet, B is the blank symbol, δ is a relation on states and symbols, where there is k number of tapes, q0 is the initial state, F is the set of final states
484.	**What is a Turing Machine (TM)?**	a mathematical model which consists of an infinite length tape divided into cells on which input is given. It consists of a head which reads the input tape. A state register stores the state of the Turing machine. After reading an input symbol, it is replaced with another symbol, its internal state is changed, and it moves from one cell to the right or left. If the TM reaches the final state, the input string is accepted, otherwise rejected.

485.	**What is decidability?**	A language is called Decidable or Recursive if there is a Turing machine which accepts and halts on every input string w. If a language L is decidable, then its complement L' is also decidable.
486.	**what is in the class NP?**	SAT, CLIQUE, HAMPATH, VERTEXCOVER, SUBSET-SUM
487.	**What is NP-hard?**	"At least as hard as the hardest problems in NP"
488.	**What is reducibility?**	A way of converting one problem to another problem, so that the solution to the second problem can be used to solve the first problem
489.	**What is subset construction?**	A method to convert NFAs to DFAs. 1. Create a state table. 2. Add start state from NFA. 3. Find out the combination of states for each possible input symbol. 4. Repeat step 3 until you have run out of possible DFA states
490.	**What is the Accepting Problem**	U recognizes Accept, but it doesn't decide. -If M loops forever, U loops as well. -To decide, U would detect when M is looping and reject
491.	**What is U**	U is a universal Turning machine b/c it runs all TMs.
492.	**What language is not recognizable?**	complement Atm
493.	**What makes a language decidable**	when the language and the complement of the language is Turing-recognizable
494.	**What problems belong in the NP class?**	Complex problems that can be verified in polynomial time. contains problems we would love to solve but are unable to do so exactly. It means that it is possible to "guess" the solution (by some nondeterministic algorithm) and then check it
495.	**What problems belong in the P class?**	Problems that can be decided by an algorithm whose running time is bounded by a polynomial. contains all of the problems we solve using computers.
496.	**When are context free languages closed?**	Union, Concatenation, Kleene Star operation
497.	**When does a PDA accept a string?**	Final State Acceptance: after reading the entire string, the PDA is in a final state. The stack values are irrelevant as long as we end up in a final state. Empty Stack Acceptance: after reading the entire string, the PDA has emptied its stack.
498.	**When is a language accepted or decided?**	Accepted: if it enters into a final state for any input string Decided: if it accepts it and enters into a rejecting state for any input not in the language. A language is recursive if it is decided by a Turing machine
499.	**When is a problem NP-complete?**	If the problem is in the NP class and every problem in NP can be reduced to it in polynomial time
500.	**yields**	if a Turing machine can legally go from configuration C_1 to configuration C_2 in a single step, we say C_1 ___ C_2