CS101 Algorithms and Data Structures
Fall 2022
Midterm Exam

Instructors: Dengji Zhao, Yuyao Zhang, Xin Liu, Hao Geng

Time: November 2nd 8:15-9:55

INSTRUCTIONS

Please read and follow the following instructions:

- You have 100 minutes to answer the questions.
- You are not allowed to bring any papers, books or electronic devices including regular calculators.
- You are not allowed to discuss or share anything with others during the exam.
- You should write the answer to every problem in the dedicated box **clearly**.
- You should write **your name and your student ID** as indicated on the top of **each page** of the exam sheet.

Name	
Student ID	
Exam Classroom Number	
Seat Number	
All the work on this exam is my own. **(Please copy this and sign)**	

THIS PAGE INTENTIONALLY LEFT BLANK

1. (20 points) True or False

For each of the following statements, please judge whether it is **true(T) or false(F)**. **Write your answers in the following table**.

(a)	(b)	(c)	(d)	(e)	(f)	(g)	(h)	(i)	(j)

(a) (2') Accessing the k-th element in a linked list takes more than constant time, so we cannot apply Quick Sort or Merge Sort to an unsorted linked list.

(a) _____

(b) (2') If $f(n) = \omega(g(n))$, then $\log(f(n)) = \omega(\log(g(n)))$. $f(n)$ and $g(n)$ are both positive-valued.

(b) _____

(c) (2') The choice of hash function does not affect the load factor of a hash table. Assume all candidate hash functions share the same modulo as the table capacity.

(c) _____

(d) (2') The reason why Merge Sort is stable is that both its best-case and worst-case time complexity are $\Theta(n \log n)$.

(d) _____

(e) (2') In Insertion Sort, after m passes through the array, **the first m elements** are sorted in ascending order and these elements are the m smallest elements in the array.

(e) _____

(f) (2') Assume we implement in-place Heap Sort with a min-heap in an array. After popping m times, **the last m elements** in the array are sorted in descending order and these elements are the m smallest elements in the array.

(f) _____

(g) (2') When we Depth-First traverse a tree, a node with smaller depth will be visited before that with larger depth. Breadth-First Traversal has no such property.

(g) _____

(h) (2') After Huffman coding, more frequently used symbols tend to have shorter binary code words and a shorter code word cannot be the prefix of a longer one.

(h) _____

(i) (2') Given a lower bound a, an upper bound b and a binary search tree with n nodes, searching all the elements x in the tree such that $a \leq x \leq b$ takes $O(\log n)$ time.

(i) _____

(j) (2') Building an AVL tree from an arbitrary unbalanced binary search tree with n nodes can be implemented in $O(n)$ time.

(j) _____

2. (15 points) Single Choice

Each question has **exactly one** correct answer. **Write your answers in the following table.**

(a)	(b)	(c)	(d)	(e)

(a) (3') Which of the following statements about **asymptotic bounds** is TRUE?

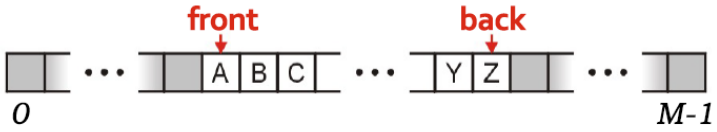
A. $n^2 + \log n = o(n^2 + \log^2 n^2)$

B. $\log(\log n) = \Omega(\sqrt{\log(n^{1/3})})$

C. If $T(n) = T(n/5) + T(7n/10) + 3n$, then $T(n) = O(n)$.

D. If $T(n) = 8T(n/2) + n^3$, then $T(n) = O(n^3)$.

(b) (3') Assume you implement a **queue with a two-ended array** (the naïve version of circular array) of capacity M. You have performed m push operations and n pop operations ($m, n < M$) starting with an empty queue, as shown in the figure below.



A. The index of the front pointer of the queue is m.

B. The index of the back pointer of the queue is m.

C. The number of elements in the queue is $m - n - 1$.

D. You can push $M - m$ more elements onto the queue at most.

(c) (3') Given an array $\langle 1, 5, 4, 3, 2, 6\rangle$, how many passes are needed to sort the array in ascending order with **Flagged Bubble Sort** (the implementation of Bubble Sort that stops immediately when no swaps happen in a pass)?

A. 3 B. 4 C. 5 D. 6

(d) (3') Which of the following statements about **binary trees** is TRUE?

A. Let $\mathcal{F}_k$ be the number of nodes with depth k in a binary tree. Then $0 \le \mathcal{F}_k \le 2k$.

B. Let $\mathcal{G}_h$ be the number of nodes in a binary tree of height h. Then $h+1 \le \mathcal{G}_h \le 2^h + 1$.

C. Let $\mathcal{H}_n$ be the minimum height of a binary tree of n nodes. Then $\mathcal{H}_n = \lfloor \log_2 n \rfloor$.

D. If a binary tree of n nodes has the minimum height $\mathcal{H}_n$, then it is complete.

(e) (3') Which of the following statements about **tree traversal** is TRUE?

A. The in-order traversal of an AVL tree is an ascending sequence.

B. The pre-order traversal of a binary search tree is an ascending sequence.

C. The in-order traversal of a min-heap is a descending sequence.

D. The post-order traversal of a max-heap is a descending sequence.

3. (20 points) Multiple Choices

Each question has **one or more** correct answer(s). Select all the correct answer(s). For each question, you will get 0 points if you select one or more wrong answers, but you will get 2.5 points if you select a non-empty subset of the correct answers. **Write your answers in the following table.**

(a)	(b)	(c)	(d)

(a) (5') Which of the following statements about **List ADT** is/are TRUE?

A. If we want to frequently delete the first element from the list, we prefer to implement the list using a linked list.

B. If we want to randomly access the elements in the list, we prefer to implement the list using an array.

C. Merging two sorted lists, where each list has n elements, to a sorted one takes $O(n)$ time, no matter whether the list is implemented using an array or a linked list.

D. Erasing the tail node of the linked list takes $O(1)$ time, whether the linked list is a singly linked list or a doubly linked list.

(b) (5') Given an initially empty **hash table** of capacity 10 and hash function $h(x) = x \bmod 10$. Assume we are inserting $\langle 101, 27, 91, 46, 13, 52, 42\rangle$ to the table in order. Which of the following statements is/are TRUE?

A. If we deal with collisions using chaining, there will be 5 non-empty chains.

B. If we deal with collisions using using linear probing, 2 collisions happen.

C. If we deal with collisions using quadratic probing, insertion might fail even if the table is not full.

D. It is possible for us to avoid collisions when inserting the given numbers by redesigning the hash function.

(c) (5') In the partition procedure of **Quick Sort**, assume the pivot is chosen uniformly at random from the array $\langle A_1, \cdots, A_n\rangle$. You can assume that n is an odd number. Which of the following statements is/are TRUE?

A. The pivot is most likely to be chosen as the median of $\{A_1, \cdots, A_n\}$.

B. The pivot is expected to be the mean of $\{A_1, \cdots, A_n\}$.

C. In this way, the algorithm runs in $\Theta(n \log n)$ time in the worst case.

D. After partitioning into two subarrays, the length of the shorter one is expected to be about $n/4$ as $n \to \infty$.

(d) (5') Assume we are inserting $\langle 1, 2, 3, 4, 5, 6, 7\rangle$ to an initially empty **AVL tree** in a random order. Which of the following insertion orders will not cause any imbalance i.e. the tree keeps AVL balanced after each insertion?

A. $4, 2, 6, 1, 3, 5, 7$

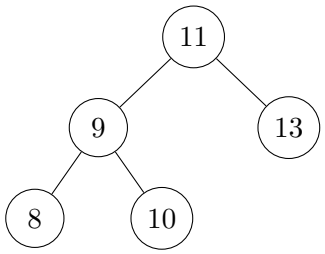
B. $3, 5, 1, 6, 4, 7, 2$

C. $4, 6, 2, 3, 5, 7, 1$

D. $5, 3, 6, 7, 4, 2, 1$

4. (8 points) Play with AVL Tree

Given an AVL tree, as shown in the figure below:

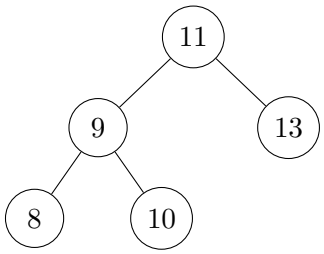


(a) (2') Assume you want to insert 15. Which nodes shall you step through to locate where to insert 15? Please write them down in order.

(b) (3') Based on (a), assume 15 is already inserted and we continue to insert 14. Dose this insertion cause any imbalance in the tree? If so, please draw the tree after rotation and re-balance. If not, explain why.

(c) (3') **This sub-question is independent of (a) and (b).**

For your convenience, the figure is given again below. Is it possible that 11 is the last element inserted to the AVL tree shown in the figure? If so, please draw one possible AVL tree before inserting 11. If not, explain why.



5. (10 points) Insertion Sort on Small Arrays in Merge Sort

Although Merge Sort runs in $\Theta(n \log n)$ worst-case time and Insertion Sort runs in $\Theta\left(n^2\right)$ worst-case time, the constant factors in Insertion Sort often make it faster in practice for small size problems.

Please recall the modified version of Merge Sort, which has been covered in our lectures, where the subarrays of length $\leqslant k$ will be sorted directly using Insertion Sort, instead of continuing dividing it into two halves. Note that there will be approximately n/k subarrays to perform Insertion Sort on.

(a) (2') Show that Insertion Sort can sort the n/k subarrays of length k in $\Theta(nk)$ worst-case time.

(b) (4') Show that it takes $\Theta(n \log(n/k))$ time to merge the n/k sorted subarrays into one.

(c) (4') From the analysis above, we conclude that this algorithm has $\Theta(nk + n \log(n/k))$ time compexltiy. In practice, we may observe that the algorithm takes $T(n) = ank + bn \ln(n/k)$ time, where $a, b > 0$ are some constants. What value of k (in terms of a and b) will you choose to minimize $T(n)$? Justify your answer. It is OK if your answer is not necessarily an integer.

6. (12 points) Count β-Inversions

Given an array $A = \langle A_1, \cdots, A_n \rangle$, a pair of elements (A_i, A_j) form **a β-inversion** ($\beta > 0$) if

$$i < j \text{ and } A_i > \beta \cdot A_j$$

For example, a 1-inversion is the inversion introduced in our lectures when $\beta = 1$.

Note: For the following sub-questions, you should describe your algorithms in **natural language** or **pseudocode** clearly.

(a) (5') Recall that we have studied an enhanced Merge Sort to count the number of inversions in our lectures. When merging two sorted subarrays L and R, we also count the number of inversions (l, r) such that $l \in L$ and $r \in R$.

Now, please come up with a similar $\Theta(n)$ algorithm to **count the number of β-inversions** (A_i, A_j) **such that** $A_i \in L$ **and** $A_j \in R$, where $L = \langle A_1, \cdots, A_{mid} \rangle$ and $R = \langle A_{mid+1}, \cdots, A_n \rangle$ are two **sorted subarrays** of A.

Hint: You could come up with a $\Theta(n^2)$ algorithm first, and then optimize the inner loop.

(b) (7') Based on (a), please come up with a **divide-and-conquer** algorithm to count the number of β-inversions in A. Analyse the time complexity of your algorithm.

Note:

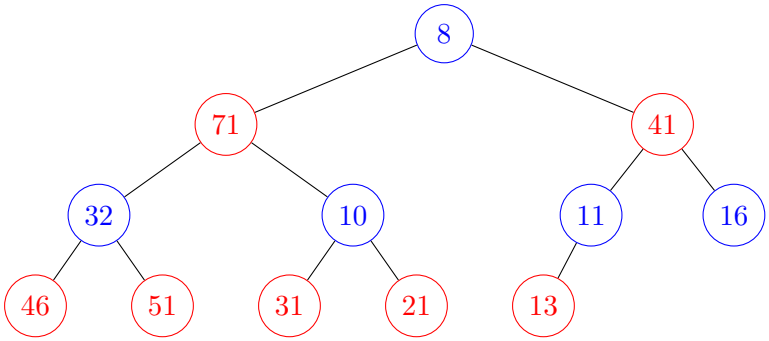
- You can directly use the subroutine of merging two sorted sub-arrays into one (MERGE(L, R)) and the subroutine you implemented in (a) (COUNT-β-INVERSIONS-L-R(L, R)).
- Please describe how you **divide** a problem into sub-problems and **combine** the solutions to the subproblems clearly and remember to show the **base case**.
- Your algorithm should be better than $\Theta(n^2)$.

7. (15 points) Min-Max-Heap

A *min-max-heap* is a data structure which supports both `popMin()` and `popMax()` operations in $O(\log n)$ time. A min-max-heap is still a complete binary tree, but it has the following properties:

- For every element at even depth (on min-level), it is smaller than its parent (if existing) and greater than its grandparent (the parent node of its parent node, if existing). In short: **`x<parent(x)` and `x>grandparent(x)` for `x` on min-level**.
- For every element at odd depth (on max-level), it is greater than its parent and smaller than its grandparent(if existing). In short: **`x>parent(x)` and `x<grandparent(x)` for `x` on max-level**.

You can assume that all nodes are distinct and there are always more than one nodes in this question. Please see the following example min-max-heap, as shown in the figure below:



(a) (4') **Let's See What's New about Min-Max-Heap**

For each of the following statements about min-max-heap, please judge whether it is true or false. Tick ($\surd$) the correct answer. Assume the parent and the children of the node `x` all exist.

i. If `x` is on min-level, then its parent and children are also on min-level. ◯ True ◯ False

ii. If `x` is on max-level, then `x` is greater than both of its children. ◯ True ◯ False

iii. If `x` is on min-level and `x` is a strict ancestor of `y`, then `x<y`. ◯ True ◯ False

iv. The minimum is on min-level and the maximum is on max-level. ◯ True ◯ False

(b) (3') **Where is Min and Max?**

How do we search the minimum and the maximum element in a min-max-heap with n elements? What is their time complexity?

(c) **Percolate it Up!**

Similar to a common heap we learned in the lectures, the new element is inserted at the first available leaf and we will maintain the heap property along the path from this leaf node to the root node, which is called a **percolation-up**. For your reference, the pseudocode of **percolation-up in a common heap** is given below:

Algorithm 4 Percolation-Up in a Common Heap

```
function PERCOLATE-UP(x)                          ▷ Assume x is inserted at some leaf
    while parent(x) exists and x<parent(x) do     ▷ Is it a min-heap or a max-heap?
        Swap x and parent(x)
    end while
end function
```

However, there are more properties to be considered in a min-max-heap. The pseudocode (with some incomplete lines) of **percolation-up in a min-max-heap** is given below:

Algorithm 5 Percolation-Up in a Min-Max-Heap

```
function PERCOLATE-UP(x)                          ▷ Assume x is inserted at some leaf and parent(x) exists
    if x is on max-level then
        if _____(1)_____ then                     ▷ If the property between two adjacent levels is not broken
            while grandparent(x) exists and _____(2)_____ do     ▷ Similar to a max-heap
                Swap x and grandparent(x)
            end while
        else                                      ▷ Else the property is broken
            Swap x and parent(x)                  ▷ x is now on min-level
            while grandparent(x) exists and _____(3)_____ do     ▷ Similar to a min-heap
                Swap x and grandparent(x)
            end while
        end if
    else                                          ▷ Else x is on min-level. You don't need to finish this case
        ...
    end if
end function
```

i. (2') Choose one of the following statements to fill in the blank (1):

◯ `x<parent(x)` ◯ `x<grandparent(x)` ◯ `x>parent(x)` ◯ `x>grandparent(x)`

ii. (1') Fill in the blank (2): _____

iii. (1') Fill in the blank (3): _____

(d) (4') **Percolate it Down?**

Similarly, when popping a node off the min-max-heap, we move the last leaf node to the *hole* and percolate it down. However, the **percolation-down** procedure in a min-max-heap is a bit more complicated.

In this question, just consider you are popping the minimum off the example min-max-heap shown above (it is shown again below for your convenience). Please draw the min-max-heap after this `popMin()`.