CS101 Algorithms and Data Structures
Fall 2024
Midterm Exam

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

Time: 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)**	

HINTS

1. Master's Theorem

$$T(n) = aT\left(\frac{n}{b}\right) + \Theta(n^d) \text{ for } a > 0, b > 1, d \geq 0$$

$$T(n) = \begin{cases} \Theta\left(n^d\right) & d > \log_b a \\ \Theta\left(n^d \log n\right) & d = \log_b a \\ \Theta\left(n^{\log_b a}\right) & d < \log_b a \end{cases}$$

2. Inversions:
Given a permutation of n elements $a_0, a_1, \ldots, a_{n-1}$. An inversion is defined as a pair of entries which are reversed. That is, (a_j, a_k) forms an inversion if $j < k$ but $a_j > a_k$.

3. Some Mathematical Formulae

$$\sum_{i=1}^{n} \frac{1}{i} = \Theta(\log n)$$

$$\sum_{i=1}^{\infty} \frac{1}{i^2} = \frac{\pi^2}{6}$$

$$\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$$

$$\text{Binomial Coefficient: } \binom{n}{m} = \frac{n!}{m!(n-m)!}$$

$$\text{Stirling Formula (deformed): } n! = \Theta(n^{n+\frac{1}{2}} e^{-n})$$

$$\lim_{n \to \infty} \left(1 + \frac{c}{n}\right)^n = e^c$$

Name: ID:

1. (30 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 1 points if you select a non-empty subset of the correct answers. Write your answers in the **answer sheet**.

(a) (3') Which of the following implementations do/does not affect the time complexity of any stack/queue operation?

A. When we implement a stack by an array, we put `stack.top()` at the first element of the array.

B. When we implement a stack by a singly linked-list with maintaining tail pointer, we put `stack.top()` at the tail of the linked-list.

C. When we implement a queue by a singly linked-list with maintaining tail pointer, we put `queue.back()` at the head of the linked-list and `queue.front()` at the tail.

D. When we implement a queue by a doubly linked-list with maintaining tail pointer, we put `queue.back()` at the head of the linked-list and `queue.front()` at the tail.

Solution:

A. `stack.push()` and `stack.pop()` should be done on the last element of the array to ensure $\Theta(1)$ complexity.

B.C. `stack.pop()` and `queue.pop()` should be done on the head element of the linked-list to ensure $\Theta(1)$ complexity, instead of the tail element because we cannot get the previous node of the tail element in singly linked-list.

D. Obviously.

(b) (3') For any two functions $f(n)$ and $g(n)$ such that $f(n) > 0, g(n) > 0$ and $g(n) = \Omega(1), g(n) = o(f(n))$, which of the following is/are **TRUE**?

A. $f(n) = \omega(g(n))$

B. $|f(n) \pm g(n)| = \Theta(f(n))$

C. $\log g(n) = o(\log f(n))$

D. $e^{f(n)} = \omega(e^{g(n)})$

Solution:

A. $g(n) = o(f(n)) \iff f(n) = \omega(g(n))$

B. $\lim\limits_{n\to\infty} \frac{f(n)\pm g(n)}{f(n)} = \lim\limits_{n\to\infty} \left(\frac{f(n)}{f(n)} \pm \frac{g(n)}{f(n)}\right) = 1 \pm 0 = 1$

C. Counterexample: $g(n) = n!, f(n) = n^n$, or $g(n) = 2^n, f(n) = 4^n$.

D. $\lim\limits_{n\to+\infty} \frac{e^{f(n)}}{e^{g(n)}} = \lim\limits_{n\to+\infty} e^{f(n)-g(n)} = +\infty$

(c) (3') Which of the following statements is/are **TRUE**?

A. The time complexity of bubble sort (no matter which optimization) is always not lower than insertion sort because it always performs not fewer swaps than insertion sort.

B. The worst-case time complexity of counting the number of swaps of insertion sort on an array must be $\Omega(n^2)$.

C. Insertion sort is more suitable for sorting small arrays compared to quick sort.

D. Quick sort is more suitable for sorting large arrays than merge sort in all cases, especially for distributed data.

Solution:

A. Their number of swaps is always identical. Actually bubble sort is worse because it has more unnecessary comparisons instead of swaps.

B. Enhance Merge Sort is $\Theta(n \log n)$ for counting inversion pairs.

C. Obviously. Although the time complexity of insertion sort is $O(n^2)$, it has a smaller constant.

D. Merge sort is better for distributed data. (Quick sort is faster due to its being local and cache-friendly.)

(d) (3') Consider an array$\{a_i\}$ with $n (n \geq 5)$ **distinct** elements. We want to use randomized quick sort to make it sorted. Denote $\{p_i\}$: the index of the i-th smallest number, i.e. a_{p_i} is the i-th smallest number. Which of the following statements is/are **TRUE**?

A. The probability that a_{p_1} and a_{p_n} are compared during sorting is $\frac{1}{n}$.

B. a_{p_i} and $a_{p_{i+1}}$ will always be compared during sorting. $(i \in \{1, \cdots, n-1\})$

C. If we randomly erased one of the elements (except $a_{p_{i-1}}$ and $a_{p_{i+1}}$) in the array(i.e. each element will be equally likely to be selected). The probability of $a_{p_{i-1}}$ and $a_{p_{i+1}}$ are compared is less than $\frac{2n-3}{3n-6}$, for every $i \in \{2, \cdots, n-1\}$.

D. If we randomly erased one of the elements (except $a_{p_{i-1}}$ and $a_{p_{i+1}}$) in the array(i.e. each element will be equally likely to be selected). The probability of $a_{p_{i-1}}$ and $a_{p_{i+1}}$ are compared is greater than $\frac{2n-3}{3n-6}$, for every $i \in \{2, \cdots, n-1\}$.

Solution:

A. Consider the first partition and pivot choosing, if and only if a_{p_1} and a_{p_n} are chosen they will be compared. So the probability is $\frac{2}{n}$.

Moreover, if you remember $\frac{2}{|j-i|+1}$ you can directly calculate the probability.

B. Always compared. $\frac{2}{|j-i|+1}$ is also applicable.

C.D. If we erase a_{p_i}, then $a_{p_{i-1}}$ and $a_{p_{i+1}}$ will be always compared. If we erase other elements, the comparison probability of $a_{p_{i-1}}$ and $a_{p_{i+1}}$ is $\frac{2}{3}$. So the probability of $a_{p_{i-1}}$ and $a_{p_{i+1}}$ are compared is $\frac{1}{n-2} + \frac{n-3}{n-2}\frac{2}{3} = \frac{2n-3}{3n-6}$.

(e) (3') Which of the following statements about BFS/DFS on a tree is/are **TRUE**?

A. When performing queue implemented BFS while checking a node v, the number of nodes we will push into the queue is $O(\text{depth}(v))$.

B. When performing queue implemented BFS, whenever we look at any two nodes u and v inside the queue, $|\textbf{depth}(u) - \textbf{depth}(v)| < 2$.

C. When performing stack-implemented DFS, if we want to visit the children of node v in a specific order, we should push them into the stack in the reversed order.

D. When performing stack implemented DFS, whenever we look at any two nodes u and v inside the stack, u is neither the ancestor nor the descendant of v.

Solution:

A. It should be $O(\text{degree}(v))$.

B. If any node v of depth d has enqueued, then every node of depth $d-1$ should have enqueued, therefore every node of depth less than $d-2$ should have dequeued.

C. Obviously, because the stack is LIFO.

D. If any node v is in the stack, then all of v's ancestors should have been visited, and all of v's descendants should have not been visited.

(f) (3') Consider a complete binary max-heap with 99 nodes (without duplicated nodes). Which of the following statements is/are **TRUE**?

A. Every node of depth 1 is greater than at least 34 other nodes in the heap.

B. The second largest node's depth must be 1.

C. The second smallest node must be a leaf node.

D. There may be a sub-tree with a height of at least 1 that is also a BST.

Solution:

A. The number of descendants of root's left child is $1+2+4+8+16+32=63$, and the number of descendants of root's right child is $1+2+4+8+16+4=35$.

B. It is either the left child or the right child of the root node.

C.D. In this case, each non-leaf node has two children, both of which are smaller than this non-leaf node.

(g) (3') Which of the following statements about the Huffman Coding is/are **TRUE**?

A. Characters with distinct frequencies must have distinct lengths of Huffman Code.

B. The Huffman Code of one character cannot be a prefix of the Huffman Code of another character.

C. Characters with the same frequency must have the same length of Huffman Code.

D. Given the Huffman Code for each character, if one character has the only shortest length, we can infer that it has the largest frequency among those.

Solution:

A. Counterexample: $(2,3,4,5)$.

B. Characters must be the leaf nodes of the Huffman Coding Tree, one cannot be the ancestor of another.

C. Counterexample: $(2, 2, 2)$.

D. By the fact that Huffman Coding minimizes $\sum$(length $\times$ frequency), if it has the shortest length but not the largest frequency, then $\sum$(length $\times$ frequency) is not minimized, which contradicts.

(h) (3') When performing binary search operations in a BST that stores integers from 1 to 100, which sequences of visited nodes is/are **IMPOSSIBLE**?

A. 50, 90, 73, 60, 80, 65, 69

B. 1, 2, 3, 4, 5, 99, 100

C. 94, 22, 91, 30, 35, 80, 50

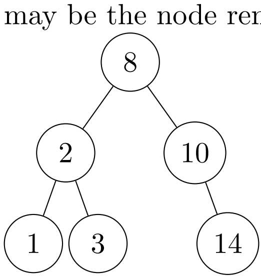
D. 95, 24, 81, 34, 39, 92, 36

Solution:

A. 80 is impossible. It should be between $(50, 73)$.

D. 92 is impossible. It should be between $(34, 81)$.

(i) (3') After erasing a node and making it balanced again, the AVL tree looks like below, what may be the node removed in the original tree?



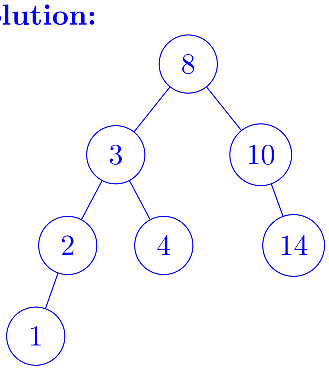
A. 4

B. 9

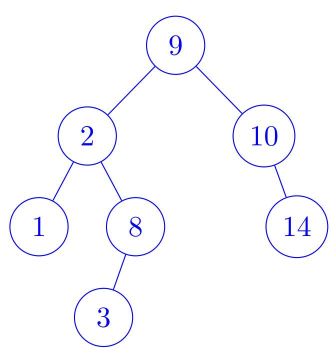
C. 13

D. 20

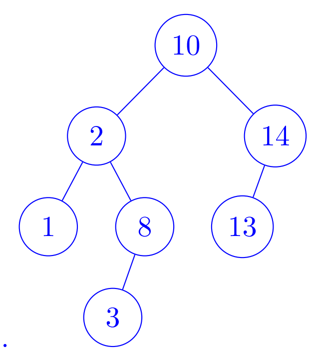
Solution:



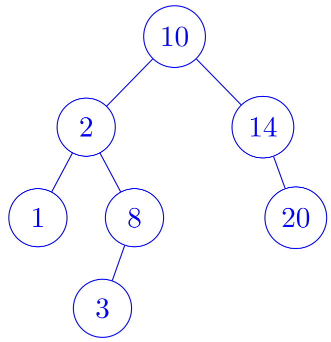
A.



B.



C.



D.

(j) (3') Which of the following is/are **TRUE** about the relation of the number of nodes and the height of AVL trees?

A. The minimal height of an AVL tree with n nodes is $\Theta(\log n)$.

B. The maximal height of an AVL tree with n nodes is $\Theta(n)$.

C. The minimal number of nodes of an AVL tree of height h is $F(h)$, where $F(h) = F(h-1) + F(h-2)$ for $h \geq 2$.

D. The maximal number of nodes of an AVL tree of height h is $G(h)$, where $G(h) = 2G(h-1) + 1$ for $h \geq 1$.

Solution:

A.D. In this case $n = G(h) = 2G(h-1) + 1 = 2^{h+1} - 1$.

Therefore $h = \Theta(\log n)$.

B.C. In this case $n = F(h) = F(h-1) + F(h-2) + 1 = \Theta\left(\left(\frac{\sqrt{5}+1}{2}\right)^h\right)$.

Therefore $h = \Theta(\log n)$ too.

2. (14 points) Fill in Blanks

In this problem, you are asked to fill in the blanks with the correct answers (satisfying the requirements). **Write the most precise and most simplified answer you can think of.**

(a) (2') If we implement a stack by array and when the array is full, we move elements to another double-sized array, then for consecutively n `push`es to the empty stack, the time complexity of each `push` in general is $O(\underline{\quad n \quad})$, but the amortized time complexity is $\Theta(\underline{\quad 1 \quad})$.

(b) (1') If we implement a chained hash table (using singly linked-list) of size M, suppose there are already n elements in it, and the hash function is uniformly random and computed in $\Theta(1)$ time, then the average-case time complexity of inserting an element is $\Theta(\underline{\quad 1 \quad})$.

> **Solution:**
> This problem is controversial. You can get full points no matter whether you answer 1, $\frac{n}{M}$ or $1 + \frac{n}{M}$.
> The correct description of this problem should include a condition: we are sure that the element we will insert is not in the hash table. In this case we should directly insert the element x at the head of the $H(x)$-th linked-list, which is $\Theta(1)$.
> In this case $\frac{n}{M}$ is not correct, because n can be arbitrary large, for example, $M = 10, n = 1000000000$, therefore $\frac{n}{M} \neq \Theta(1)$.

(c) (1') There are $\underline{\quad 15 \quad}$ inversions in the sequence $[4, 5, 7, 3, 2, 6, 8, 1]$.

(d) (1') Consider an array of length n holding an uncommon type of elements, whose comparison take $\Theta(\log(n))$ time. Any in-place sorting algorithm based on comparison will have a worst-case time complexity of $\Omega(\underline{\quad n\log^2 n \quad})$.

(e) (1') Let $T(n) = T(0.3n) + T(0.4n) + \Theta(n), S(n) = S(0.99n) + \Theta(\log n)$, $T(1) = S(1) = 1$, then $T(n) = \underline{\quad \omega \quad}(S(n))$. (Write Θ, o or ω in the blank)

(f) (2') There are $\underline{\quad 132 \quad}$ different binary trees with 6 nodes.

(g) Array $[a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8, a_9, a_{10}]$ represents a binary min-heap containing 10 items, where the key of each item is a distinct integer. (Write all item(s) satisfying requirements.)

i. (2') $\underline{\quad a_1 \quad}$ could be the smallest integer.

ii. (2') $\underline{\quad a_3, a_4, a_{10} \quad}$ could be the fourth smallest integer, if a_5 is the third smallest one.

(h) Assume we have an AVL tree of size n,

i. (1') Insertion operation takes $\Theta(\underline{\quad \log n \quad})$ time,

ii. (1') Erasion operation takes $\Theta(\underline{\quad \log n \quad})$ time.

3. (12 points) Hash Table Operations

There are a series of tuples (a, b) to be stored in a hash table using

- Quadratic probing. The probing function is $H_i(a, b) = (a + b + 0.5i + 0.5i^2) \mod 11$.
- Lazy erasing. A lazy-erased element is marked as E.

There is a hash table T which looks like

Index	0	1	2	3	4	5	6	7	8	9	10
Key Value			$(6, 7)$	$(1, 2)$		$(4, 9)$	E		E		

Hint: Probing sequence means the sequence of Index that you visit, including the initial index i.e. $H_i(a, b)$ when $i = 0$. Follow the format below to fill in the blanks:

$$8, 9, 10, 0, 1, \ldots$$

(a) (2') The load factor of T (including lazy-erased elements) is $\lambda =$ $\frac{5}{11}$.

(b) (2') If we search for $(1, 1)$ in T, the probing sequence is 2, 3, 5, 8, 1.

(c) (3') If we want to insert $(4, 1)$ into T (but we are not sure if it is in T), the probing sequence is 5, 6, 8, 0, and finally it will be inserted at Index 6.

(d) (2') If we want to erase $(4, 9)$ from T, the probing sequence is 2, 3, 5, and finally it will be marked as E at Index 5.

(e) (3') If we create another hash table using lazy erasing but **linear probing**, then is it possible that starting from an empty hash table and after some insert and erase operations, the hash table looks the same as T?

If possible, please write one of such sequence of operations like

$$\text{Insert } (6, 7), \text{ Erase } (6, 0), \cdots$$

If not possible, please explain the reason.

Solution:
Not possible. (1pt)
Explanation: (2pt as below, as long as reasonable)
The probing sequence of searching $(4, 9)$ is $2, 3, 4$ and we find that Index 4 is empty. (1pt)
Therefore $(4, 9)$ is in T but we can't find it after searching, which contradicts. (1pt)

4. (12 points) Merge sort on Linked-lists

Linked-list is a kind of linear data structure that is able to access data one by one. Merge sort is a kind of sort algorithm that takes $\Theta(n \log n)$ time to sort n elements. (Suppose we sort in ascending order, and comparison takes constant time.)

(a) (2') Fill in the table according to what you know about merge sort and doubly linked-lists:

Data Structure \ Operation	Divide	Sub-problem(s)	Merge
Array	$\Theta(1)$	$2T(\frac{n}{2})$	$\Theta($ n $)$
Doubly Linked-list	$\Theta($ n $)$	$2T(\frac{n}{2})$	......(Don't Need This)

(b) Then we think about how to merge two linked-lists. Here is a C++ function about it.

```
/*
x.head() returns the head node of linked-list x(non-empty, otherwise error),
x.nohead() returns a linked-list started from x's second element,
x.empty() return whether x is empty.
con(x,y) returns the list obtained by connecting x and y, where one of x,y
   should be a non-empty linked list and another should be an element.
*/

List merge(const List &x, const List &y) {
    if (x.empty())
        return ____B____;
    if (y.empty())
        return ____A____;
    auto xh = x.head(), yh = y.head();
    if (xh < yh)
        return ____A____;
    else
        return ____C____;
}
```

i. (2') Use some of the following options to fill in the first two blanks:
Hint: Handle the cases where one of the given Lists is empty.
A. `x` B. `y` C. `con(x.head(), y)` D. `con(y.head(), x)`

ii. (2') Use some of the following options to fill in the last two blanks:
Hint: Finish the work in a recursive way
A. `con(xh, merge(x.nohead(), y))` B. `con(merge(y, x.nohead()), xh)`
C. `con(yh, merge(y.nohead(), x))` D. `con(merge(x, y.nohead()), yh)`

iii. (2') The time complexity of the function `merge` is $\Theta($ $n + m$ $)$ (Assume the length of x is n and length of y is m).

(c) (2') We merge two sorted linked-lists $(a_1, a_2, a_3, a_4, a_5)$ and $(b_1, b_2, b_3, b_4, b_5)$ into one. Assume that these elements are distinct except $a_4 = b_3$. Suppose the result is $(b_1, b_2, a_1, a_2, a_3, b_3, a_4, b_4, a_5, b_5)$. From this, you can infer that the number of inversions in the original array is at least ____12____.

(d) (2') In the above parts we discuss situations of doubly linked-lists, if `merge` is used to singly linked-lists, will the time complexity change? (Write "Yes" or "No".) ____**No**____.

5. (12 points) Karatsuba's Algorithm

Let A, B are two polynomials where $A(x) = \sum_{i=0}^{n} a_i x^i, B(x) = \sum_{i=0}^{n} b_i x^i$.

The goal of this problem is to invent an algorithm to multiply two polynomials faster.

(a) (1') The time complexity of calculating the product of two polynomials of n or less degree trivially (i.e. use $C(x) = \sum_{0 \le i,j \le n} a_i b_j x^{i+j}$) is ___**A**___.

A. $\Theta(n^2)$
B. $\Theta(n)$
C. $\Theta(n \log n)$
D. $\Theta(n^{1.5})$

(b) (3') Recall the vertical multiplication on polynomials. Rewrite $A(x) = A_1(x)x^{\frac{n}{2}} + A_2(x)$ and $B(x) = B_1(x)x^{\frac{n}{2}} + B_2(x)$, where A_i, B_i are polynomials of $\frac{n}{2}$ or less degree :

$$\begin{array}{rrr}
 & A_1 x^{\frac{n}{2}} & +A_2 \\
 & \times B_1 x^{\frac{n}{2}} & +B_2 \\
\hline
 & A_1 B_2 x^{\frac{n}{2}} & +A_2 B_2 \\
A_1 B_1 x^n & +A_2 B_1 x^{\frac{n}{2}} & \\
\hline
A_1 B_1 x^n & +(A_1 B_2 + A_2 B_1) x^{\frac{n}{2}} & +A_2 B_2.
\end{array}$$

Write down the recurrence relation of this algorithm's time complexity and calculate it:
$T(n) =$ ___4___ $T(\frac{n}{2}) + \Theta($___n___$) = \Theta($___n^2___$)$.

(c) (3') Recall how Strassen's algorithm works for matrix multiplication: It decreases one time of multiplication (from 8 to 7), then its time complexity turns into $\Theta($___$n^{\log_2 7}$___$)$ from $\Theta(n^3)$. Consider how to reduce one multiplication, we wonder if $A_1B_2 + A_2B_1$ can be calculated with 1 multiplication with proper polynomial calculation:
Hint: You can use any number of additions. Think out reuse A_1B_1 and A_2B_2. (A_1B_1 and A_2B_2 won't be considered as extra multiplication.)
$A_1B_2 + A_2B_1 =$ ___$(A_1 + A_2)(B_1 + B_2) - A_1B_1 - A_2B_2$___

(d) (3') After the modification, we obtain a method to use 1 less multiplication to calculate the polynomial multiplication.
Write down the recurrence relation of the modified algorithm's time complexity and calculate it:
$T(n) =$ ___3___ $T(\frac{n}{2}) + \Theta($___n___$) = \Theta($___$n^{\log_2 3}$___$)$.
The algorithm we get is Karatsuba's algorithm.

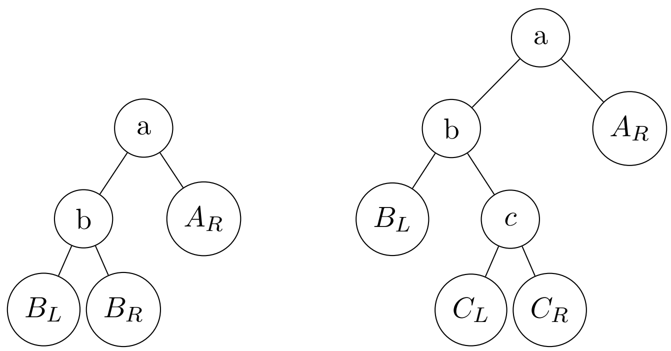
(e) (2') Based on this modification, could the time of multiplication each time decrease more? (Write "Yes" or "No".) ___**No**___.

6. (10 points) Rotate To Balance

In an AVL tree, the imbalance caused by insertion can generally be categorized into four types: **LL** (Left-Left), **RR** (Right-Right), **LR** (Left-Right), and **RL** (Right-Left), where 'L' indicates a left subtree and 'R' indicates a right subtree.

First, consider the tree depicted on the left, where each lowercase letter represents a node, and the uppercase letters represent subtrees. The height of B_L, B_R, A_R is denoted as h, and they are all balanced.

After inserting a node into B_R, the tree transforms into the configuration shown on the right, where C_L or C_R are the subtrees of c. Now, the tree is **unbalanced**.



(a) (b)

(a) (2') The heights of the new subtree C_L and C_R may be ______ and ______ separately (Given one possible solution is enough).

> **Solution:** $h-1$ and h, or h and $h-1$.
> update: This question has some problems. In graph (b), if C_L's height is $h-1$, C_R's height is $h-2$, and the new node is insert as C_L's child, then node a and node c are all unbalanced. So (a) may have another solution: h and $h-2$. And the next question (b) has an undecidable solution for node c, and LR type for node a.
>
> A possible modification method is to add a constrain in the description of the problem that node a is the only unbalanced node, then it will not have such a problem.

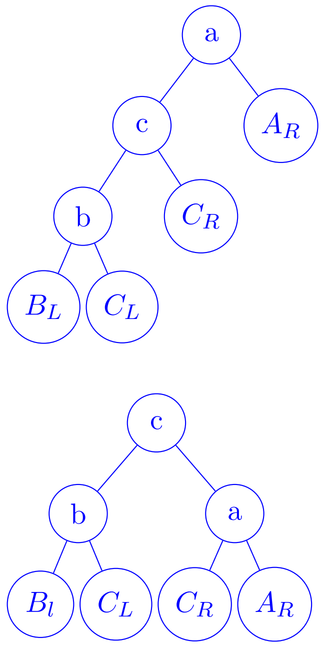
(b) (2') The type of imbalance present in the tree is ___**C**___.

A. LL B. RR C. LR D. RL

(c) (4') Draw the tree after each of the following steps:

1. Perform an RR rotation on the node b.
2. Perform an LL rotation on the tree after step 1.

> **Solution:**



(d) (2') Determine if the final tree is balanced. What conclusion can you draw from this result? (**Hint**: Consider experimenting with additional cases on your own. Your goal is to find the relation between LR, RL, and LL, RR.)

Solution: Yes, it is balanced. The LR and RL rotations are a combination of LL and RR rotations.

7. (10 points) Time complexity of the n-choose-m algorithm

If we want to enumerate all ways of choosing m numbers from $\{1, 2, \cdots, n\}$, we can implement it by m `for` loops:

```
int main() {
    for (a[1] = 1; a[1] <= n - m + 1; ++a[1])
        for (a[2] = a[1] + 1; a[2] <= n - m + 2; ++a[2])
            ...
                for (a[m] = a[m - 1] + 1; a[m] <= n; ++a[m])
                    for (int j = 1; j <= m; ++j)
                        printf("%d%c", a[j], j < m ? ' ' : '\n');
    /* Example: When n = 4, m = 3, the output should be
        1 2 3
        1 2 4
        1 3 4
        2 3 4
    */
}
```

However, if m is not a constant, we are unable to write such m `for` loops.

For either constant or variable m, we can implement it by recursive functions like:

```
void loop(int i, int start, int end) {
    for (a[i] = start; a[i] <= end; ++a[i])
        if (i < m)
            loop(____________, ____________, ____________);
        else
            for (int j = 1; j <= m; ++j)
                printf("%d%c", a[j], j < m ? ' ' : '\n');
}

int main() {
    loop(1, 1, n - m + 1);
}
```

(a) (3') Please fill in the three blanks in the code above.

Solution:
`loop(i + 1, a[i] + 1, end + 1).`
`loop(i + 1, a[i] + 1, n - m + i + 1)` is also OK.

(b) (5') It is difficult to directly evaluate the time complexity of `loop(1, 1, n - m + 1)`. However, by the well-known fact that this algorithm enumerates all ways of choosing m numbers from n elements, we can simply derive the time complexity using the Binomial Coefficient:

$$T(n, m) = \Theta\left(m\binom{n}{m}\right) = \Theta\left(\frac{m \cdot n!}{m!(n-m)!}\right)$$

Now you need to prove mathematically that if m is a constant c, then the time complexity should be $T(n, c) = \Theta(n^c)$. **[Hint: you can check the Stirling Formula on the HINTS page.]**

Solution: Prove by showing that $\frac{T(n,c)}{n^c} = \Theta(1)$.

$$
\begin{aligned}
\frac{T(n,c)}{n^c} &= \frac{c \cdot n!}{c!(n-c)!n^c} \\
&= \Theta\left(\frac{n!}{(n-c)!n^c}\right) && \text{(ignoring constant coefficient, 1pt)} \\
&= \Theta\left(\frac{n^{n+\frac{1}{2}}e^{-n}}{(n-c)^{n-c+\frac{1}{2}}e^{-n+c}n^c}\right) && \text{(Stirling Formula, 1pt)} \\
&= \Theta\left(\frac{n^{n-c+\frac{1}{2}}}{(n-c)^{n-c+\frac{1}{2}}}\right) \\
&= \Theta\left(\left(\frac{u+c}{u}\right)^{u+\frac{1}{2}}\right) && (u \leftarrow n-c) \\
&= \Theta\left(\left(1+\frac{c}{u}\right)^{u} \cdot \left(1+\frac{c}{u}\right)^{\frac{1}{2}}\right) \\
&= \Theta\left(e^c \cdot 1\right) = \Theta(1) && \text{(calculating limits as } u \to \infty\text{, 1pt)}
\end{aligned}
$$

2pts for mathematical correctness and step-by-step deduction.

Method 2:

$$\frac{n!}{(n-c)!} = n(n-1)(n-2)\cdots(n-c+1) = \Theta(n \cdot n \cdot n \cdots n) = \Theta(n^c)$$

but it is not suitable for solving (c).

Method 3:

$$(n-c)^c \leq \frac{n^{n-c+\frac{1}{2}}}{(n-c)^{n-c+\frac{1}{2}}} \leq n^c$$

(c) (2') Now you need to show that if $m = \frac{n}{2}$ instead of a constant, then the time complexity should be $T\left(n, \frac{n}{2}\right) =$ _____ o _____ $\left(n^{\frac{n}{2}}\right)$. (Write Θ, o or ω in the blank).

Solution: We can prove by $T\left(n, \frac{n}{2}\right) = \Theta\left(n^{\frac{1}{2}} 2^n\right) = o\left(n^{\frac{n}{2}}\right)$.

$$
\begin{aligned}
T\left(n, \frac{n}{2}\right) =& \frac{\frac{n}{2} \cdot n!}{\left(\frac{n}{2}\right)! \left(\frac{n}{2}\right)!} \\
=& \Theta\left(\frac{n \cdot n^{n+\frac{1}{2}} e^{-n}}{\left(\frac{n}{2}\right)^{\frac{n}{2}+\frac{1}{2}} e^{-\frac{n}{2}} \left(\frac{n}{2}\right)^{\frac{n}{2}+\frac{1}{2}} e^{-\frac{n}{2}}}\right) && \text{(Stirling Formula)} \\
=& \Theta\left(\frac{n^{n+\frac{3}{2}} e^{-n}}{\left(\frac{n}{2}\right)^{n+1} e^{-n}}\right) \\
=& \Theta\left(\frac{n^{n+\frac{3}{2}} 2^{n+1}}{n^{n+1}}\right) \\
=& \Theta\left(n^{\frac{1}{2}} 2^n\right)
\end{aligned}
$$

And then

$$
\begin{aligned}
& n^{\frac{1}{2}} 2^n = o\left(n^{\frac{n}{2}}\right) \\
\iff & \lim_{n \to \infty} \frac{n^{\frac{1}{2}} 2^n}{n^{\frac{n}{2}}} = 0 \\
\iff & \lim_{n \to \infty} 2^{\left(\frac{1}{2} \log n + n - \frac{n}{2} \log n\right)} = 2^{-\infty} = 0 \\
\Longleftarrow & \frac{1}{2} \log n + n = o\left(\frac{n}{2} \log n\right)
\end{aligned}
$$

Name: ID:

(Please make it clear otherwise unspecified answers may not be counted.)

1 Multiple Choice

(a)	◯ A	◯ B	◯ C	◯ D	(b)	◯ A	◯ B	◯ C	◯ D
(c)	◯ A	◯ B	◯ C	◯ D	(d)	◯ A	◯ B	◯ C	◯ D
(e)	◯ A	◯ B	◯ C	◯ D	(f)	◯ A	◯ B	◯ C	◯ D
(g)	◯ A	◯ B	◯ C	◯ D	(h)	◯ A	◯ B	◯ C	◯ D
(i)	◯ A	◯ B	◯ C	◯ D	(j)	◯ A	◯ B	◯ C	◯ D

2 Fill in Blanks

(a) ________________ ________________ (b) ________________ (c) ________________

(d) ________________ (e) ________________ (f) ________________

(g) ________________ ________________ (h) ________________ ________________

3 Hash Table Operations

(a) __________ (b) ____________________________

(c) ____________________________ __________ (d) __________

(e)

4 Merge sort on Linked-lists

(a)

Data Structure \ Operation	Divide	Sub-problem(s)	Merge
Array	$\Theta(1)$	$2T(\frac{n}{2})$	$\Theta(\_\_\_\_\_\_\_\_\_\_)$
Doubly Linked-list	$\Theta(\_\_\_\_\_\_\_\_\_\_)$	$2T(\frac{n}{2})$	......(Don't Need This)

(b)

(i) 1st blank ◯ A ◯ B ◯ C ◯ D (i) 2nd blank ◯ A ◯ B ◯ C ◯ D
(ii) 1st blank ◯ A ◯ B ◯ C ◯ D (ii) 2nd blank ◯ A ◯ B ◯ C ◯ D

(iii) ________________ (c) ________________ (d) ________________

5 Karatsuba's Algorithm

(a) ◯ A ◯ B ◯ C ◯ D

(b) __________ __________ __________

(c) ____

(d) __________ __________ __________

(e) ◯ Yes ◯ No

6 Rotate To Balance

(a) ________________ ________________ (b) ________________

(c)

(d) ____

7 Time complexity of the n-choose-m algorithm

(a) ________________ ________________ ________________

(b)

(c) ________________