CS 101 *Data Structure and*
Fall 2021 *Algorithm* *Midterm Exam*

INSTRUCTOR: Dengji Zhao, Yuyao Zhang, Zhihao Jiang

TIME: Nov 10th 8:00-10:00

INSTRUCTIONS

- You have 2 hours.
- 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 of every problem in the dedicated box.
- You should write **your name and your student ID** as indicated on the top of each page of the exam sheet.
- You should write your answers **clearly**.

Name	
Student ID	
ShanghaiTech Email(Before @)	
Room 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

Note that you should write you answers of this section in the table below.

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

(a) **(2 pt)** Double-linked list requires less time complexity than single-linked list on finding the kth element in average.

(b) **(2 pt)** For a single linked array, erasing the element after the current pointer takes $O(1)$, and erasing the element pointed by the current pointer also takes $O(1)$.

(c) **(2 pt)** Bubble sort and insertion sort require only $O(1)$ additional space, whereas selection sort, Merge sort and Quicksort require at least $\Omega(\log n)$ additional space.

(d) **(2 pt)** Among all the sorting algorithms we have learned, Quicksort will take the smallest amount of time when all elements of input array are identical.

(e) **(2 pt)** Only given the depth of all leaf nodes in a tree, we can infer the height of this tree.

(f) **(2 pt)** Mergesort has a worst-case runtime of $\Theta(N \log N)$, where N is the number of elements in the list that we're sorting.

(g) **(2 pt)** If we choose the pivot to be a randomly chosen element of each subsequence then QuickSort is $O(n \log n)$ in the worst case.

(h) **(2 pt)** There are 133 distinct shapes of binary trees with 6 nodes.

(i) **(2 pt)** The main idea of Huffman Coding is to compress input data by using fewer bits to encode more frequently occurring characters.

(j) **(2 pt)** Suppose there are n nodes in an AVL tree, its minimum possible height is given by $\lceil \log_2(n+1) \rceil - 1$.

2. (15 points) Single Choice (Choose the ONLY ONE correct answer)

Note that you should write you answers of this section in the table below.

(a)	(b)	(c)	(d)	(e)
D	C	B	B	A

(a) (3 pt) Which of the following options is NOT correct?

(A) $\log(n!) = \Theta(\log(n^n))$
(B) $\sqrt{n} = \Omega((\log n)^2)$
(C) $n\log(n^5) = O(n^2\log(n^2))$
(D) $n + \log n = o(n + (\log n)^2)$

(b) (3 pt) You are given an open-addressing hash table with m slots and we are using linear probing, the probability that the first two slots of the table are filled after the first two insertions is:

(A) $\frac{1}{m^2}$
(B) $\frac{2}{m^2}$
(C) $\frac{3}{m^2}$
(D) $\frac{4}{m^2}$

(c) (3 pt) Consider a situation where swap operation is very costly. Which of the following sorting algorithms should be preferred so that the number of swap operations are minimized in general?

(A) Heap sort
(B) Selection sort
(C) Insertion sort
(D) Bubble sort

(d) (3 pt) What is the purpose of the following pseudocode if the function runs on a tree?

```
1: function FUNCTION(node)
2:     X_node ← 0
3:     for all child ∈ children(node) do
4:         X_child ←FUNCTION(child)
5:         X_node ←MAX(X_node, X_child + 1)
6:     end for
7:     return X_node
8: end function
```

(A) Breadth-first traverse the tree and compute the depth of each node
(B) Depth-first traverse the tree and compute the height of the tree
(C) Depth-first traverse the tree and compute the degree of each node
(D) Depth-first traverse the tree and compute the size of all subtrees with each node as the root

(e) (3 pt) In a binary tree with n nodes, every node has an odd number of descendants. What is the number of nodes in the tree that have exactly one child?

(A) 0
(B) 1
(C) $(n-1)/2$
(D) $n-1$

3. (25 points) Multiple Choices (There will be one or multiple correct answers. Choose ALL the correct answers. You will get half credits if you only choose a subset of the correct answers.)

Note that you should write you answers of this section in the table below.

(a)	(b)	(c)	(d)	(e)
ABCD	AD	ABD	ABD	BD

(a) (5 pt) Which of the following statement(s) are correct? ABCD

(A) Reversing a linked list of length n could be done with extra space complexity $O(n)$.

(B) Reversing an array of length n could be done with extra space complexity $O(1)$.

(C) A stack of any length can be reversed using a certain number of extra stacks.

(D) A queue of any length can be reversed using a certain number of extra stacks.

(b) (5 pt) Given the recurrence $T(n) = aT(n/b) + n^d$, which of the following statement(s) are correct?AD

(A) If the recurrence relation indicates a divide-and-conquer algorithm, a problem of size n will be divided into a subproblems of size n/b to be conquered recursively.

(B) If the recurrence relation indicates a divide-and-conquer algorithm, $\Theta(n^d)$ is the time complexity of finding solutions to all subproblems of the original problem.

(C) $(a, b, d) = (2, 2, 1)$ if it indicates Binary Search algorithm.

(D) $(a, b, d) = (7, 2, 2)$ if it indicates Strassen's Matrix Multiplication algorithm.

(c) (5 pt) Which of the following statement(s) are correct?ABD

(A) The worst case, best case, and average case time complexity for MergeSort is $O(n \log n)$, $\Omega(n \log n)$, $\Theta(n \log n)$, respectively.

(B) The worst case extra space complexity for MergeSort is $O(n)$.

(C) The worst case, best case, and average case time complexity for QuickSort is $O(n \log n)$, $O(n \log n)$, $O(n \log n)$, respectively.

(D) The Mergesort is more efficient than Quicksort if the data to be sorted can only be efficiently accessed sequentially.

(d) (5 pt) There is a binary tree T with n nodes, $n > 0$. Which of the following statements is(are) true about T?ABD

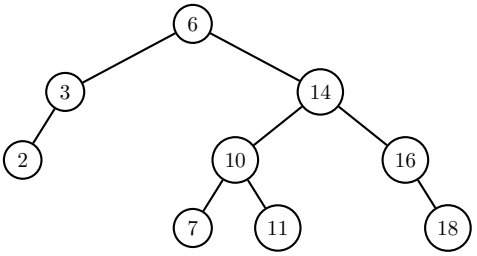
(A) The maximum height of T is $n - 1$ and the minimum height of T is $\lfloor log_2 n \rfloor$.

(B) If $n = 1$, T is a perfect binary tree.

(C) If the number of internal nodes having one child is 5, and the number of internal nodes having two children is 10. The number of leaf nodes in the binary tree T is 10.

(D) If T was transformed from a general tree T' through **Knuth transform**, which means T is a left-child right-sibling binary tree. Then the post-order traversal of T' identical to the in-order traversal of T.

(e) (5 pt) You are given an AVL tree as blow. Suppose we promote the minimum element in the right sub-tree when erase a node with 2 children. Which of the following operation sequences will cause 2 imbalances that must be corrected in total in order to rebalance the tree? B D



(A) Erase 2 6

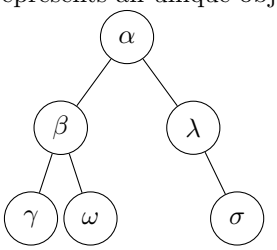
(B) Insert 4 5 12 Erase 2

(C) Erase 6 2 3 Insert 20

(D) Insert 1 0 4 13 19

4. (10 points) Magic AVL

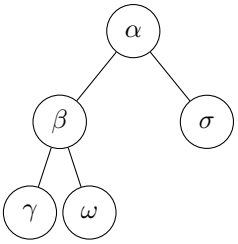
Consider the AVL tree below. Each symbol represents an unique object stored in the AVLtree.

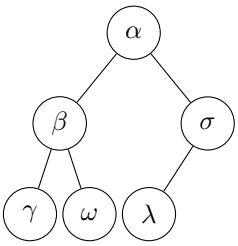


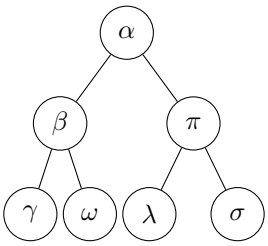
The following 4 questions are **independent** of each other, i.e. for each question, your answer should be build on the original AVL Tree above, instead of the AVL Tree from the answer of the last question.

(a) (3 pt) If we delete object λ, then insert object λ, then insert new object π ($\lambda < \pi < \sigma$), please draw the AVL tree after each insert/delete operation.

Ans:

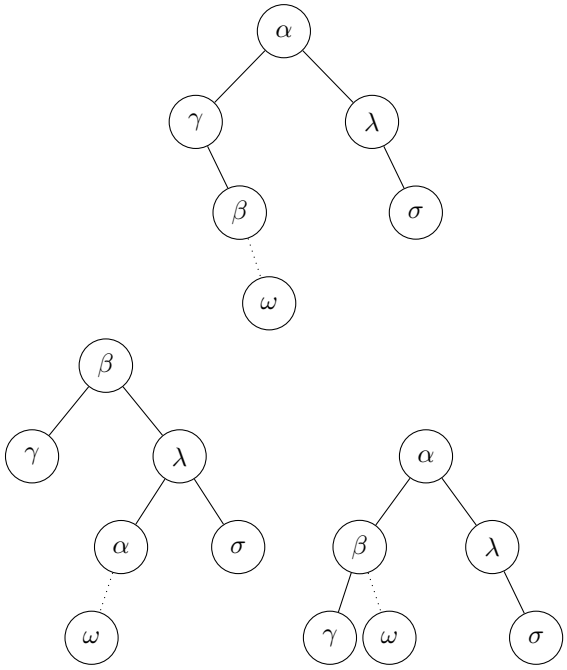






(b) (3 pt) If we want to insert a new object π(not equal to the 6 objects)but we don't want to change the tree's current structure to maintain balance. What are the ranges of the object we can insert? (Note: the range is denoted by the objects, example: $\alpha < \pi < \beta$,$\pi < \gamma$). Ans: $\pi < \gamma, \gamma < \pi < \beta, \beta < \pi < \omega, \omega < \pi < \alpha, \alpha < \pi < \lambda$

(c) (2 pt) If we know the last object we insert is ω. What is the tree before inserting it? Symbols must be unique. You may only use the 6 printed symbols. If there are more than 3 correct answers, give only 3 correct answers would lead to a full credit.
Ans:



(d) **(2 pt)** If we don't know the last object we insert, but we know we change the tree's structure to maintain balance. What is the object we may insert? Please write down the object and the corresponding tree which is inserted to. Symbols must be unique. You may only use the 6 printed symbols. If there are more than 3 correct answers, give only 3 correct answers would lead to a full credit.
Ans: β γ σ ω

5. (10 points) Heap

We are given the following array representing a min-heap where each letter represents a unique number. Assume the root of the min-heap is at index zero, i.e. A is the root. Note that there is no significance of the alphabetical ordering, i.e. just because B precedes C in the alphabet, we do not know if B is less than or greater than C.

Array: [A, B, C, D, E, F, G]

Six unknown operations are then executed on the min-heap. An operation is either a removeMin or an insert. The resulting state of the min-heap is shown below

Array: [A, G, B, X, E, F, C]

(a) (6 pt) The first and the final operations have already been filled in for you. Fill in the space below using the following options.
Note: You are free to use each of the following options 0 time, once or more than once.

(A) removeMin()
(B) insert(B)
(C) insert(C)
(D) insert(X)

Hint: The number of elements in the heap doesn't change after 6 operations; In the final heap, X's position is in the left sub-heap.

1. removeMin()
2. D insert(X)
3. A removeMin()
4. A removeMin()
5. B insert(B)
6. insert(A)

(b) (4 pt) Fill in the following comparisons with either $>$, $<$, or ? if unknown. We recommend considering which elements were compared to reach the final array.

1. X $\geq$ B
2. X $\geq$ C
3. G ? F
4. G $\geq$ D

6. (10 points) Monotone Matrix

Consider **A** as an $m \times n$ $(m < n)$ matrix without duplicate elements. Let min_i denote the column index of the minimum element in the i-th row of **A**. Then we say matrix **A** is **monotone** if $min_1 < min_2 < \ldots < min_m$. Now given **A** is a monotone matrix, Ge Ziwang wonders how to find the minimum element in each row of A i.e. min_i for all rows. You can assume that indices start from index 1 in a row or a column of the matrix.

(a) (2 pt) Ge Ziwang now obtains $min_{19} = 26$ for the 19-th row of **A**. What is the range of possible values of min_{18} amd min_{20}? Assume $m > 19$ and $n > 26$ in this setting.

$min_{18} \in \{18, 19, \ldots, 25\}$ and $min_{20} \in \{27, 28, \ldots, n - m + 20\}$

(b) (8 pt) According to (a), come up with a divide-and-conquer algorithm to help Ge Ziwang find the minium elements in all rows of **A**. Briefly explain your main idea step by step and analyse its time complexity.

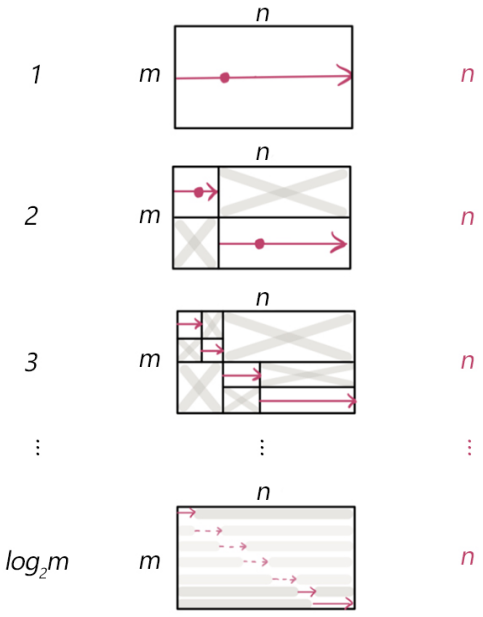
Hint: partition the matrix along the middle row.

Algorithm Design:

1. If the problem is reduced into a row vector$(m = 1)$, then iterate it to find the minimum and return it.
2. Else we cut the matrix rows in the middle and iterate the middle row to find the index of minimum element in this row, denoted by min_i.
3. For the left submatrix with indices $< min_i$, discard the bottom half because of the monotonicity. Similarly, discard the upper half of the right submatrix with indices $> min_i$. Recur for both. (This is equivalent to partition the matrix along the middle row and the min_i-th column, and recur for the upper left and the bottom right submatrices.)

Time Comlexity Analysis:

As the figure shown below, each time we cut the matrix in the middle row, so after $O(\log m)$ levels of partitions we will reach the base case. At each level, since we discard half submatrices so only n itrations are needed to find the minimum of the middle row for all submatrices and each iteration with constant comparision operation takes $O(1)$. Hence the total work at each level is $O(n)$. Therefore, the time complexity of this problem is $T(n) = O(n \log m) = O(n \log n)$.

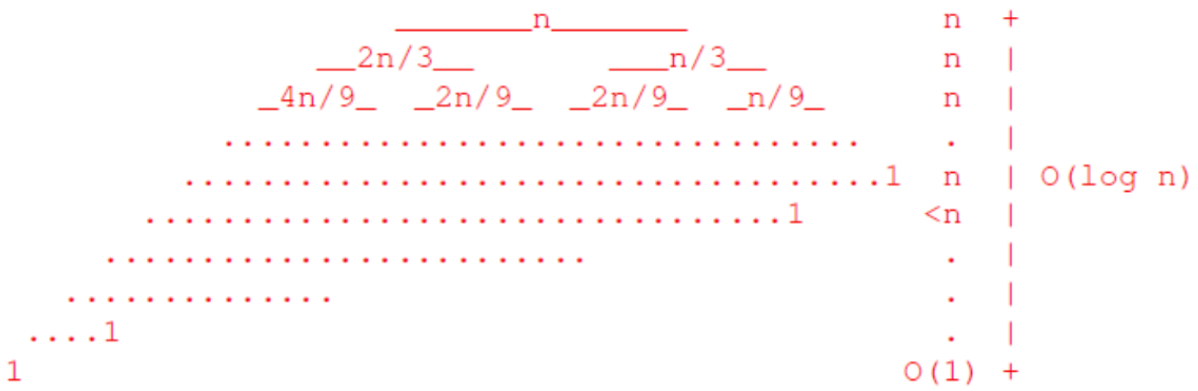


7. (10 points) Inequal Divide

In this problem, we will analyze an alternative to divide step of merge sort. Consider an algorithm `InequalSort()`, identical to `MergeSort()` except that, instead of dividing an array of size n into two arrays of size $\frac{n}{2}$ to recursively sort, we divide into two arrays with inequal size. For simplicity, you may ignore floors and ceilings).

(a) (6 pt) Analysis: If we divide into two arrays with sizes roughly $\frac{n}{3}$ and $\frac{2n}{3}$. Write down a recurrence relation for `InequalSort()`. Assume that merging takes $\Theta(n)$ time. Show that the solution to your recurrence relation is $O(n \log n)$ by drawing out a recursion tree, assuming $T(1) = O(1)$. Note, you need to prove both upper and lower bounds.

$T(n) = T(\frac{n}{3}) + T(\frac{2n}{3}) + \Theta(n)$.



The root does no more than cn work for some positive constant c. Then the second level does no more than c(n/3) + c(2n/3) = cn(1/3 + 2/3) = cn work, the third level no more than c(n/9) + c(2n/9) + c(2n/9) + c(4n/9) = cn(1/3 + 2/3)2 = cn, and so on. In fact, each level requires at most cn work, though may do significantly less work near the bottom of the unbalanced tree. The longest root to leaf path has height $\log_{\frac{3}{2}} n = O(\log n)$, leading to a O(n log n) upper bound. The work is also lower bounded by $\Omega(nlogn)$, because the subtree above and including level $h = \log_3 n$ is a complete binary tree with height h, for which each level does at least c'n for some positive constant c'.

(b) (2 pt) Generalization: If we divide array into two arrays of size $\frac{n}{a}$ and $\frac{(a-1)n}{a}$ for arbitrary constant $a > 1$ recursively, what is the asymptotic runtime of the algorithm? Is there any change on time complexity compared to (a)?

In recursive tree, each level requires at most $cn(1/a + (a-1)/a)^i = O(n)$ work, while the height of the tree is $\log_{\frac{a}{a-1}} n$ which is $O(\log n)$ for any constant $a > 0$.

(c) (2 pt) Limitation: If we divide array into two arrays of size a and $n-a$ for some positive constant integer a recursively, what is the asymptotic runtime of the algorithm? Is there any change on time complexity compared to (a)? Assume that merging still takes $\Theta(n)$ time, $T(a) = O(a)$. It may help to draw a new recursion tree.

Here, the amount of work done at the root (level 0) is cn, while the work done at the next level is ca + c(n - a) = cn. Then the work done at level i + 1 is c(n - ia). Then the total runtime of the algorithm is:

$$cn + \sum_{i=0}^{n/a} c(n - ia) = cn + cn(\frac{n}{a} + 1) - ca\frac{\frac{n}{a}(\frac{n}{a} + 1)}{2} = \Theta(n^2)$$