Monday, February 9, 2026 Instructor: Cynthia Bailey

CS106B Midterm Exam - Winter 2026

Name:	SUID (numeric):

All SUIDs	ADTs	ADTs	Recursion	Big-O	TOTAL
1pt	12pts	19pts	15pts	12pts	59pts

Instructions:

- **The time for this exam is 2 hours**, or 120 minutes. **Time limits are strict**, and you are expected to do *all* writing (including, e.g., writing SUID on each page), *before* time is called, or lose all points.
- Use of anything other than a pencil, eraser, pen, and one 8.5x11 page (two sides) of notes, is prohibited. In particular, no digital devices of any kind are permitted.
- Scratch writing areas are included in the exam, and **no additional scratch paper is permitted**.
- **Write your SUID on each page, as marked**, in case pages become separated during scanning.
- For coding problems, you may assume `#include` and function prototypes already exist for you as needed.
- Please **write in boxes provided**, or the automated grading software in Gradescope will not see your response.

Stanford Honor Code:

> The Honor Code is an undertaking of the Stanford academic community, individually and collectively. Its purpose is to uphold a culture of academic honesty. Students will support this culture of academic honesty by neither giving nor accepting unpermitted academic aid on this examination.

This course is participating in the proctoring pilot overseen by the Academic Integrity Working Group (AIWG), therefore proctors will be present in the exam room. The purpose of this pilot is to determine the efficacy of proctoring and develop effective practices for proctoring in-person exams at Stanford.

Please sign before you begin: I agree to abide by the spirit and letter of the Honor Code, and to follow the instructions above.

Signature:

Exam Break Sign-out:

I pledge that during my exam break:

- I will not bring any paper, electronic devices (phone, smart watch, smart glasses, etc) *out of or into* the exam room, or access any aid (permitted or unpermitted).
- I will not use AI or communicate with anyone other than the course staff about the content of the exam.

Signature Confirming Honor Code	Exit Time	Return Time	Proctor Initial	Length (mins)

SUID:

1. ADTs: Stacks and Queues (12 points)

Consider the following code, with the input **value of `v` as shown in the first `cout`**. As a reminder for you, below is what the Stanford Library Documentation webpages say about the format when you `cout <<` these structures, and we have marked the answer boxes for you accordingly.

- **`Queue`**: "The elements are listed left-to-right **from the front of the queue to the back**, such as `{value1, value2, value3}`."
- **`Stack`**: "The output is in the form `{value1, value2, value3}` where elements are listed left-to-right **from the bottom of the stack to the top**."
- **`Vector`**: "Elements are listed **in order of ascending index**."

```
void collectionMystery(Vector<int>& v) {
    int N = v.size();          // For use in Problem 4 (Big-O).
    cout << v << endl;         // For use in Problem 1, output is {5, 4, 6, 3, 1, 2}
    Queue<int> q;
    for (int i = 0; i < v.size(); i+=2) {
        q.enqueue(v[i]);
        q.enqueue(v[i]);
    }
    cout << q << endl;         // (a) Checkpoint 1
    Stack<int> s;
    while (!q.isEmpty()) {
        s.push(q.dequeue());
    }
    cout << s << endl;         // (b) Checkpoint 2
    v.remove(0);
    for (int i = 0; i < v.size(); i+=2) {
        q.enqueue(v[i]);
    }
    cout << q << endl;         // (c) Checkpoint 3
    Vector<int> v2;
    while (!s.isEmpty()) {
        v2.add(s.pop());
        s.pop();
        v2.add(q.dequeue());
    }
    cout << v2 << endl;        // (d) Checkpoint 4
}
```

(scratch space–problem 1 continues on next page)

(a) (3pts) Write the output of the line marked Checkpoint 1 in this box:

Front { } *Back*

(b) (3pts) Write the output of the line marked Checkpoint 2 in this box:

Bottom { } *Top*

(c) (3pts) Write the output of the line marked Checkpoint 3 in this box:

Front { } *Back*

(d) (3pts) Write the output of the line marked Checkpoint 4 in this box:

{ }

(scratch space–end of problem 1)

SUID:

2. ADTs: BTS is coming! (19 points)

BTS is coming to Stanford stadium in May, and tickets sold out almost instantly! You've been asked to help write software to manage the re-sale website, where ticketholders can set a price to sell their tickets (like StubHub). The layout of seating in the stadium has been simplified to a rectangular shape, and will be stored in a `Grid<int> prices`, where the rows of the Grid are rows of seats starting at the stage, and the number of columns is the number of seats per row. Each value in the Grid is the ticketholder of that seat's asking price (in dollars; a non-negative value; zero is offering the ticket for free), or the constant `NOT_FOR_SALE`, which is defined for you at the top of your file as `const int NOT_FOR_SALE = INT_MAX;`.[1] Here is an example Grid (not the actual size, so use the usual Grid class functions to find the dimensions of the Grid you are given):

`prices`:
(Example)

	0	1	2	3
0	NOT_FOR_SALE	NOT_FOR_SALE	NOT_FOR_SALE	NOT_FOR_SALE
1	500	450	NOT_FOR_SALE	NOT_FOR_SALE
2	NOT_FOR_SALE	275	125	300
3	NOT_FOR_SALE	100	130	95
4	35	99	NOT_FOR_SALE	NOT_FOR_SALE

For convenience, we also keep track of the seat with the cheapest price in each row, by maintaining a `Vector<int> bestInRow`, where `bestInRow[row]` is the **price** of the cheapest seat in that row, or the constant `NOT_FOR_SALE` if no seats are available in that row. Here is what `bestInRow` would look like for the above `prices` example (again, this may not be the actual size):

`bestInRow`:
(Example)

0	1	2	3	4
NOT_FOR_SALE	450	125	95	35

(scratch space–problem 2 continues on next page)

[1] `INT_MAX` is a special value defined in the C++ language as the largest possible `int` value the computer can hold (it's over 9 quintillion). For this problem, assume all actual ticket prices will be far less than `INT_MAX`. (Even BTS isn't *that* popular. 😅)

(a) (4pts) Given a buyer's budget limit, return the number of the **closest row** to the stage where at least one ticket is available for the **`budget`** price or cheaper, or `NOT_FOR_SALE` if no tickets anywhere fit the `budget`. For this function, we want you to show good judgement about which data structure is the better choice for doing the job, so you should **check one box for your choice of function header**, and then write the code accordingly. For simplicity, you may assume `budget >= 0`.

```
☐ int closestForBudget(int budget, Grid<int>& prices) {
☐ int closestForBudget(int budget, Vector<int>& bestInRow) {









}
```

(b) (7pts) Write a function that processes the sale of one ticket, by setting its place in `prices` to `NOT_FOR_SALE` and updating `bestInRow`, if needed. If the provided `seat` is out of bounds or not currently for sale, use `error("invalid seat")` to report it.

```
void sellTicket(GridLocation seat, Grid<int>& prices, Vector<int>& bestInRow) {









}
```

SUID:

(c) (8pts) Now you'll add a group seating feature. Given a group `size`, return all the blocks of adjacent seats (in the same row) that match that `size` (for simplicity, you may assume that `size > 0`). You'll collect and return this information in a `Map<GridLocation, int>`, where the **key** is the leftmost seat in the block, and the **value** is the total price for that block of `size` tickets. If there is no matching block, return an empty `Map`. Note that if a sequence of available seats within a row is wider than `size`, that means there are multiple matching blocks there, and you should include all of them (all the `GridLocation` values where a block of width `size` could start should be included in the keys of the `Map`). Write your answer on the next page.

`prices`:
(Example)

	0	1	2	3
0	NOT_FOR_SALE	NOT_FOR_SALE	NOT_FOR_SALE	NOT_FOR_SALE
1	500	450	NOT_FOR_SALE	NOT_FOR_SALE
2	NOT_FOR_SALE	275	125	300
3	NOT_FOR_SALE	100	130	95
4	35	99	NOT_FOR_SALE	NOT_FOR_SALE

Example: For `size = 2` and the `prices` example shown above, the `Map` you return should contain the following: `{ r1c0:950, r2c1:400, r2c2:425, r3c1:230, r3c2:225, r4c0:134 }`. *One block is circled to illustrate the connection.*

(scratch space–problem 2 continues on next page)

```
Map<GridLocation, int> getGroupOptions(int size, Grid<int>& prices) {









}
```

SUID:

3. Recursion: Dorm Struggles (15 points)

Your dorm has a tradition of going to Stanford games together and holding up signs that spell out encouraging messages, where each person is holding a sign with one letter painted on it. For example, if 10 people are going to the game, they might bring 10 signs, spelling "GOSTANFORD." Unfortunately, some of the people in the dorm are flakes, and when they don't show up, the remaining people are left scrambling to figure out what, if any, valid English words can be spelled with only some of the planned letters. For example, if 2 of the 10 people who had planned "GOSTANFORD" flake, they could do "TORNADOS" (10 - 2 = 8 letters) instead. Or, under more dire circumstances, "FROGS" (5 flakes, 5 letters), "ODOR" (6 flakes, 4 letters), or "OOF" (7 flakes, 3 letters). You've decided to write a function to assist in these circumstances:

```
Set<string> signRescue(string plan, int nFlakes);
```

Instructions and hints:

- **`plan`** is the message that had been planned, so its length is the number of people who originally planned to attend. For simplicity, you may assume it contains only uppercase letters (no spaces, punctuation, etc).
- **`nFlakes`** is the number of people who did not show up. For simplicity, you may assume that `nFlakes <= plan.length()`.
- **Return a set of all the allowed strings** that can be made with length that is exactly the remaining number of people. So for our "GOSTANFORD" example, if `nFlakes = 6`, include "ODOR" (assuming `isAllowed("ODOR")` is `true`) and not "OOF" (too short–one of the people who did show up wouldn't get to hold a sign).
- To test whether a string would be allowed as a new sign possibility (i.e., detect English words), assume you have access to a function **`bool isAllowed(string candidate)`** that returns `true` if `candidate` would be allowed for a sign, and `false` otherwise. For simplicity, you may assume that `isAllowed()` accepts input that is in all uppercase letters.
- You may want to implement a recursive helper function for `signRescue()`. In that case, write both functions in the provided box.
- Your solution must not use any global variables, and must be fundamentally recursive in approach.
- There is no specific Big-O requirement, but excessively repetitious execution may lose points.

(scratch space–problem 3 continues on next page)

```
Set<string> signRescue(string plan, int nFlakes) {
```

SUID:

4. Big-O (12 points)

Give the Big-O analysis as we learned in class (i.e., tight bound worst-case runtime complexity class) for each of the following code fragments, **in terms of the variable `N`**. As a reminder, when doing Big-O analysis, we write a simple expression, such as $O(N^2)$ or O(log N), *not a precise count*. Write your answer in the boxes on the right side. For each problem, the N that you use for your analysis is defined as the value of the variable `N` shown in the code. If running the code will make an infinite loop or stack overflow, **write a large X in the answer box**. Each part is worth 2pts.

	Code Fragment	**Answer**

(a)	`void functionA(int N) {`
	`  Vector<int> v;`
	`  for (int i = 0; i < N; i++) {`
	`  for (int j = 0; j < N; j++) {`
	`  v.add(i);`
	`  }`
	`  }`
	`}`
		O( )

(b)	`void functionB(int N) {`
	`  while (N > 0) {`
	`  N /= 3;`
	`  }`
	`  cout << N << endl;`
	`}`
		O( )

(c)	`void functionC(int N) {`
	`  Vector<int> v(N);`
	`  helper(v);`
	`}`
	
	`void helper(Vector<int> v) {`
	`  cout << v[0] << endl;`
	`}`
		O( )

(scratch space–problem 4 continues on next page)

(d)	`// Assume that str has nonzero length and starts out containing`
	`// no 'X' characters.`
	`void functionD(string& str) {`
	`  int N = str.length();`
	`  while (str[N - 1] != 'X') {`
	`  for (int i = 0; i < N; i++) {`
	`  if (str[i] != 'X') {`
	`  str[i] = 'X';`
	`  break;`
	`  }`
	`  }`
	`  }`
	`}`
		O( )

(e)	`// Assume that str has nonzero length and starts out containing`
	`// no 'X' characters.`
	`void functionE(string& str) {`
	`  int N = str.length();`
	`  if (str[N - 1] == 'X') {`
	`  return;`
	`  }`
	`  for (int i = 0; i < N; i++) {`
	`  if (str[i] != 'X') {`
	`  str[i] = 'X';`
	`  break;`
	`  }`
	`  }`
	`  functionE(str);`
	`}`
		O( )

(f)	Analyze the `collectionMystery()` function in Problem 1 (Stacks and Queues).	O( )

(scratch space–end of problem 4)

SUID:

(scratch space–end of exam)