UNIQNAME (print): ________________________________

EECS 376 Midterm Exam, Spring 2024

Instructions:

This exam is closed book, closed notebook. No electronic devices are allowed. You may use one 8.5×11-inch study sheet (both sides) that you prepared yourself. The last few pages of the exam are scratch paper; you may not use your own. Make sure you are taking the exam at the time slot and location you were assigned by the staff. ***Please print your UNIQNAME at the top of each page.***

Any deviation from these rules may constitute an honor code violation. In addition, the staff reserves the right **not** to grade an exam taken in violation of this policy.

The exam consists of **9 multiple-choice** questions, **4 short-answer** questions, and **2 longer-answer** questions. For the short- and longer-answer sections, please write your answers clearly in the spaces provided. If you run out of room or need to start over, you may use the blank pages at the end, but you **MUST** make that clear in the space provided for the answer. The exam has 24 pages printed on both sides, including this page and the blank pages at the end.

Leave all pages stapled together in their original order.

Sign the honor pledge below.

Pledge:
I have neither given nor received aid on this exam, nor have I concealed any violations of the Honor Code.

I will not discuss the exam with anyone until 10pm on Thursday May 30th (once every student in the class has taken the exam, including alternate times).

I attest that I am taking the exam at the time slot and the location I was assigned by the staff.

Signature: ________________________

Print clearly:

Full Name: ________________________

Uniqname: ________________________

UNIQNAME (print): ______________________________

Standard Languages

You may rely on the following definitions without repeating them.

- Σ^* = The set of all finite-length strings
- $\emptyset$ = An empty set
- $L_{\text{HALT}} = \{(\langle M \rangle, x) : M$ is a Turing machine and M halts on $x.\}$
- $L_{\text{ACC}} = \{(\langle M \rangle, x) : M$ is a Turing machine and M accepts $x.\}$
- $L_{\text{BARBER}} = \{\langle M \rangle : M$ is a Turing machine and M does not accepts $\langle M \rangle.\}$

UNIQNAME (print): ______________________________

Single Answer Multiple Choice (16 points, 4 points each)

For each of the problems in this section, select **the one** correct option. Each one is worth 4 points; **no partial credit is given**.

1. There are N students in a room standing in a straight line, and there are N seats that are also in a straight line parallel to the students' line. The students can move in any direction, taking one step at a time, and each step takes one second. The task is to assign the students to seats so that the time for all students to reach their assigned seat is minimized. The algorithm finds the student that is closest to an unoccupied chair (breaking ties arbitrarily), assigns them to that chair, and then repeats the process on the remaining students.

   Which of the following is an accurate description for the paradigm used by this algorithm?

   - [✓ **Greedy**]
   - ○ Dynamic programming
   - ○ Both divide and conquer and greedy
   - ○ Both dynamic programming and greedy

   Solution: At no point in the algorithm is the problem of assigning N students to N seats subdivided into smaller subproblems, therefore this algorithm has no divide-and conquer-aspects to it.
   There is no overlapping subproblems or memoization or tabulation of solutions to subproblems of any kind either, so this algorithm is certainly not a dynamic-programming algorithm.
   At every step, the algorithm looks for a locally optimal decision to make (for every student, it finds the closest unoccupied chair without considering other students), therefore this is a greedy algorithm, and greedy only.

2. Let $f(n, m)$ be the number of length n strings over the alphabet $\{0, 1, 2, 3\}$ with **exactly** m 2's. Which recurrence relation accurately describes $f(n, m)$?

   - ○ $f(n, m) = f(n-1, m-1) + f(n-1, m)$
   - [✓ $f(n, m) = f(n-1, m-1) + 3f(n-1, m)$]
   - ○ $f(n, m) = 2f(n-1, m-1) + 2f(n-1, m)$
   - ○ $f(n, m) = 3f(n-1, m-1) + 2f(n-1, m)$

   Solution: We will count this by cases of what the last character in the string is.

   First, count the number of ways we can have such a string that ends in a 2. We can construct these strings by adding a 2 to the end of any string of length $n-1$ that has $m-1$ 2's in it. This contributes $f(n-1, m-1)$ to $f(n, m)$.

   Next, count the number of ways to have such a string that ends in 0, 1, or 3. We can construct these strings by adding a 0, 1, or 3 to the end of any string of length $n-1$ with m 2's in it. This contributes $f(n-1, m)$ for each of 0, 1, and 3, for a total contribution of $3f(n-1, m)$ to $f(n, m)$.

   Since these cases are exhaustive and mutually exclusive, we have that $f(n, m) = f(n-1, m-1) + 3f(n-1, m)$.

3. Suppose we want to use a **one-tape Turing machine** to decide the following language:

   $$L_{\text{sameOne}} = \{(b_1, b_2) : b_1 \text{ and } b_2 \text{ are binary strings with the same number of 1's.}\}.$$

   What is the **smallest** possible size of Γ, the **tape alphabet** needed to decide L_{sameOne}?

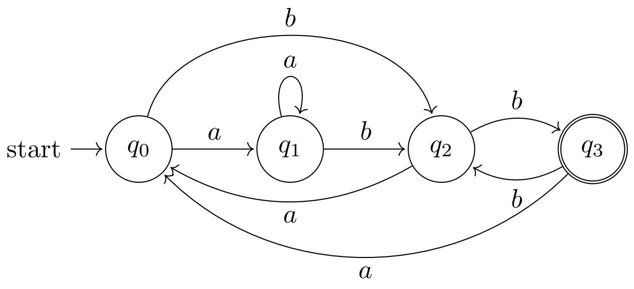
○ 1
○ 2
○ 3
[✓ **4**]

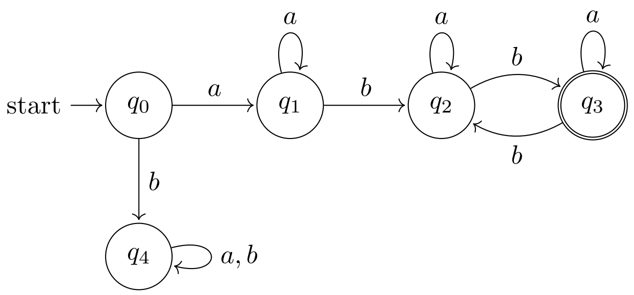
Solution: Since we have two binary strings, we have `0` and `1`, along with a special character as the separator between the two strings. So, the smallest possible size of Σ is 3. Then, $\Gamma = \Sigma \cup \{\perp\}$, so the smallest possible size of Γ is 4.

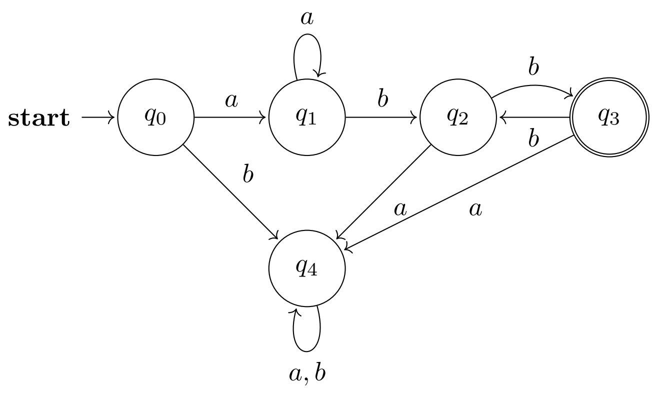
4. Consider the following language that decides if a string over the alphabet $\Sigma = \{a, b\}$ has n a's followed by $2m$ b's, where $n, m \geq 1$:

$$L_{anb2m} = \{a^n b^{2m} : n, m \geq 1\}.$$

Which of the following is a valid DFA that decides L_{anb2m}?

○


○


[✓]


○ No such DFA exists.

Solution:

- Option A is incorrect. Consider the string "bb". It should be rejected because it does not start with an 'a', but its accepted by the DFA.
- Option B is incorrect because it does not enforce the language's constraint that all a's must precede the b's. For example, consider the string $abba$, which will be accepted by the DFA while it shouldn't.
- Option D is incorrect because the language L_{anb2m} as it can be expressed using the regular expression $a^+(bb)^+$ and therefore, a DFA must exist to decide it.

Multiple Answer Multiple Choice (30 points, 6 points each)

For each of the problems in this section, select **all** valid options; this could be all of them, none of them, or something in between. **If none of the options applies, clearly write <u>None</u> on the line next to the question.** For each option, a correct (non-)selection is worth one point. In addition, for each problem you will earn one additional point (for a total of six) if all five of your (non-)selections are correct.

1. Suppose you are given three algorithms with the following running time for the worse case for input of size n:

   - Algorithm X has running time $T_X(n) = O(n^2)$
   - Algorithm Y has running time $T_Y(n) = \Theta(n \log n)$
   - Algorithm Z has running time $T_Z(n) = \Theta(n)$

   Which of the following claim(s) is/are **necessarily** true? ______________________

   ● **We can upper bound the runtime of Y by $O(n^2)$.**
   ○ On every input, Y runs faster than X.
   ○ On every input, Z runs faster than Y.
   ○ For all large enough n, there is an input of size n for which Y runs faster than X.
   ● **For all large enough n, there is an input of size n for which Z runs faster than Y.**

Solution:

- Option A is necessarily true because $T_Y(n) = \Theta(n \log n) \implies T_Y(n) = \Omega(n \log n)$ and $O(n \log n)$. Since $n \log n = O(n^2)$, we can indeed upper bound the runtime of Y by n^2.
- Option B is not necessarily true; it can be false since X might be faster than Y for some *small* inputs (i.e., small input size n). For example, it could be the case that $T_X(n) = n^2$ and $T_Y(n) = 100n \log_2 n$. Then on *any* input of size $n = 2$, X runs in time *at most* $T_X(2) = 4$. By contrast, on a *worst-case* input of size $n = 2$, Y runs in time *exactly* $T_Y(2) = 100 \cdot 2 \cdot 1 = 100$. So, X is faster than Y on such an input.
- Option C is not necessarily true; it can be false since Y might be faster than Z or some *small* inputs (i.e., small input size n). For example, it could be the case that $T_Y(n) = n \log_2 n$ and $T_Z(n) = 2n$. Then on *any* input of size $n = 2$, Y runs in time *at most* $T_Y(2) = 2 \log_2(2) = 2$. By contrast, on a *wort-case* input of size $n = 2$, Z runs in time *exactly* $T_Z(s) = 2^2 = 4$. So, Y is faster than Z on such an input.
- Option D is not necessarily true; it can be false because the $O(n^2)$ upper bound on $T_X(n)$ could be very *loose*, and X might actually run in much faster than quadratic time (for all inputs). For example, it could be the case that $T_X(n) = 10 = O(n^2)$, whereas Y takes time $100n \log_2(2n) = \Theta(n \log n)$ for *every* input of size n. Then because $100n \log_2(2n) \geq 100n > 10$ for every $n \geq 1$, X is faster than Y on *every* input.
- Option E is necessarily true. For each value of n, there is some *worst-case* size-n input y_n for Y, i.e., Y run on y_n takes time exactly $T_Y(n)$. Then because $T_Y(n) = \Omega(n \log n)$, there is a positive constant $c > 0$ such that for all large enough n, Y run on y_n takes time *at least* $cn \log_2 n$. Because $T_Z(n) = O(n)$, there is a positive constant $c' > 0$ such that for all large enough n, Z run on y_n takes time *at most* $c'n$. Finally, because $c'n < cn \log_2 n$ for all large enough n (namely, when $n > 2^{c'/c}$), we conclude that Z runs faster than Y on y_n for all large enough n.

UNIQNAME (print): ______________________________

2. Consider the following function, which takes in non-negative integers a and b that are powers of two.

```
function ALG($a$,$b$)
    if $a = 1$ or $b = 1$ or $a = b$ then return 0
    if $a > b$ then return ALG($a/2, 2b$)
    if $b > a$ then return ALG($2a, b/2$)
```

Which of the following function(s) can be used as a potential function to prove that the algorithm ALG halts on all valid inputs a, b? ____**None**____

- ◯ $a + b$
- ◯ $a \cdot b$
- ◯ a/b
- ◯ $|b - a|$
- ◯ $2a + 2b$

Solution: To use the potential method, it would have to be the case that we can create a function with a and b as inputs such that the potential decreases on each iteration of ALG. Nevertheless, if we set $a = 4$, $b = 8$, then after one iteration, we have $a = 8$, $b = 4$, but after two iterations we again have $a = 4$, $b = 8$. Since a, b take on identical values in round 0 and round 2, then there cannot exist a potential function that is monotonically decreasing, therefore, no function can exist as a potential function for ALG. It is also worth noting that the algorithm does not terminate on input $(4, 8)$ as well as many other inputs.

UNIQNAME (print): ______________________

3. Recall the bottom-up dynamic programming algorithm for solving the 0-1 Knapsack problem.

```
Input: Integers $n, C$, arrays $W, V$.
Output: The maximum total value of objects the Knapsack can hold.
function KNAPSACK($n, C, W, V$)
    Initialize an empty table $DP$
    for $i = 0$ to $n$ do
        $DP[i][0] \leftarrow 0$
    for $j = 0$ to $C$ do
        $DP[0][j] \leftarrow 0$
    for $i = 1$ to $n$ do
        for $j = 1, \ldots, C$ do
            if $W[i] > j$ then
                $DP[i][j] \leftarrow DP[i-1][j]$
            else
                $DP[i][j] \leftarrow$ the greater of either taking item $i$ or not taking it
    return $DP[n][C]$
```

Select all correct statements about the algorithm. ______________________

- ○ It takes $O(C)$ to compute the value on line 12 as it requires considering all previous capacity values smaller than j to find the maximum.
- ● **Every column of the DP table will always hold a sequence of values in nondecreasing order.**
- ● **Every row of the DP table will always hold a sequence of values in nondecreasing order.**
- ● **The bottom-up algorithm is asymptotically more time-efficient than the top-down recursive algorithm, but it uses more space than top-down recursive approach.**
- ○ The bottom-up algorithm is efficient (i.e., runs in polynomial time) with respect to the input size.

Solution:

- Option A is incorrect because the maximum value at each entry $DP[i][j]$ is determined by comparing only two values: the value of not taking the current item (i.e., $DP[i-1][j]$) and the value of taking the current item (i.e., $DP[i-1][j - W[i]] + V[i]$). This comparison is constant time, $O(1)$, for each i and j.
- Option B is correct because column j in the DP table represents the maximum value achievable with a knapsack capacity j. As we increase the number of items available from 1 to n, the value that can be achieved with the same capacity cannot decrease.
- Option C is correct because row i of the DP table represents the maximum values achievable by considering the first i items. As we increase the capacity available from 1 to C, the value that can be achieved with the same number of items available cannot decrease.
- Option D is correct because the bottom-up algorithm runs in $O(nC)$, whereas the top-down recursive algorithm runs in $O(2^n)$; the bottom-up algorithm need $O(nC)$ of auxiliary space, whereas the top-down recursive algorithm only needs $O(n)$ auxiliary space

- Option E is incorrect. Recall that the input size of an integer k is $O(\log k)$. Hence, the input size is $O(\log n) + O(\log C) + 2O(n)$, and $O(nC)$ is exponential with respect to the input size.

4. Which of the following set(s) is/are **countably infinite**? ____________

   ○ The set of languages over the alphabet $\{3, 7, 6\}$.

   ○ The set of DFAs that do not decide any language.

   [✓ **The set of all Turing machines.**]

   [✓ **The set of all Turing machines that are deciders.**]

   ○ The power set of the set of all Turing machines.

Solution:

- Option A is incorrect because languages can be represented as infinite strings. Therefore using the Cantor's diagonalization argument, the set of languages over the alphabet $\{3, 7, 6\}$ is uncountable. (The role of the alphabet here is not important, as any alphabet can be written in binary.)
- Option B is incorrect because there does not exist a DFA that decides no language at all; at minimum, every DFA decides either the empty language or some language with at least one string. Hence, the set of DFAs that do not decide any language is $\emptyset$, which is not infinite.
- Option C is correct because every Turing machine can be represented as a finite length string. There are countably infinite finite length strings.
- Option D is correct because it is just a subset of C that is countable. Moreover, there are infinitely many Turing machines that are deciders, hence the set is countably infinite.
- Option E is incorrect because the power set of a countably infinite set is uncountable, as proven in the homework.

UNIQNAME (print): ______________________________

5. Define the language $L_{\text{BOTH}} = \{(\langle M_1\rangle, \langle M_2\rangle) :$ there is an input that is accepted by **both** M_1 and M_2.$\}$. Using a black-box (oracle) decider D_{BOTH} that decides L_{BOTH}, the following (incomplete) decider for L_{ACC} is used to show that $L_{\text{ACC}} \leq_T L_{\text{BOTH}}$.

```
$D_{\text{ACC}}$ = "on input $(\langle M\rangle, x)$:
  1: Construct a Turing machine $M_1$ as follows:
       $M_1$ = "on input $w$:
         1: **if** $w \in \{3, 7, 6\}$ **then accept**
         2: **reject**"
  2:     Construct another Turing machine $M_2$ as follows: $\boxed{\textbf{Missing part}}$
  3:     **if** $D_{\text{BOTH}}(\langle M_1\rangle, \langle M_2\rangle)$ accepts **then accept**
  4:     **else reject**"
```

Which of the following description(s) of M_2 is/are suitable for the reduction? ______________

○
```
$M_2$ = "on input $w$:
  1: **if** $w \in \{3, 7, 6\}$ **then accept**
  2: **reject**"
```

●
```
$M_2$ = "on input $w$:
  1: Run $M$ on $x$
  2: **if** $M(x)$ accepts **then accept**
  3: **reject**"
```

○
```
$M_2$ = "on input $w$:
  1: Run $M$ on $x$
  2: **if** $M(x)$ accepts **then reject**
  3: **accept**"
```

○
```
$M_2$ = "on input $w$:
  1: Run $M$ on $x$
  2: **if** $M(x)$ accepts **then**
  3:     **if** $w \in \{3, 7, 0\}$ **then accept**
  4: **else**
  5:     **if** $w \in \{2, 1, 7\}$ **then accept**
  6: **reject**"
```

●
```
$M_2$ = "on input $w$:
  1: Run $M$ on $x$
  2: **if** $M(x)$ accepts **then**
  3:     **if** $w \in \{3, 7, 0\}$ **then accept**
  4: **else**
  5:     **if** $w \in \{2, 8, 1\}$ **then accept**
  6: **reject**"
```

Solution: We want our decider to work as follows: D_{ACC} accepts if and only if $(\langle M\rangle, x) \in L_{\text{ACC}}$. Since D_{ACC} will return the same as D_{BOTH}, we should have D_{BOTH} accepts if and only if $(\langle M\rangle, x) \in L_{\text{ACC}}$. In other words, we want

- $(\langle M\rangle, x) \in L_{\text{ACC}} \implies M(x)$ accepts $\implies ... \implies (\langle M_1\rangle, \langle M_2\rangle) \in L_{\text{BOTH}} \implies D_{\text{BOTH}}(\langle M_1\rangle, \langle M_2\rangle)$ accpets $\implies D_{\text{ACC}}(\langle M\rangle, x)$ accepts
- $(\langle M\rangle, x) \notin L_{\text{ACC}} \implies M(x)$ does not accept $\implies ... \implies (\langle M_1\rangle, \langle M_2\rangle) \notin L_{\text{BOTH}} \implies D_{\text{BOTH}}(\langle M_1\rangle, \langle M_2\rangle)$ rejects $\implies D_{\text{ACC}}(\langle M\rangle, x)$ rejects

Note that $(\langle M_1\rangle, \langle M_2\rangle) \in L_{\text{BOTH}}$ means $L(M_1) \cap L(M_2) \neq \emptyset$. Now, we examine which description of M_2 would yield $(\langle M_1\rangle, \langle M_2\rangle) \in L_{\text{BOTH}}$ if and only if $(\langle M\rangle, x) \in L_{\text{ACC}}$.

- Option A is incorrect because $L(M_1) = L(M_2)$ regardless of whether $M(x)$ accepts, so $(\langle M_1\rangle, \langle M_2\rangle) \in L_{\text{BOTH}}$ regardless of whether $(\langle M\rangle, x) \in L_{\text{ACC}}$, which is not desired.
- Option B is correct because in this case,
  - $M(x)$ accepts $\implies L(M_2) = \Sigma^* \implies L(M_1) \cap L(M_2) = \{3, 7, 6\} \neq \emptyset \implies (\langle M_1\rangle, \langle M_2\rangle) \in L_{\text{BOTH}}$
  - $M(x)$ does not accepts $\implies L(M_2) = \emptyset \implies L(M_1) \cap L(M_2) = \emptyset \implies (\langle M_1\rangle, \langle M_2\rangle) \notin L_{\text{BOTH}}$
- Option C is incorrect because in this case,
  - $M(x)$ accepts $\implies L(M_2) = \emptyset \implies L(M_1) \cap L(M_2) = \emptyset \implies (\langle M_1\rangle, \langle M_2\rangle) \notin L_{\text{BOTH}}$
  - $M(x)$ does not accepts $\implies L(M_2) = \Sigma^* \implies L(M_1) \cap L(M_2) = \{3, 7, 6\} \neq \emptyset \implies (\langle M_1\rangle, \langle M_2\rangle) \in L_{\text{BOTH}}$

  which is opposite of the desired behavior.
- Option D is incorrect because in this case, $L(M_1) \cap L(M_2) \neq \emptyset$ regardless of whether $M(x)$ accepts, so $(\langle M_1\rangle, \langle M_2\rangle) \in L_{\text{BOTH}}$ regardless of whether $(\langle M\rangle, x) \in L_{\text{ACC}}$, which is not desired.
- Option E is correct because in this case,
  - $M(x)$ accepts $\implies L(M_2) = \{3, 7, 0\} \implies L(M_1) \cap L(M_2) = \{3, 7\} \neq \emptyset \implies (\langle M_1\rangle, \langle M_2\rangle) \in L_{\text{BOTH}}$
  - $M(x)$ does not accepts $\implies L(M_2) = \{2, 8, 1\} \implies L(M_1) \cap L(M_2) = \emptyset \implies (\langle M_1\rangle, \langle M_2\rangle) \notin L_{\text{BOTH}}$

UNIQNAME (print): ____________________

Short Answer (24 points, 6 points each)

1. Given an array A of length n, we say that an array C is a *circular shift* of A if there exists an integer k between 1 and n (inclusive) such that

$$C = [A[k], A[k+1], \ldots, A[n], A[1], A[2], \ldots, A[k-1]]$$

For example, if $A = [1,4,5,7,8]$, then $C = [5,7,8,1,4]$ is a circular shift of A with $k = 3$.

Consider the following (incomplete) algorithm that takes in an array C which is a circular shift of a **sorted** array with distinct positive integers, and returns the value of the **maximum value** in C.

```
Input: $C$ : $C$ is a circular shift of a sorted array of distinct positive integers.
Output: The maximum value in $C$.
1: function FindMax($C[1,\ldots,n]$)
2:     $n \leftarrow$ length of $C$
3:     if $C[1] < C[n]$ then return $C[n]$
4:     $m \leftarrow \lfloor n/2 \rfloor$
5:     if $C[m] > C[1]$ then ____ return findMax($C[m,\ldots,n]$) ____
6:     else ____ return findMax($C[1,\ldots,m-1]$) ____
```

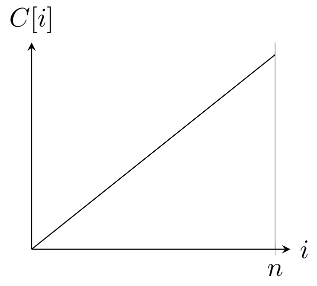
Fill in the two missing parts of the pseudocode to make it a correct algorithm. **You do not need to justify its correctness.** In addition, provide the recurrence relation and the tightest asymptotic bound (big-O) on the algorithm's runtime.

Recurrence relation: ____ $T(n) = T(n/2) + O(1)$ ____

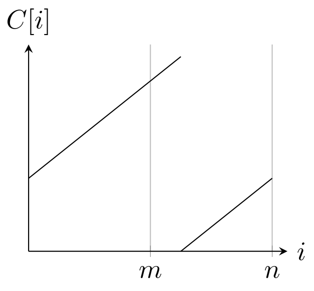
Asymptotic runtime: ____ $O(\log n)$ ____

Solution: Since C is a circular shift of a sorted array, there are three possible cases:

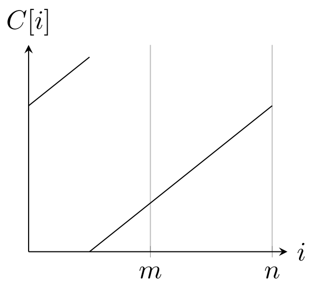
1. $C[1] < C[n]$: In this case, C must be equal to A, so the maximum value is $A[n] = C[n]$.



2. $C[m] > C[1]$: The maximum value must be to the right of $C[m]$ (or C[m] itself). Hence, we recurse into the right subarray.



3. $C[m] < C[1]$: The maximum value must be to the left of $C[m]$. Hence, we recurse into the left subarray.



Note that on each recursive call we either enter line 4 or 5, not both, so the recurrence relation is $T(n) = T(n/2) + O(1)$. Then by second case of the Master Theorem, we have $O(n^0 \log n) = O(\log n)$.

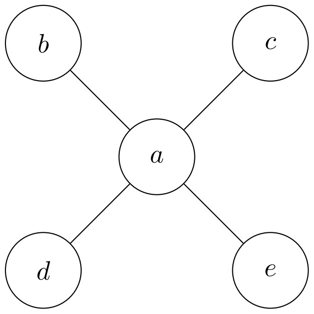
2. Suppose L_1 is a decidable language while L_2 is an undecidable language. Is $L_1 \setminus L_2$ (always/ sometimes/ never) decidable? **Justify your answer.** *Recall that* $A \setminus B = A \cap \bar{B}$.

Solution: Sometimes. Consider the case where $L_1 = \Sigma^*$, so $L_1 \setminus L_2 = \overline{L_2}$. We know that this is undecidable because the complement of an undecidable language is undecidable. Now consider the case where $L_1 = \emptyset$, so $L_1 \setminus L_2 = \emptyset$ which is trivially decidable.

3. An *independent set* of a graph $G = (V, E)$ is a subset of vertices $S \subseteq V$ such that no two vertices in S are adjacent.

   We are interested in a **maximum** independent set, i.e., the independent set with the **most** number of vertices. (There may be more than one maximum independent set.)

   For example, the following graph has a maximum independent set $S = \{b, c, d, e\}$: every vertex in S are not connected to each other.



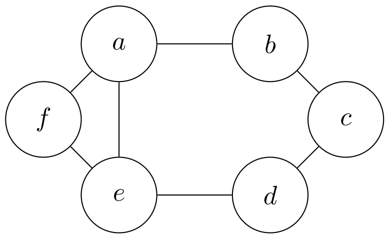
Consider the following greedy algorithm for finding an independent set in a graph.

```
1: function GreedyIS($G = (V, E)$)
2:     $S \leftarrow \emptyset$
3:     while $G$ has at least one vertex do
4:         Select any vertex $v$ in $G$ that has smallest degree (i.e., the least neighbors)
5:         Add $v$ to $S$
6:         Remove $v$ and all its neighbors, including all incident edges, from $G$
7:     return $S$
```

Give a small graph G on which the algorithm might not return a maximum independent set. Additionally, give a sequence of vertices that the algorithm might choose to make up its final output set, and give a maximum independent set of G that is larger than this output set. Note that multiple maximum independent sets may exists. In that case, just give any *one* of them.

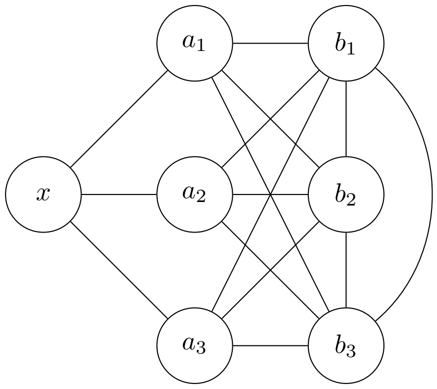
Your graph here:

Solution: There are many different graphs on which the algorithn does not *necessarily* give a maximum independent set. Here is one example:



Possible sequence of vertices chosen by the algorithm: $\{c, f\}$; maximum independent set: $\{b, e, f\}$.

Here's another example that doesn't depend on how the algorithm breaks ties. The idea is as follows: we have three sets of nodes $\{x\}, \{a_1, \ldots, a_k\}$, and $\{b_1, \ldots, b_k\}$. The vertex x is linked to every a_i, every a_i is linked to every b_j, and the b_j's form a fully-connected subgraph. On this graph, the algorithm will first pick x, remove all $\{a_1, \ldots, a_k\}$, then pick a node in $\{b_1, \ldots, b_k\}$, and stop. The independent set returned has size 2, but $\{a_1, \ldots, a_k\}$ is an independent set of size k. Therefore, if $k > 2$, the algorithm will not return the maximum independent set. For example, let $k = 3$ we have

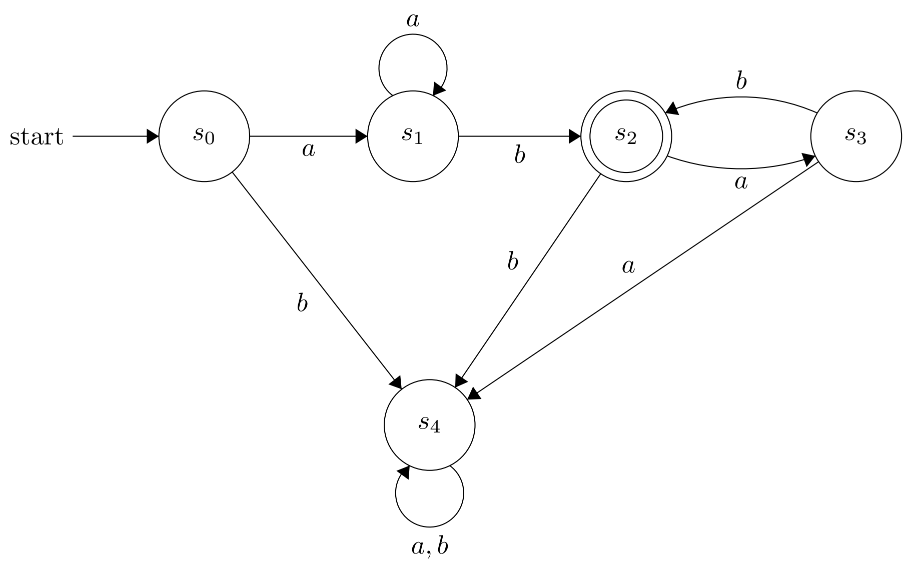


Possible sequence chosen by the algorithm: $\{x, b_1\}$; maximum independent set: $\{a_1, a_2, a_3\}$.

4. **In this problem, you may describe the language in words or regular expression.**

   For each of the following DFA below, determine the languages decided by the DFA.

   (a)



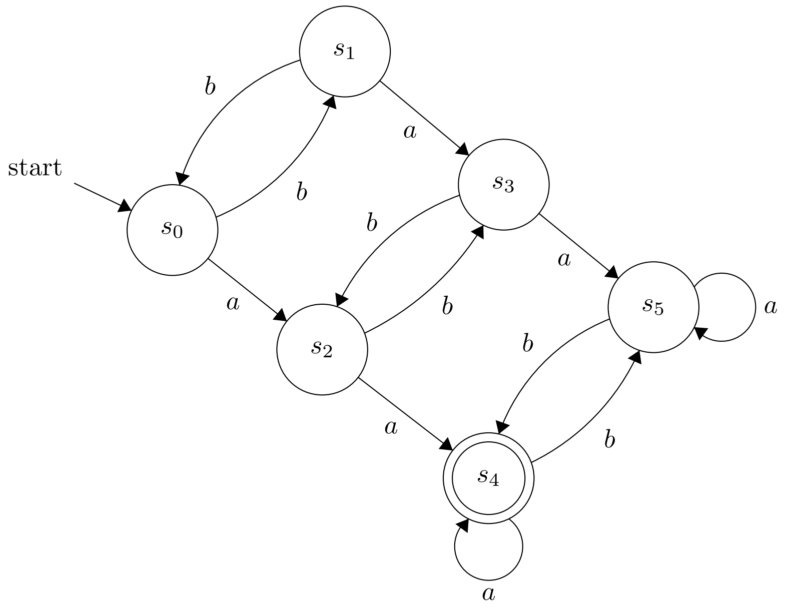
Solution: The DFA decides the language $L = a^+b(ab)^*$.

First, note that there is no escape from s_4 so every string that visits s_4 will not be accepted.

There is one accept state s_2. The only path from the initial state s_0 to s_2 should visit s_1 first. The DFA visits s_1 if and only if the input string starts with more than 1 a's. Then, one more b will transition to the accept state s_2. Finally, any number of repetition of ab from s_2 will alternate between s_2 and s_3.

Therefore, the language decided by the DFA is $\{a^n b(ab)^m : n \geq 1, m \geq 0\} = a^+b(ab)^*$. Alternative regular expression includes $a^*(ab)^+$, $aa^*b(ab)^*$.

(b)



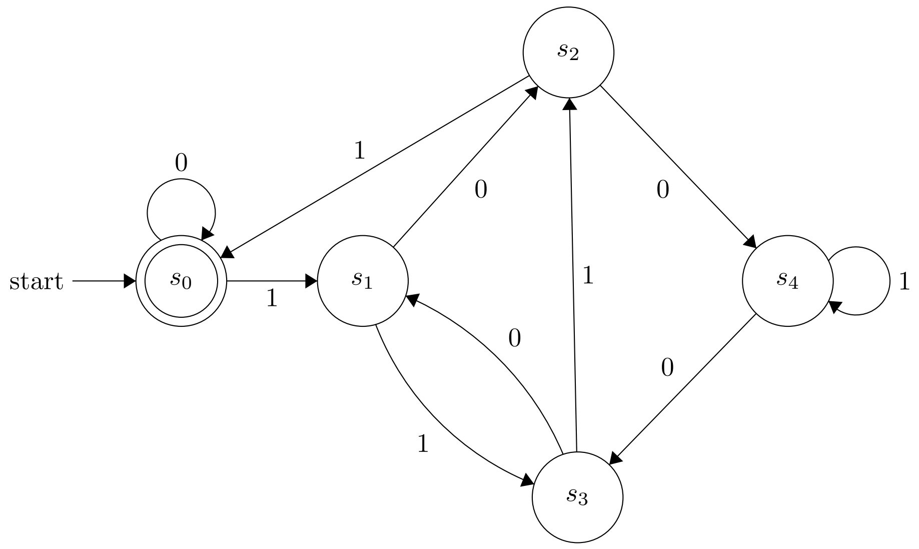
Solution: The DFA decides the language

$$L = \{s \in \{a,b\}^* : s \text{ has at least two } a\text{'s and even number of } b\text{'s.}\}.$$

First, note that the character a transitions towards bottom right from $\{s_0, s_1\}$ to $\{s_2, s_3\}$ and then to $\{s_4, s_5\}$ while b alternates between $\{s_0, s_2, s_4\}$ and $\{s_1, s_3, s_5\}$.
There is only one accept state s_4. To reach s_4, the DFA should transition bottom right (by seeing a) at least twice. Moreover, the DFA also needs to see b for even number of times to stay in $\{s_0, s_2, s_4\}$.
Therefore, the language decided by the DFA is

$$\{s \in \{a,b\}^* : s \text{ has at least two } a\text{'s and even number of } b\text{'s.}\}.$$

(c)



Solution: The DFA decides the language

$$L = \{s \in \{0,1\}^* : s \text{ is a binary representation of a number that is divisible by 5.}\}.$$

First, note that there are 5 states in total. Moreover, the character 0 transitions s_i into s_j with $j = 2i \mod 5$ while the character 1 transitions s_i into s_k with $j = 2i + 1 \mod 5$. Therefore, the state s_i represents that the binary number read so far equals i modulo 5.
Since the accept state is s_0, the language decided by the DFA is

$$\{s \in \{0,1\}^* : s \text{ is a binary representation of a number that is divisible by 5.}\}.$$

UNIQNAME (print): ______________________________

Long Answer (30 Points, 15 points each)

1. Recall that the longest increasing subsequence (LIS) of a sequene of numbers is the longest subsequence where each element is greater than the preceding one, whereas the longest common subsequences (LCS) of two arrays is the longest sequence that can be derived from both arrays in the same order without rearranging them.

   Now we consider the combination of the two: Given two arrays of n distinct integers A and B, we are interested in the *length* of the longest common subsequence that is also increasing, refer to as the longest common increasing subsequence (LCIS). For example, if we have $A = [5, 3, 8, 9, 1]$ and $B = [2, 3, 4, 9, 1]$, then the LCIS is $[3, 9]$, and the length is 2. Note that multiple LCISs may exists.

   (a) Let $L(i, j)$ = length of LCIS of $A[1, \dots, i]$ and $B[1, \dots, j]$ that ends at $B[j]$ (we do **not** enforce it to end at $A[i]$ here). Define a recurrence relation for $L(i, j)$, including base case(s), that is suitable for a dynamic-programming solution for this problem. Briefly justify its correctness.

   Solution:

$$L(i,j) = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ \max\{L(i-1,j), L(i,j-1)\} & \text{if } A[i] \neq B[j] \\ 1 + \max_{t:1\leq t<j, B[t]<B[j]} L(i-1,t) & \text{otherwise} \end{cases}$$

   - Base case: When $i = 0$ or $j = 0$, there is no common increasing subsequence so $L(i, j) = 0$.
   - If $A[i] \neq B[j]$: The LCIS up to $A[i]$ and $B[j]$ cannot include both elements since they don't match. Therefore, the problem reduces to finding the longest among the LCIS that ends up to $A[i-1]$ and $B[j]$ or $A[i]$ and $B[j-1]$.
   - If $A[i] = B[j]$: We seek to extend the sequence by finding the maximum length of any LCIS that can be extended by $B[j]$. This is done by considering all indices t where $B[t] < B[j]$ (ensuring an increasing sequence) and is before j.

   (b) Based on your recurrence relation in (a), give pseudocode for a **bottom-up** dynamic programming algorithm for this problem. Analyze the running time of the algorithm, as a function of n.

   Solution:

```
Input: Arrays of distinct integers $A[1,\dots,n]$ and $B[1,\dots,n]$
Output: Length of LCIS of $A$ and $B$
function LCIS($A, B$)
    Initialize an $(n+1)\times(n+1)$ table $DP$
    for $i = 0$ to $n$ do
        $DP[i][0] \leftarrow 0$
    for $j = 0$ to $n$ do
        $DP[0][j] \leftarrow 0$
    for $i = 1$ to $n$ do
        for $j = 1$ to $n$ do
            if $A[i] \neq B[j]$ then
                $DP[i][j] \leftarrow \max\{DP[i-1][j], DP[i,j-1]\}$
            else
                $DP[i][j] \leftarrow 1 + \max_{t:1\leq t<j, B[t]<B[j]} L(i-1,t)$
    return $\max_{j:1\leq j\leq n} DP[n,j]$
```

Runtime analysis: The nested for loops on lines 7-12 dominates the running time of the algorithm, which yields $O(n^3)$.

(c) Describe how you would modify your algorithm in (b) to reduce the *space complexity* from $O(n^2)$ to $O(n)$. You may refer to specific lines in your pseudocode in (b), but you do not need to rewrite the whole pseudocode.

Solution: We need two arrays `current` and `previous` of length $n + 1$ each. Initialize all elements of `current` and `previous` to 0 as in the base case.

At the start of each new iteration of i, swap `current` and `previous`. This makes the last computed values available for the next row's computation, effectively retaining only the necessary data from the previously completed step.

For each $j = 1, \dots n$, if $A[i] \neq B[j]$, update `current[j]` to the maximum between `current[j-1]` and `previous[j]`.

If $A[i] = B[j]$, find the maximum LCIS ending with any $B[t]$ where $t < j$ and $B[t] < B[j]$ from `previous` and add 1 to it. This captures extending the LCIS to $B[j]$.

After processing all i and j, the length of longest LCIS can be found as the maximum value in the `current` array.

A pseudocode (not required to receive full credit) for the above modification is as follows:

```
Input: Arrays of distinct integers $A[1, \dots, n]$ and $B[1, \dots, m]$
Output: Length of LCIS of $A$ and $B$
1: function LCIS($A, B$)
2:     Initialize arrays previous[0...m] and current[0...m] to 0
3:     for $i = 1$ to $n$ do
4:         Swap current and previous
5:         for $j = 1$ to $m$ do
6:             if $A[i] \neq B[j]$ then
7:                 current[j] $\leftarrow$ max(current[j-1], previous[j])
8:             else
9:                 $max\_val \leftarrow 0$
10:                for $t = 1$ to $j - 1$ do
11:                    if $B[t] < B[j]$ then
12:                        $max\_val \leftarrow \max(max\_val,$ previous[t]$)$
13:                current[j] $\leftarrow 1 + max\_val$
14:    return max(current[1...m])
```

UNIQNAME (print): ________________________________

2. **In this problem, you may describe your Turing machines in words or pseudocode.**
Consider the function defined by

$$f(n) = \begin{cases} 3n+1 & \text{for odd } n \\ n/2 & \text{for even } n \end{cases}$$

for any natural number n. Iterating this function starting from any n generates a sequence:

$$n, f(n), f(f(n)), f(f(f(n))), \ldots.$$

The iteration stops when the sequence reaches 1. For example, starting from $n = 13$ produces the sequence 13, 40, 20, 10, 5, 16, 8, 4, 2, 1. This problem, where it is conjectured that all positive starting points eventually reach 1, is known as the *Collatz Conjecture*, or the $3n + 1$ problem.

(a) Construct a Turing machine, M_{query}, that halts on an input n, where n is a natural number, if iterating f starting from n eventually results in 1, otherwise it loops indefinitely. **Yo do not need to justify its correctness.**

Solution: There are many ways to construct such Turing machine. For example, we could use a while-loop

```
$M_{\text{query}}$ = "on input $n$:
1: while $n \neq 1$ do
2:     if $n$ is odd then
3:         $n \leftarrow 3n + 1$
4:     else
5:         $n \leftarrow n/2$
6: accept"                          ▷ Could be reject here, as long as it halts
```

(b) Suppose that L_{HALT} is decidable by a decider H.
Using H and M_{query} from (a), construct a Turing machine M_{search} that halts if there exists a natural number n for which iterating f starting from n does **not** reach 1 (a counterexample to the Collatz Conjecture), otherwise it loops indefinitely. **You do not need to justify its correctness.**

Solution: There are many ways to construct such Turing machine. For example, we could use a while-loop:

```
$M_{\text{search}}$ = "on input $w$:
1: Ignore $w$
2: $n \leftarrow 1$
3: while $H$ accepts $(\langle M_{\text{query}} \rangle, n)$ do
4:     $n \leftarrow n + 1$
5: accept"                          ▷ Could be reject here, as long as it halts
```

Alternatively, we could also use a for-loop instead

```
$M_{\text{search}}$ = "on input $w$:
1: Ignore $w$
2: for $n = 1, 2, 3, \ldots$ do
3:     Run $H$ on $(\langle M_{\text{query}} \rangle, n)$
4:     if $H$ rejects then
5:         accept"                  ▷ Could be reject here, as long as it halts
```

(c) Again, suppose that L_{HALT} is decidable by a decider H.
Using H and M_{search} from (b), construct a Turing machine, M_{3n+1}, that accepts if the Collatz Conjecture is true (all natural numbers terminate at 1) and rejects otherwise. **Justify its correctness** and conclude that if L_{HALT} were decidable, then we can prove/disprove the Collatz Conjecture.

Solution:

```
$M_{3n+1}$ = "on input $v$:
1: Ignore $v$
2: Run $H$ on $(\langle M_{\text{search}} \rangle, \varepsilon)$        ▷ Could be anything, doesn't have to be $\varepsilon$
3: if $H$ accepts then
4:     reject
5: accept"
```

Since H is a decider by assumption, it will always halt. Therefore, M_{3x+1} will halt on all inputs.

- If Collatz Conjecture is true $\implies M_{\text{query}}$ halts for all $n \in \mathbb{N} \implies M_{\text{search}}$ will loop on $\varepsilon \Rightarrow H(\langle M_{\text{loop}}, \varepsilon \rangle)$ rejects $\implies M_{3n+1}$ accepts
- If Collatz Conjecture is false $\implies \exists n' : M_{\text{query}}(n')$ loops $\implies H(\langle M_{\text{query}} \rangle, n')$ rejects $\implies M_{\text{search}}(\varepsilon)$ accepts $\implies H(\langle M_{\text{search}} \rangle, \varepsilon)$ accepts $\implies M_{3n+1}$ rejects

Therefore, if L_{HALT} were decidable (and hence H exists), we can use it to construct M_{search} and M_{3n+1}, which decides the problem "whether the Collatz Conjecture is true."

UNIQNAME (print): ______________________________

This page is intentionally left blank for scratch work.

If you have answers on this page, write “answer continues on page 22” in the corresponding solution box.

UNIQNAME (print): ______________________________

This page is intentionally left blank for scratch work.

If you have answers on this page, write “answer continues on page 23” in the corresponding solution box.

UNIQNAME (print): ____

This page is intentionally left blank for scratch work.

If you have answers on this page, write “answer continues on page 24” in the corresponding solution box.