

CS106B Practice Midterm 1 (previously: Winter 2026)

Name:	SUID (numeric):
-------	-----------------

Instructions:

- **Stay within designated answer boxes**, or graders may not see your response on Gradescope. If you require extra space, write “continued” in the answer box, and then use designated scratch space somewhere in the exam. No additional scratch paper.
- **The time limit is strict.** When time is called, immediately drop your pencil/pen, and proceed to turn in your exam. No writing.
- For coding problems, **you may assume #include and function prototypes** already exist for you as needed.
- **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.
- **Permitted and unpermitted aid:** You may have one double-sided, 8.5” × 11” page of notes. No other aids (laptops, cell phones, etc). No electronics including smartwatches and earbuds within reach or sight. Other than course staff, no consulting with other humans or AI in any way during the exam period.
- **Proctoring:** Proctors will be present in the exam room. If you have any questions or concerns during this exam, please speak with a proctor. At some point during the exam, proctors may take a few broad photos of the room, for later seating chart verification. Backpacks and phones must be left at the front of the room or with proctors. Proctors will verify your photo ID when you turn in your exam.
- If you need a **break, (e.g., bathroom)**, give your exam to a proctor, who will log your break.

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:

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 3 (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:

<i>Front</i>	{		}	<i>Back</i>
--------------	---	--	---	-------------

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

<i>Bottom</i>	{		}	<i>Top</i>
---------------	---	--	---	------------

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

<i>Front</i>	{		}	<i>Back</i>
--------------	---	--	---	-------------

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

{		}
---	--	---

(scratch space—end of problem 1)

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) {
```

(problem 2 continues on next page)

- (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) {
```

```
}
```

(end of problem 2)

3. Big-O (10 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)	<pre>void functionA(int N) { Vector<int> v; for (int i = 0; i < N; i++) { for (int j = 0; j < N; j++) { v.add(i); } } }</pre>	$O(\quad)$
(b)	<pre>void functionB(int N) { while (N > 0) { N /= 3; } cout << N << endl; }</pre>	$O(\quad)$
(c)	<pre>void functionC(Vector<int>& v) { int N = v.size(); helper(v); } void helper(Vector<int> v) { cout << v[0] << endl; }</pre>	$O(\quad)$
(d)	<pre>// 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; } } } }</pre>	$O(\quad)$
(e)	Analyze the <code>collectionMystery()</code> function in Problem 1 (Stacks and Queues).	$O(\quad)$

(scratch space—end of exam)