CS101 Algorithms and Data Structures
Fall 2023
Midterm Exam

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

Time: Nov 29th 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. Some Mathematical Formulae

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

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

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

1. (20 points) True or False

Each of the following statements is true (T) or false (F). Write your answers in the **answer sheet**.

(a) (2') The memory consumption of doubly linked lists is $\Theta(n)$ larger than that of singly linked lists, where n is the number of elements.

√ **True** ◯ False

Solution: Doubly linked list needs to store `prev` for each node while singly linked list does not.

(b) (2') There exists an algorithm with run time $f(n)$ such that $f(n) = \omega(n)$ and $f(n) = o(n \log n)$.

√ **True** ◯ False

Solution: Example: $f(n) = n \log \log n$

(c) (2') Under all circumstances, insertion sort has the time complexity $\Theta(n^2)$

◯ True √ **False**

Solution: Best case $\Theta(n)$.

(d) (2') Apply the randomized quick-sort algorithm on the sequence $\langle 7, 1, 4, 2, 8, 5 \rangle$. Suppose the element 1 and 8 are compared, then this comparison must occur in the first round of sequence partitioning and the pivot for that round must be one of 1 and 8.

√ **True** ◯ False

Solution: If the pivot in the first round is not 1 or 8, they will be separated by the pivot.

(e) (2') Given an array A of length n, let $f(n)$ be the expected number of comparisons of applying the randomized quick-sort algorithm to sort it and let $g(n)$ be the expected number of inversions in it, then $f(n) = \Omega(g(n))$.

◯ True √ **False**

Solution: $f(n) = O(n \log n)$ while $g(n) = \Theta(n^2)$

(f) (2') Given two recurrence relation $T(n) = T(0.01n)+T(0.02n)+\Theta(n)$ and $S(n) = S(0.99n)+\Theta(1)$ where $T(0) = S(0) = 0$ and $T(1) = S(1) = 1$, then $T(n) = O(S(n))$.

◯ True √ **False**

Solution: $T(n) = \Theta(n)$ by HW4 while $S(n) = \Theta(\log n)$

(g) (2') When we use divide and conquer to solve a problem, we should divide the problem into one or more subproblems with the same scale, then recursively do them and merge their answers at last.

◯ True √ **False**

Solution: Notice that many divide-and-conquer algorithms divide the problem into subproblems of different scales. (e.g. quick sort, median of median)

(h) (2') Any Huffman Coding Tree is a full binary tree with n leaves, where n is the number of characters.

√ **True** ◯ False

Solution: Huffman Coding Trees are all full binary trees and each leaf represents a character.

(i) (2') Given a BST of height h, suppose node x is the only node of depth h. If node y is one of x's ascendants, then deleting y will cause the BST's height to decrease by 1.

◯ True √ **False**

Solution: If y has two children, then y will be deleted by replacing it by the minimal element in its right sub-tree. In this case, the depth of x will not change, and neither does the height of the BST.

(j) (2') An AVL tree with n nodes guarantees the $O(\log n)$ time complexity for search, insert, and delete operations due to the balance property.

√ **True** ◯ False

Solution: All of them do not exceed the tree height $\Theta(h) = \Theta(\log n)$.

2. (15 points) Single Choice

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

(a) (3') Which of the following statements about linked lists, stacks, queues and heaps is **FALSE**?

A. If we use a stack to do DFS on a tree, then the memory we need is related to the height of the tree.

B. We can delete a node in a doubly linked list in $O(1)$ of time.

C. Queues follow the FIFO (First-In-First-Out) rule.

D. If we use a singly linked list, a stack, a queue and a complete binary heap to store 10000 float numbers respectively, the heap costs more memory than others.

Solution:
D. Singly linked list needs to store `next`, while stack, queue, and complete binary heap only requires $O(1)$ additional memory. Thus the singly linked list costs the most memory.

(b) (3') You are given an open addressing hash table of size $M > 2$ with a uniformly distributed hash function, and we are using linear probing. The probability that both the first and the last slot 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. $\frac{4}{M^2}$

Solution: There are 3 cases, each with probability $\frac{1}{M^2}$

- $h(x_1)$ maps to the first slot, and $h(x_2)$ maps to the last slot;
- $h(x_1)$ maps to the last slot, and $h(x_2)$ maps to the first slot;
- $h(x_1)$ maps to the last slot, and $h(x_2)$ maps to the last slot.

(c) (3') You have to sort n data with $\Theta(1)$ extra memory. Which sorting technique will not be appropriate?

A. Merge sort

B. Heap sort

C. Insertion sort

D. Quick sort

Solution: You will get full points if you choose A, D or AD.
The description of this problem is not appropriate. The appropriate one could be

- You have to sort n data with very limited extra memory. Which sorting technique will be the least appropriate one?
- You have to sort n data in-place/with $o(n)$ extra memory.

A. Merge sort takes $\Theta(n)$ extra space.

There are actually some algorithms called "in-place merge sort", but when we refer to merge sort in CS101, we mean the algorithm we taught in class, which is not in-place.

D. Quick sort takes average-case $\Theta(\log n)$ but worst-case $\Theta(n)$ extra space.

However, if tail recursion optimization is applied, the worst-case space complexity can be reduced to $\Theta(\log n)$, but when we refer to quick sort in CS101, we mean the algorithm we taught in class, which is with worst-case $\Theta(n)$ space complexity.

Tail recursion optimization reference in CS101 WeChat official account:

`https://mp.weixin.qq.com/s/ZLGX2y_qR0U4bhL4HFdmWw`

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

A. A binary tree with n nodes has height $O(n)$.

B. The number of descendants of a node is called degree.

C. The height of a tree is equal to $D - 1$, where D is the maximum depth among nodes.

D. In a binary tree, a node is either a full node or a leaf node.

Solution:

B. The number of children of a node is called degree.

C. The height of a tree is also D.

D. Only a full binary tree satisfies this.

(e) (3') Let T_{AVL} and T_{BST} be the time complexity of searching for an element in an AVL tree/BST with n nodes, respectively. Which of the following is **TRUE**?

A. In the best case, $T_{AVL} = O(\log n)$, and $T_{BST} = \Theta(\log n)$.

B. In the best case, $T_{AVL} = \Theta(\log n)$, and $T_{BST} = O(\log n)$.

C. In the worst case, $T_{AVL} = O(n)$, and $T_{BST} = \Theta(n)$.

D. In the worst case, $T_{AVL} = \Theta(n)$, and $T_{BST} = O(n)$.

Solution: In the best case, $T_{AVL} = T_{BST} = \Theta(1)$.
In the worst case, $T_{AVL} = \Theta(\log n)$, and $T_{BST} = \Theta(n)$.

3. (25 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 **answer sheet**.

(a) (5') Which of the following statements is/are **TRUE**? Suppose the number of elements in data structures is n.

A. The average runtime complexity of finding an element in a singly linked list is $O(1)$.

B. The runtime complexity of pushing an element into a queue with enough capacity is $O(1)$.

C. The runtime complexity of getting access to the last element of a singly linked list is $O(n)$, but in doubly linked list, it's $O(1)$.

D. Given a node in a singly linked list, we can get access to any node in the linked list.

Solution:

C. should be $O(n)$.

D. In singly linked list, we cannot get the `prev` node.

(b) (5') For any two functions $f(n), g(n)$ such that $f(n) > 0, g(n) > 0, f(n) = \Theta(g(n))$ and a constant $a > 0$, which of the following is/are **TRUE**?

A. $f(n) + a = \Theta(g(n) + a)$

B. $af(n) = \Theta(ag(n))$

C. $f(n)^a = \Theta(g(n)^a)$

D. $a^{f(n)} = \Theta(a^{g(n)})$

Solution: By the definition, $\exists n_0, \forall n \geq n_0, c_1 g(n) \leq f(n) \leq c_2 g(n)$.

A. We can prove $\forall n \geq n_0, c_3(g(n)+a) \leq f(n)+a \leq c_4(g(n)+a)$, where $c_4 = \begin{cases} 1, & c_2 \leq 1 \\ c_2, & c_2 > 1 \end{cases}$ and c_3 similarly.

$$\frac{f(n)+a}{g(n)+a} \leq \frac{c_2 g(n)+a}{g(n)+a} \leq \begin{cases} \frac{g(n)+a}{g(n)+a} = 1, & c_2 \leq 1 \\ \frac{c_2 g(n)+c_2 a}{g(n)+a} = c_2, & c_2 > 1 \end{cases}$$

B. We can prove $\forall n \geq n_0, c_1 a g(n) \leq a f(n) \leq c_2 a g(n)$.

C. We can prove $\forall n \geq n_0, c_1^a g(n)^a \leq f(n)^a \leq c_2^a g(n)^a$.

D. Counterexample: $f(n) = n, g(n) = 2n, a = 2$, then $2^n = o(4^n)$.

(c) (5') The algorithm of `std::sort` is based on quick sort, but combined with insertion sort and heap sort. The difference is that the base case of quick sort is when the length of the array is only 1, while the base cases of `std::sort` are

- when the length of the array is not more than a small constant $L = 16$, use insertion sort and return.

- when the recursion depth is more than $2\lfloor \log n \rfloor$ (n is the length of the initial array), use heap sort and return.

Which of the following statements is/are **TRUE** about the comparison between `std::sort` and quick sort?

A. Using insertion sort does not affect the average-case asymptotic time complexity, but reduces the time in a constant level.

B. Because the insertion sort is used, the worst-case asymptotic time complexity of `std::sort` is still $\Theta(n^2)$.

C. Using heap sort reduces the worst-case asymptotic space complexity from $\Theta(n)$ to $\Theta(\log n)$.

D. Using heap sort reduces the worst-case asymptotic time complexity from $\Theta(n^2)$ to $\Theta(n \log^2 n)$.

Solution:

B.D. The time complexity of `std::sort` is $\Theta(n \log n)$ in all cases.

Denote x as the array length in a base case. We have $\sum_x x \leq n$.

The total time of insertion sort is $O(n)$, because

$$\sum_x \Theta(x^2) \leq \sum_x \Theta(L^2) = \sum_x \Theta(1) \leq \Theta(n).$$

The total time of heap sort is $O(n \log n)$, because

$$\sum_x \Theta(x \log x) \leq \sum_x \Theta(x \log n) = \sum_x x\Theta(\log n) \leq \Theta(n \log n)$$

And the total time of quick sort recursion is $O(n \log n)$ because the height of the recursion tree is $\Theta(\log n)$ and the total complexity of each layer is $O(n)$.

Therefore `std::sort` is $O(n \log n)$, and also $\Omega(n \log n)$ by the fact that this is a comparison-based sorting algorithm.

(d) (5') Consider an array$\{a_i\}$ with length $n (n > 3)$, where the elements are **distinct**. We want to use randomized quick sort to make it sorted. Denote the sorted array as $\{b_n\}$(i.e. $b_1 < b_2 < \cdots < b_n$ and $\forall a_i, \exists k_i, b_{k_i} = a_i$). Which of the following statements is/are **TRUE**?

A. $k_i < k_j$ holds if and only if $a_i < a_j$ holds.

B. The probability that b_1 and b_n are compared during sorting is $\frac{1}{n}$

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

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

Solution:

A. Notice that $k_i < k_j \Leftrightarrow b_{k_i} < b_{k_j} \Leftrightarrow a_i < a_j$.

B. Consider the first partition and pivot choosing, if and only if b_1 and b_n are chosen they will be compared. So the probability is $\frac{2}{n}$.

C.D. Notice that if we erase b_i, then b_{i-1} and b_{i+1} will be always compared. If we erase other elements, b_{i-1} and b_{i+1} may be compared. So the probability of b_{i-1} and b_{i+1} are compared is greater than the probability of erasing $b_i = \frac{1}{n-2}$.

(e) (5') Which of the following statements about heap and BST is/are **TRUE**?

A. If a tree satisfies that for each node x, element x is greater than its children (if exist), then the tree is a heap.

B. If the pre-order traversal of a tree is an ascending sequence, then the tree is a heap.

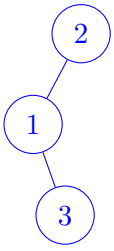
C. If a perfect tree satisfies that for each node x, element x is greater than its left child (if exists) and smaller than its right child (if exists), then the tree is a BST.

D. If a tree is both a heap and a BST with distinct elements, then there cannot be a node with 2 children.

Solution:

B. This condition indicates that any node is smaller than its children.

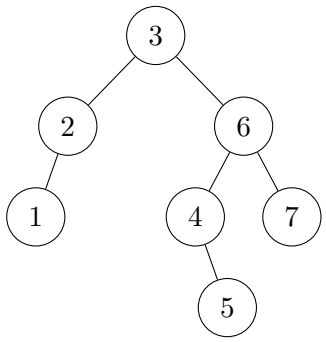
C. It has counter-examples. See the tree below.



D. Suppose node x has two children l, r, then $l < x < r$ by BST, then it is impossible whether it's min heap $x < l, x < r$ or max heap $x > l, x > r$.

4. (9 points) AVL tree operations

Here is an AVL tree. Denote it as T.



For each operation on T, please draw the AVL tree after balance corrections.

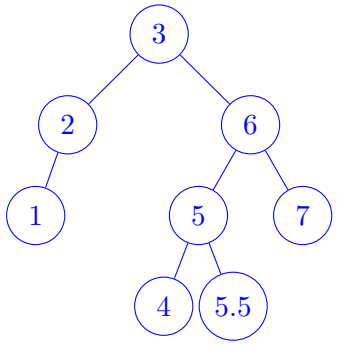
(a) (3') Insert 5.5 into the original tree T.

(b) (3') Remove 2 from the original tree T (**NOT from the previous answer!**).

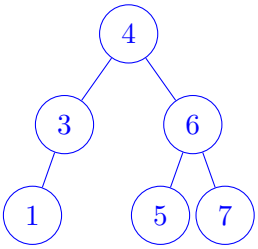
(c) (3') Remove 3 from the original tree T (**NOT from the previous answer!**).

Solution:

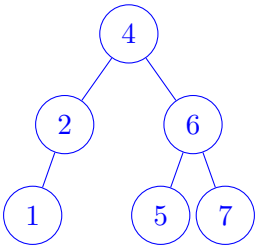
(a)



(b)



(c)



5. (9 points) Analysing the Time Complexity of a C++ Function

This is an algorithm to find the factors of $1, 2, 3, ..., n$. Take $n = 5$ as an example:

- factor[1]: 1
- factor[2]: 1, 2
- factor[3]: 1, 3
- factor[4]: 1, 2, 4
- factor[5]: 1, 5

```
std::vector<std::vector<int>> get_all_factors(int n) {
    std::vector<std::vector<int>> factor(n); // This line is Theta(n)
    for (int i = 1; i <= n; ++i) {
        for (int j = i; j <= n; j += i) {
            factor[j].push_back(i);
        }
    }
    return factor;
}
```

NOTE: Please both analyze and answer in $\Theta(\cdot)$, and simplify your answer. Otherwise you will lose some points. And you should answer clearly like

(a) (3') What is the time complexity of the inner loop? And give an explanation.
Hint: What is the amortized complexity of `push_back`?

> **Solution:** The time complexity of the inner loop is amortized $\Theta(\frac{n}{i})$.
> Explanation: it will run $\lfloor \frac{n}{i} \rfloor = \Theta(\frac{n}{i})$ times, and the amortized time complexity of `push_back` is $\Theta(1)$.

(b) (3') What is the time complexity of the outer loop? And give an explanation.

> **Solution:** The time complexity of the outer loop is $\Theta(n \log n)$.
> Explanation: it will run $\sum_{i=1}^{n} \Theta(\frac{n}{i})$ times,
>
> $$\sum_{i=1}^{n} \frac{n}{i} = n \sum_{i=1}^{n} \frac{1}{i} = \Theta(n \log n)$$

(c) (3') What is the overall time complexity of `get_all_factors`? And give an explanation.

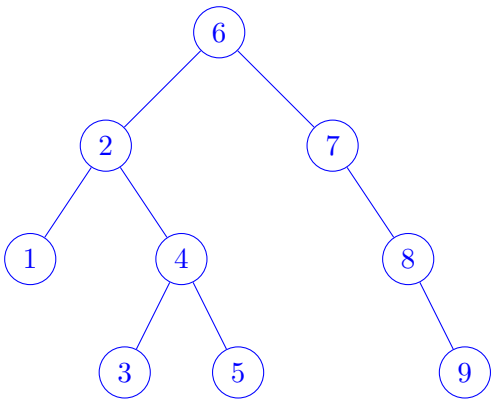
> **Solution:** The overall time complexity is $\Theta(n \log n)$.
> Explanation: $\Theta(n) + \Theta(n \log n) + \Theta(1) = \Theta(n \log n)$.

6. (10 points) Divide-and-Conquer × Post-order Traversal

In this question, you will be given a sequence and required to determine if there exists a binary search tree such that the given sequence is the result of the **post**-order traversal of that binary search tree.

(a) (2') Draw the binary search tree whose **post**-order traversal is $\langle 1, 3, 5, 4, 2, 9, 8, 7, 6 \rangle$ below.

Solution:



(b) (6') Given a sequence $A = \langle a_1, a_2, \ldots, a_n \rangle$ consisting of n **distinct** postive integers, design a **divide-and-conquer** algorithm that determines whether there exists a binary search tree such that A is the result of the **post**-order traversal of that binary search tree. As a reminder, your algorithm should return a Boolean value (i.e. return `True` or `False`). Make sure to provide **clear description** of your algorithm design in **natural language**, with **pseudocode** if necessary. You may use $\texttt{IsPostOrd}(l, r, A)$ to represent the answer of the sub-problem w.r.t. the range $[l, r]$.

Solution:

Algorithm Design:

Key Observation: If the sequence $\langle a_l, a_{l+1}, \ldots, a_r \rangle$ is indeed the post-order traversal of a BST, then a_r must be the root of that BST. Based on this, one of the following must hold:

1. $\forall\, i \in [l, r-1] \cap \mathbb{Z}$, $a_i < a_r$ or $\forall\, i \in [l, r-1] \cap \mathbb{Z}$, $a_i > a_r$
2. $\exists\, k \in [l+1, r-1] \cap \mathbb{Z}$ s.t. $\forall\, i \in [l, k-1] \cap \mathbb{Z}$, $a_i < a_r$ while $\forall\, j \in [k, r-1] \cap \mathbb{Z}$, $a_j > a_r$

Hence, our algorithm design goes as follows:

1. If there is only one element in the sequence (i.e. $l = r$), return `True`.
2. Otherwise, traverse the sequence w.r.t the range $[l, r]$ to check whether it meets one of the two conditions mentioned above and if not, return `False`. Besides, if it meets the second condition, find the corresponding k using the method shown in the pseudocode.
3. If the sequence w.r.t. the range $[l, r]$ meets the first condition, we recursively call the procedure w.r.t. the range $[l, r-1]$ and return its value.
4. If the sequence w.r.t. the range $[l, r]$ meets the second condition, we recursively call the procedure w.r.t. the range $[l, k-1]$ and the range $[k, r-1]$ for the k we find previously and return `True` if and only if both of them return `True`.

Pseudocode:

```
1:  function IsPostOrd(l, r, A = ⟨a_1, a_2, ..., a_n⟩)
2:      if l = r then
3:          return True
4:      end if
5:      root ← a_r, ptr ← l
6:      while a_ptr < root do
7:          ptr ← ptr + 1
8:      end while
9:      boundary ← ptr
10:     while a_ptr > root do
11:         ptr ← ptr + 1
12:     end while
13:     if ptr ≠ r then
14:         return False
15:     else if boundary = l or boundary = r then
16:         return IsPostOrd(l, r − 1, A)
17:     else
18:         return (IsPostOrd(l, boundary − 1, A) && IsPostOrd(boundary, r − 1, A))
19:     end if
20: end function
```

(c) (2') Provide the run-time complexity analysis in the **worse** case scenario of your algorithm in part (b). Make sure to include the **recurrence relation** of the run-time in your solution.

Solution:

The worst case happens when the tree is actually a chain (i.e. a tree such that all its internal nodes has only one child, whose depth is $\Theta(n)$), which lead to $k = l$ or $k = r$ in every recursive call. Let $T(n)$ be the run-time of the algorithm with input of length n in the worst case scenario, then we claim:

$$T(n) = \begin{cases} T(n-1) + \Theta(n), & n > 1, \\ \Theta(1), & n = 1. \end{cases}$$

From this we conclude that $T(n) = \Theta\left(n^2\right)$.

7. (12 points) An Implementation of Double Ended Array

Here we give an implementation of an array that the two operations below

- `push_front`: insert a new item before the first item of the array
- `push_back`: insert a new item after the last item of the array

can both be done in amortized $\Theta(1)$ time, and it's capacity are dynamically expanded.

Specifically, the initial array is empty with capacity 1, and the strategy of expanding the capacity is: If there are no free space (before the first item when a `push_front` is coming/after the last item when a `push_back` is coming), suppose there are c items in the array, we will

- allocate new $3c$ spaces;
- copy all c items to the middle c of the $3c$ spaces (the left c and right c spaces are free initially);
- do the coming `push_front`/`push_back` normally.

(a) (3') Please complete the following demo of this implementation. Use X to indicate a free space.

- initial array: X
- after `push_back(1)`: 1
- after `push_back(2)`: X12
- after `push_back(3)`: XX123X
- after `push_front(4)`: **X4123X**
- after `push_front(5)`: **54123X**
- after `push_front(6)`: **XXXX654123XXXXX**

(b) (4') We are going to analyze some bounds. Denote

- n: the total number of insertions (`push_front`/`push_back`) that are done;
- k: the total times of expanding capacity during these insertions;
- $c_i (1 \le i \le k)$: the number of items copied in the i-th capacity expansion.

For each blank below, please choose the correct choice to give tight bounds:

- **B** $< \frac{n}{c_k} \le$ **D**;
- **C** $\le \frac{c_{i+1}}{c_i} \le$ **D** for all $i = \{1, 2, ..., k-1\}$.

A. $\frac{1}{2}$ B. 1 C. 2 D. 3

Solution:
$\frac{n}{c_k} > 1$ i.e. $n \ge c_k + 1$ because after c_k items copied, at least one insertion is done.
$\frac{n}{c_k} \le 3$ i.e. $n \le 3c_k$ because we can fill at most all $3c_k$ spaces.
$\frac{c_{i+1}}{c_i} \ge 2$ i.e. $c_{i+1} \ge 2c_i$, the equality is satisfied when we repeat inserting on one side.
$\frac{c_{i+1}}{c_i} \le 3$ i.e. $c_{i+1} \le 3c_i$, the equality is satisfied when all $3c_i$ spaces are filled.

(c) (5') Please use the results in (b) to prove that $\sum_{i=1}^{k} c_i = \Theta(n)$, which is the evidence that each insertion can be done in amortized $\frac{\Theta(n)}{n} = \Theta(1)$ time.

Solution:

$1 < \frac{n}{c_k} \leq 3$ means $c_k = \Theta(n)$.

$2 \leq \frac{c_{i+1}}{c_i} \leq 3$ means $2^d \leq \frac{c_k}{c_{k-d}} \leq 3^d$, or $\frac{c_k}{3^d} \leq c_{k-d} \leq \frac{c_k}{2^d}$.

$$\sum_{i=1}^{k} c_i \leq \sum_{d=0}^{k-1} \frac{c_k}{2^d} = 2\left(1 - \frac{1}{2^k}\right) c_k < 2c_k, \sum_{i=1}^{k} c_i = O(c_k)$$

And $\sum_{i=1}^{k} c_i \geq c_k, \sum_{i=1}^{k} c_i = \Omega(c_k)$, so $\sum_{i=1}^{k} c_i = \Theta(c_k) = \Theta(n)$.

1 True or False

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

2 Single Choice

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

3 Multiple Choice

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

4 AVL tree operations

(a)

(b)

(c)

5 Analysing the Time Complexity of a C++ Function

(a)

(b)

(c)

6 Divide-and-Conquer × Post-order Traversal

(a)

(b)

(c)

7 An Implementation of Double Ended Array

(a)

- after `push_front(4)`: _____
- after `push_front(5)`: _____
- after `push_front(6)`: _____

(b)

NOTE: write A,B,C or D instead of direct answers in the blanks.

- _____$< \frac{n}{c_k} \leq$_____;
- _____$\leq \frac{c_{i+1}}{c_i} \leq$_____ for all $i = \{1, 2, ..., k-1\}$.

(c)