CS101A(H) Homework 4

Brief Overview

This homework consists of four parts:

- **Single Choices (10 pts):** *Scoring per question:*
  - Select the single correct option to receive full credit.
  - Any wrong selection, selecting multiple options, or leaving it blank receives **0 pt**.
- **Multiple Choices (15 pts):** *Scoring per question:*
  - Selecting *all and only* the correct options: **full credit**.
  - Selecting a *proper subset* of the correct options (and *no* incorrect options): **half credit**.
  - Selecting *any* incorrect option or leaving it blank: **0 pt**.

  Note: If the correct set has only one option, the "half credit for a proper subset" case does not apply.
- **Others (4+3+7+7+9 = 30 pts): Answers must be written in English**. The answer format varies by problem (e.g., short proofs, derivations, explanations, code snippets). The point value for each subproblem is printed next to the problem title. Follow the instructions given in each problem; show key steps when requested.
- **Bonus (2 pts):** Some questions are marked "**bonus**". If answered correctly, you will earn the indicated extra points.

Your grade for this homework will be computed as

$$\min\Big(\text{Single Choices} + \text{Multiple Choices} + \text{Others},\ 50\text{ pts}\Big) + \text{Bonus}.$$

Notes: Unless otherwise stated, always express your final asymptotic bounds using $\Theta(\cdot)$ notation for tight complexity, not just $O(\cdot)$ or $\Omega(\cdot)$. All answers must be written **inside the provided answer boxes** and **in English**. When submitting, match your solutions to the problems correctly in Gradescope. No late submission will be accepted. Failure to follow these rule may result in partial or full loss of credit.

Usage of AI. The use of AI tools for searching information or obtaining assistance on this homework is strictly prohibited. All solutions must be derived from your own understanding and effort. Submissions will be reviewed carefully, and any indication of reliance on AI-generated content will result in severe penalties.

Academic Integrity. This course upholds the highest standards of academic integrity. Any form of academic dishonesty, including plagiarism, unauthorized collaboration, or the use of prohibited resources such as AI tools, will be treated as a serious violation. Such actions not only undermine your own learning but also violate university policies.

Due Date: Oct 28, 2025, **8:00 PM**

1 Single Choices

1. Which of the following statements about arrays and linked lists is *correct*?

A. In a **singly linked list**, accessing the k-th element takes $\Theta(1)$ time.

B. Inserting an element into the *middle* of an **array** while preserving order takes $\Theta(1)$ time.

C. Given only a pointer to a node in a **singly linked list**, we can *always* delete that node in $\Theta(1)$ time without changing any other node's contents.

D. Given a pointer to some node in a **non-circular doubly linked list**, we can reach every node by following `prev`/`next` in both directions.

E. **Arrays** and **singly linked lists** have the same asymptotic time for random access (`A[k]`).

2. Using a stack, evaluate the following Reverse Polish Notation (RPN) expression. What is the final result?

```
8 3 4 + 2 ^ * 5 1 2 + 3 ^ - 6 2 3 ^ % + +
```

A. 360 B. 376 C. 392 D. -16 E. 400

3. Queue-as-array with dynamic resizing

A queue uses an array `Q[N]` with initial $N = 8$ and the *one-slot-empty* rule:

$$\texttt{empty} \iff \texttt{front} = \texttt{rear}, \qquad \texttt{full} \iff (\texttt{rear} + 1) \bmod N = \texttt{front}.$$

Resize: at any time when the queue is full, allocate size $2N$, copy the logical order so that the old front becomes `Q'`[0], then set `front` $= 0$, `rear` $=$ (old size), $N \leftarrow 2N$, finally place the new element at `Q'`[`rear`] and advance `rear`.

Initially:

$$\texttt{front} = 6, \quad \texttt{rear} = 1, \quad N = 8,$$

and from front to `rear` $- 1$ the content is

$$A\ (Q[6]),\ B\ (Q[7]),\ C\ (Q[0]).$$

Do in order (assume no underflow):

$$\texttt{push}(D),\ \texttt{push}(E),\ \texttt{pop}(),\ \texttt{push}(F),\ \texttt{push}(G),\ \texttt{push}(H),\ \texttt{push}(I),\ \texttt{pop}(),\ \texttt{push}(J),\ \texttt{push}(K).$$

What is the final state (from front to `rear` $- 1$) and indices?

A. `front` $= 1$, `rear` $= 10$, $N = 16$; content: C, D, E, F, G, H, I, J, K.

B. `front` $= 8$, `rear` $= 3$, $N = 16$; content: I, J, K.

C. `front` $= 7$, `rear` $= 2$, $N = 16$; content: B, C, D, E, F, G, H.

D. `front` $= 0$, `rear` $= 9$, $N = 16$; content: C, D, E, F, G, H, I, J, K.

4. Hash Table.

Open addressing with *linear probing*, table size $m = 13$, hash $h(k) = k \bmod 13$. Deleted cells are marked `ERASED`; an `insert` probes $h, h{+}1, \ldots \pmod m$ and places the key at the first `ERASED` or `EMPTY` it encounters.

Starting from an empty table, perform:

$$\texttt{insert}\ 18, 31, 44, 57, 70, 83, 24, 37, 13; \quad \texttt{erase}\ 44, 83; \quad \texttt{insert}\ 96.$$

After these operations, the table is:

index	0	1	2	3	4	5	6	7	8	9	10	11	12
content	13	EMPTY	EMPTY	EMPTY	EMPTY	18	31	96	57	70	ERASED	24	37

If we now execute `insert`(5), at which **index** will 5 be placed?

A. 0 B. 1 C. 5 D. 10 E. 12

5. Which of the following is correct?

A. $\log n = o\left(n^{0.0001}\right)$.
B. $n \log n = o(n)$.
C. $2^{\sqrt{n}} = o\left(n^{10}\right)$.
D. $n^{\log \log n} = o\left((\log n)^{\log n}\right)$.
E. $n! = o(2^n)$.

2 Multiple Choices

1. Stack and Queue. Which of the following statements are correct?

A. A stack (LIFO) can be used to *reverse* the order of a sequence by pushing all items and then popping them.
B. A queue (FIFO) can be used to reverse the order of a sequence by pushing all items and then popping them.
C. Using two stacks, we can implement a queue with `push` and `pop` operations whose *amortized* time is $\Theta(1)$ per operation.
D. Using two queues, we can implement a stack so that each `push` is $\Theta(1)$ worst-case, but `pop` may be $\Theta(n)$ in the worst case.
E. For both stacks and queues, accessing the k-th stored element is $\Theta(1)$ time.

2. Hash Table

Open addressing with *double hashing*, table size $M = 17$. Primary hash $h_1(k) = k \bmod 17$, secondary step $h_2(k) = 1 + (k \bmod 16)$.

Insertion/search policy: probe $i_j = (h_1(k) + j \cdot h_2(k)) \bmod M$ for $j = 0, 1, \ldots$. Use *lazy erasing*: bins may be marked `ERASED`; when **searching**, treat `ERASED` as *occupied* and continue and stop when it meets `EMPTY` or the matching key; when **inserting**, treat `ERASED` as *unoccupied* (permitting placement).

After some operations, the table snapshot (index → content) is:

index	0	1	2	3	4	5	6	7	8
content	EMPTY	18	EMPTY	EMPTY	EMPTY	ERASED	52	ERASED	86
index	9	10	11	12	13	14	15	16	
content	103	120	137	EMPTY	EMPTY	EMPTY	EMPTY	EMPTY	

Which of the following are *correct* under this snapshot and policy?

A. Key 137 must have been inserted *after* key 18.

B. It's possible that exactly one erase operation produced the two `ERASED` bins at indices 5 and 7.

C. If we now insert a key y with $h_1(y) = 11$ and $h_2(y) = 1$, then y will be placed at index 12.

D. There exists a key $x \equiv 11 \pmod{17}$ such that `search`(x) inspects exactly 5 bins on this snapshot.

E. It is possible that key 52 was the very first key inserted into an initially empty table.

3. Which of the options for $T(n)$ share the same Θ-asymptotic solution?
Assume $T(0) = T(1) = 1$.

A. $T(n) = 2\,T(n/2) + \Theta(n)$

B. $T(n) = T(n-1) + n$

C. $T(n) = 3\,T(n/3) + n$

D. $T(n) = T(n/2) + \Theta(n)$

E. $T(n) = 4\,T(n/2) + \Theta(n^2)$

4. Asymptotics Analysis
Assume `op()` runs in constant time. Consider the following two procedures; their running times (as functions of input size n) are $F(n)$ and $G(n)$, respectively.

```
void AlgoF(int n){
  op();
  if (n % 2 == 0) {
    int r = ceil(sqrt(n));
    for (int i = 0; i < n; ++i)
      for (int j = 0; j < r; ++j)
        op();
  }
}

void AlgoG(int n){
  int L = floor(log2(max(n,1)));
  for (int i = 0; i < n; ++i)
    for (int t = 0; t < L; ++t)
      op();
}
```

Which of the following statements are true about $F(n)$ and $G(n)$?

A. $F(n) = o(n^2)$.

B. $F(n) = \Omega(n)$.

C. $F(n) + G(n) = \Theta\big(n^{1.5}\big)$.

D. $F(n) + G(n) = \omega\big(n \log(1.5n)\big)$.

5. Considering a hash table using open addressing, which of the following statements are correct?

A. If the table size m is prime. The hash function is $h_1(k) = k \bmod m$, and the step function is $h_2(k) = 1 + (k \bmod (m-1))$. The probe sequence is $i_j = (h_1(k) + j \cdot h_2(k)) \bmod m$ for $j = 0, 1, \ldots, m-1$. Then for any key, the probe sequence will visit every slot, unless it terminates early on `EMPTY`.

B. If the table size m is prime. The hash function is $h_1(k) = k \bmod m$, and the probe sequence is $i_j = (h_1(k) + \frac{1}{2}(j + j^2)) \bmod m$ for $j = 0, 1, \ldots, m-1$. Then for any key, the probe sequence will visit every slot, unless it terminates early on `EMPTY`.

C. In linear probing, as an implementation of erasing an element, moving all successive elements forward one slot until meets an empty slot preserves the correctness of future searches; however, large load factor can degrade performance unless adaptively growth of hash table size is performed.

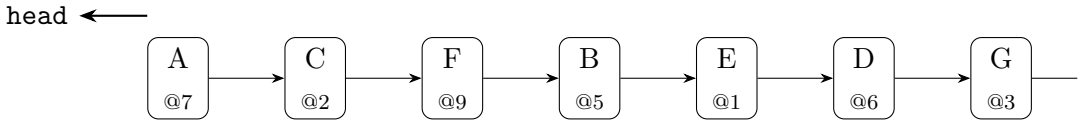
D. In quadratic probing, as an implementation of erasing an element, using a special `ERASED` marker (distinct from `EMPTY`) preserves the correctness of future searches; however, excessive `ERASED` lables can degrade performance unless periodic rehashing is performed.

3 Array Representation of a Singly Linked List (4 pts)

We store a singly linked list in two arrays `value[]` and `next[]`. At index i, the pair (`value`$[i]$, `next`$[i]$) represents one node; `next`$[i] = -1$ means tail. Let `head` be the index of the first node.

(a) From list (diagram) to arrays (fill numbers). (2 pts)

The diagram shows the logical order of nodes. Each box also shows the array index where the node is stored. Fill `head` and the entries of `next[]` below. Use -1 for the tail. For indices with *EMPTY* in `value[]`, write `-` in `next`.



Array capacity is 10 with indices $0 \sim 9$. The `value[]` array is given; fill `head` and `next[]`:

index	0	1	2	3	4	5	6	7	8	9
`value`	*EMPTY*	*E*	*C*	*G*	*EMPTY*	*B*	*D*	*A*	*EMPTY*	*F*
`next` (fill)	-				-				-	

`head` = ☐

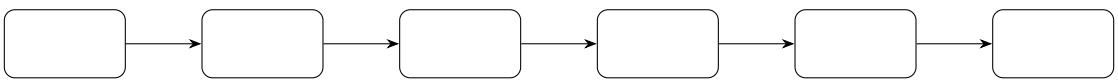
(b) From arrays to list (draw the picture). (2 pts)

Consider another configuration (capacity 10):

index	0	1	2	3	4	5	6	7	8	9
`value`	*A*	*EMPTY*	*C*	*EMPTY*	*E*	*B*	*EMPTY*	*G*	*EMPTY*	*D*
`next`	9	-	0	-	-1	4	-	2	-	5

`head` = 7

(i) Starting from `head`, draw the logical linked list (values in order with arrows). (ii) Write the sequence of visited indices.



4 Vocabulary List (3 pts)

We use a singly linked list:

```
struct Node {
  std::string word;
  Node* next;
  Node(const std::string& w, Node* n=nullptr) : word(w), next(n) {}
};
```

All lists are non-empty. No extra containers; only pointer rewiring. Return the new head where applicable.

Move the two nodes after p to the front, preserving order. (3 pts)

Complete `move_two_to_first(Node&* head, Node* p)`. You may assume `p->next` and `p->next->next` exist.

```
void move_two_to_first(Node&* head, Node* p) {
  Node* x = /* (1) */;
  Node* y = /* (2) */;
  p->next  = /* (3) */;
  /* (4) */;
  /* (5) */;
}
```

Fill:

(1) _____ (2) _____

(3) _____ (4) _____

(5) _____

5 Queues Implemented with Stacks (7 pts)

We have already studied queues and stacks in lecture. A beautiful aspect of algorithms is that one data structure can simulate another. In particular, a queue can be implemented using two stacks, preserving the FIFO behavior while using only LIFO operations. Implement a queue using two stacks as in lecture (Queue ADT with `push`/`pop`; Stack ADT with `push`/`pop`/`top`).

(a) Complete the class with minimal blanks. (4 pts)

Fill the blanks `/*(1)*/`–`/*(4)*/`. Keep the style minimal and consistent with the slides.

```
#include <stack>
#include <stdexcept>

struct underflow {};  // as in slides

template <typename Type>
class TwoStackQueue {
private:
```

```
    std::stack<Type> inS, outS;
    long long pop_count = 0;

    void transfer_if_needed() {
        if (/*(1)*/) {
            while (/*(2)*/) {
                Type v = /*(3)*/;
                ++pop_count;
                outS.push(v);
            }
        }
    }

public:
    bool empty() const { return inS.empty() && outS.empty(); }

    void push(Type const &obj) {
        inS.push(obj);
        transfer_if_needed();
    }

    Type pop() {
        if (empty()) { throw underflow(); }
        Type v = /*(4)*/;
        ++pop_count;
        transfer_if_needed();
        return v;
    }

    long long pops() const { return pop_count; }
};
```

Fill:

(1) [ ] (2) [ ]

(3) [ ] (4) [ ]

(b) Worst-case and amortized bounds. (2 pts)

For `push` in `TwoStackQueue`, the worst-case time is [ ] and the amortized time is [ ]. For `pop` in `TwoStackQueue`, the worst-case time is [ ] and the amortized time is [ ].

(c) Pop-count for a given sequence. (1 pt)

Starting from an empty queue, consider the operation sequence below (use your class from part

(a)). Count the total number of internal *stack* pops (both stacks combined), namely the value of `pop_count`. Write only the final number.

Operation sequence.

1. `push(1)`, `push(2)`, `push(3)`, `push(4)`, `push(5)`, `push(6)`, `push(7)`, `push(8)`.
2. `pop()`, `pop()`, `pop()`.
3. `push(9)`, `push(10)`, `push(11)`, `push(12)`.
4. `pop()`, `pop()`, `pop()`, `pop()`, `pop()`.
5. `push(13)`, `push(14)`, `push(15)`, `push(16)`.
6. `pop()`, `pop()`, `pop()`, `pop()`, `pop()`, `pop()`.

Answer: ☐

6 Hash Table (7 pts)

Open addressing with **linear probing**. Table size $m = 11$. Hash function $h(k) = k \bmod 11$. Lazy deletion is used: a deleted cell is marked `ERASED`; `find` keeps probing past `ERASED`, and `insert` may reuse the *first* `ERASED` encountered on its probe.

(a) Insert only (2 pts)

Starting from an empty table, perform in order:

`insert` 18, 44, 27, 59, 32, 31, 73.

Fill the table after all inserts (write the key, or `EMPTY`). Use wrap-around when probing.

index	0	1	2	3	4	5
cell						

index	6	7	8	9	10
cell					

(b) Apply deletions (2 pts)

From your table in (a), delete the keys 44 and 31. Mark their slots as `ERASED` (do not move other keys). Fill the table.

index	0	1	2	3	4	5
cell						

index	6	7	8	9	10
cell					

(c) Insert with lazy deletion and one `find` (3 pts)

Continuing from (b), perform:

`insert` 24, `insert` 35.

Remember: on `insert`, reuse the *first* `ERASED` encountered along the probe. Then:

1. Fill the final table (after the two inserts).
2. Give the exact number of inspected slots when calling `find`(35) on the final table (count each slot read during the probe until the key is found or an `EMPTY` is seen).

index	0	1	2	3	4	5
cell						

index	6	7	8	9	10
cell					

Inspections for `find(35)`:

7 Asymptotics Analysis (9 pts)

Unless otherwise stated, assume all functions are nonnegative for sufficiently large n. You may use standard limit rules, L'Hospital's rule when applicable, and elementary series facts. Do not invoke Master Theorem.

(a) Prove $n^3 = O(n^4)$ using a limit-based argument. Show the key limit and state the conclusion clearly. **(1 pts)**

(b) Find an $f(n), g(n) \geq 0$ such that $f(n) = O(g(n))$ yet $\lim_{n\to\infty} \frac{f(n)}{g(n)} \neq 0$. Provide a concrete pair example and a one-line justification. What's the case when $f(n), g(n) \geq 0$ such that $f(n) = O(g(n))$ yet $\lim_{n\to\infty} \frac{f(n)}{g(n)}$ doen't exist? Also provide a concrete pair example and a one-line justification. **(2 pts)** *Hint: think about oscillating functions!*

(c) Order the functions $f_1, \ldots, f_9$ from smaller to larger asymptotic growth so that if f_i appears before f_j, then $f_i = O(f_j)$. Write only the order as a list like $f_8, f_9, \ldots$. No justification is required. **(2 pts)**

$$f_1(n) = 3^n, \quad f_2(n) = n^{1/3}, \quad f_3(n) = 12, \quad f_4(n) = 2^{\log_2 n},$$
$$f_5(n) = \sqrt{n}, \quad f_6(n) = 2^n, \quad f_7(n) = \log_2 n, \quad f_8(n) = 2^{\sqrt{n}}, \quad f_9(n) = n^3.$$

(d) For each pair, indicate O, Ω, or Θ for $f(n) =?(g(n))$. Show the key limit and state the conclusion clearly. **(2 pts)**

(i) $f(n) = \log_3 n,\ g(n) = \log_4 n$ (ii) $f(n) = n\log(n^4),\ g(n) = n^2\log(n^3)$
(iii) $f(n) = \sqrt{n},\ g(n) = (\log n)^3$ (iv) $f(n) = n + \log n,\ g(n) = n + (\log n)^2$

For the below questions, assume $T(0) = T(1) = 1$ unless otherwise noted. Derive a $\Theta(\cdot)$ bound for $T(n)$. Clearly write down your answers and the process.

(e) $T(n) = 3\,T(n-2) + 5$ **(1 pts)**

(f) $T(n) = 3\,T(n^{1/3}) + \Theta(\log n)$ **(1 pts)**

(bonus) $T(n) = T(n-1) + T(n-2)$ **(2 pts)**