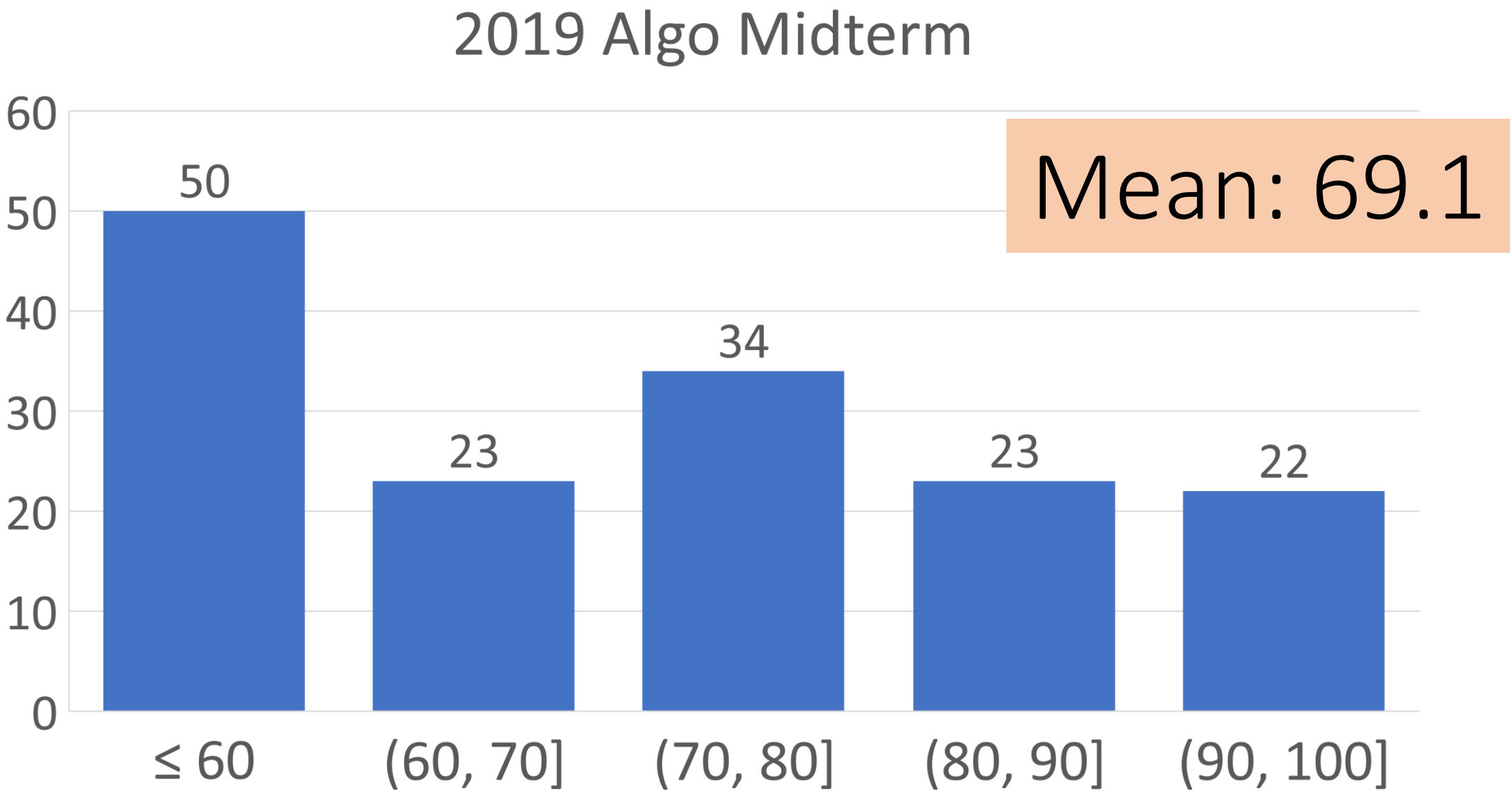
Algorithm 2019

Midterm Exam Solution

教授: 謝孫源 講座教授
助教: 姚凱勛 李昀 鄭力瑋 楊奕正


2019 Algo Midterm
60
50
40
30
20
10
0
50
23
34
23
22
≤ 60
(60, 70]
(70, 80]
(80, 90]
(90, 100]
Mean: 69.1

1. Prove or disprove the following theorem:
(1) (5 pts) For any function $f(n), g(n)$, we can always present the relation between them using big-O or big-Ω. (i.e.$f(n) = O(g(n))$ or $f(n) = \Omega(g(n))$)

<sol>
False,

if we set f(n) = $n^{1+\sin n}$, g(n) = n. because 1+sin$n$ oscillates between 0 and 2. It can't present either $f(n) = O(g(n))$ or $f(n) = \Omega(g(n))$

1. Prove or disprove the following theorem:
(2) (10 pts) $f(n) = \theta(g(n))$ if and only if $f(n) = O(g(n))$ and $\Omega(g(n))$
(Hint: This is a "if and only if" theorem, that means if it is possible to prove, you need to prove it by both two directions.)

<sol>

(=>)

$\Theta(g(n)) \Leftrightarrow 0 \le c_1 g(n) \le f(n) \le c_2 g(n)$---(1)

Because $0 \le f(n) \le c_2 g(n)$, $f(n) = O(g(n))$

Because $0 \le c_1 g(n) \le f(n)$, $f(n) = \Omega(g(n))$

Lead to $f(n) = \theta(g(n))$ => $f(n) = O(g(n))$ and $\Omega(g(n))$

(<=)

$O(g(n)) \Leftrightarrow 0 \le f(n) \le c_3 g(n)$

$\Omega(g(n)) \Leftrightarrow 0 \le c_4 g(n) \le f(n)$

Lead to $0 \le c_4 g(n) \le f(n) \le c_3 g(n) \Leftrightarrow \Theta(g(n))$

2. (10pts) Answer “True” or “False” for the following statements about sorting algorithms.

<sol>

(1)We can use a random number generator to improve the performance of Quick Sort.

True

(2)Radix Sort can only be performed on sequential lists, not on linked lists.

False

(3)Radix Sort is a stable sorting algorithm.

True

(4)The time complexity of best cases is the same for Bubble Sort and Quick Sort.

False

3. (10pts) Show that we can find the both of minimum and maximum using at most 3⌊n/2⌋ comparisons.

<sol>

Compare the elements of a pair to each other.Then compare the larger element to the maximum so far, and compare the smaller element to the minimum so far.

This leads to only 3 comparisons for every 2 elements

If n is even, we do 1 initial comparison and then $3(n-2)/2$ more comparisons.

# of comparisons

$$=\frac{3(n-2)}{2}+1$$

$$=\frac{3n-6}{2}+1$$

$$=\frac{3n}{2}-3+1$$

$$=\frac{3n}{2}-2$$

If n is odd, we do $3(n-1)/2 = 3\lfloor n/2 \rfloor$ comparisons.

In either case, the maximum number of comparisons is $\leq 3\lfloor n/2 \rfloor$

4. (10pts) Give asymptotic tight bounds for T(n) in the following recurrences.

(1) $\boldsymbol{T(n) = 4T\left(\frac{n}{2}\right) + \Theta(n^2)}$

<sol>

$\Theta(n^2 \lg n)$, by Master Theorem

4. (10pts) Give asymptotic tight bounds for T(n) in the following recurrences.

(2) $\boldsymbol{T(n) = 2T(\sqrt{n}) + \lg n}$

<sol>

$$let\ 2^k = n, k = lgn, then$$

$$T(n) = T(\sqrt{n}) + \lg n$$

$$\Rightarrow T(2^k) = T(2^{k/2}) + \lg n$$

$$= T(2^{k/2^2}) + 2 \times \lg n$$

$$= T(2^{k/2^3}) + 3 \times \lg n$$

$$\ldots\ldots$$

$$= T(2^1) + lgk \times \lg n$$

$$= 1 + lgk \times \lg n$$

$$\therefore T(n) = 1 + lgk \times \lg n = 1 + lglgn \times \lg n = \Theta(lglgn \times \lg n)$$

4. (10pts) Give asymptotic tight bounds for T(n) in the following recurrences.

(3) $\boldsymbol{T(n) = 27T\left(\frac{n}{3}\right) + \Theta\left(\frac{n^3}{lgn}\right)}$ **(Hint:** $\boldsymbol{\frac{1}{1} + \frac{1}{2} \ldots\ldots + \frac{1}{n} = \Theta(lgn)}$**)**

<sol>

$$let\ 3^k = n, k = \log_3 n\ , then$$

$$T(n) = 27T(n/3) + \Theta(n^3/lgn)$$

$$\Rightarrow T(3^k) = 3^3 T(3^{k-1}) + \Theta(3^{3k}/k)$$

$$= 3^{3\times 2} T(3^{k-2}) + \Theta(3^{3k}/k) + 3^3\Theta(3^{3(k-1)}/k-1)$$

$$= 3^{3\times 2} T(3^{k-2}) + \Theta(3^{3k}/k) + \Theta(3^{3k}/k-1)$$

$$\ldots\ldots$$

$$= 3^{3k} T(3^{k-k}) + \Theta\left(\frac{3^{3k}}{k}\right) + \Theta\left(\frac{3^{3k}}{k-1}\right) + \Theta\left(\frac{3^{3k}}{k-2}\right) \ldots\ldots + \Theta\left(\frac{3^{3k}}{k-(k-1)}\right)$$

$$= 3^{3k}(\Theta(1)) + 3^{3k}\left(\Theta\left(\frac{1}{k}\right) + \Theta\left(\frac{1}{k-1}\right) + \Theta\left(\frac{1}{k-2}\right) \ldots\ldots + \Theta(1)\right)$$

$$= 3^{3k}(\Theta(1)) + 3^{3k}\left(\sum_{i=1}^{k} \Theta\left(\frac{1}{i}\right)\right)$$

$$= 3^{3k} + 3^{3k}(\Theta(\ln k)) = 3^{3k} + 3^{3k}(\Theta(lg\, k))$$

$$\therefore T(n) = n^3 + n^3\Theta(lglgn) = \Theta(n^3 lglgn)$$

5. Prove the following problems about Heap.

(1) (5 pts) Prove that the height of a Heap with n nodes is Θ(lgn).

<sol>

Heap is a complete binary tree, so for a heap with a height *h*,

$$\sum_{i=0}^{h-1} 2^i + 1 \le n \le \sum_{i=0}^{h} 2^i$$

$$2^h \le n \le 2^{h+1} - 1$$

$$h \le \lg n < h + 1$$

Therefore, for a heap with *n* nodes,

$$\lg n - 1 < h \le \lg n$$

$$h = \lfloor \lg n \rfloor = \theta(\lg n)$$

5. Prove the following problems about Heap.

(2) (10 pts) Prove that the running time of BUILD-MAX-HEAP is O(n).

<sol>

By (1), the total height of the heap is $\lfloor \lg n \rfloor$.

At the height *h*, there is at most $\left\lceil \frac{n}{2^{h+1}} \right\rceil$ nodes.

Time required by HEAPIFY of the height *h* is O(*h*).

So, Time required by BUILD-MAX-HEAP is

$$\sum_{h=0}^{\lfloor \lg n \rfloor} \left\lceil \frac{n}{2^{h+1}} \right\rceil O(h)$$

$$= O\left(n \sum_{h=0}^{\lfloor \lg n \rfloor} \frac{h}{2^h}\right)$$

$$= O\left(n \sum_{h=0}^{\infty} \frac{h}{2^h}\right)$$

5. Prove the following problems about Heap.

(2) (10 pts) Prove that the running time of BUILD-MAX-HEAP is O(n).

By hint, when x = 2, we can get

$$\sum_{h=0}^{\infty} \frac{h}{2^h} = 2$$

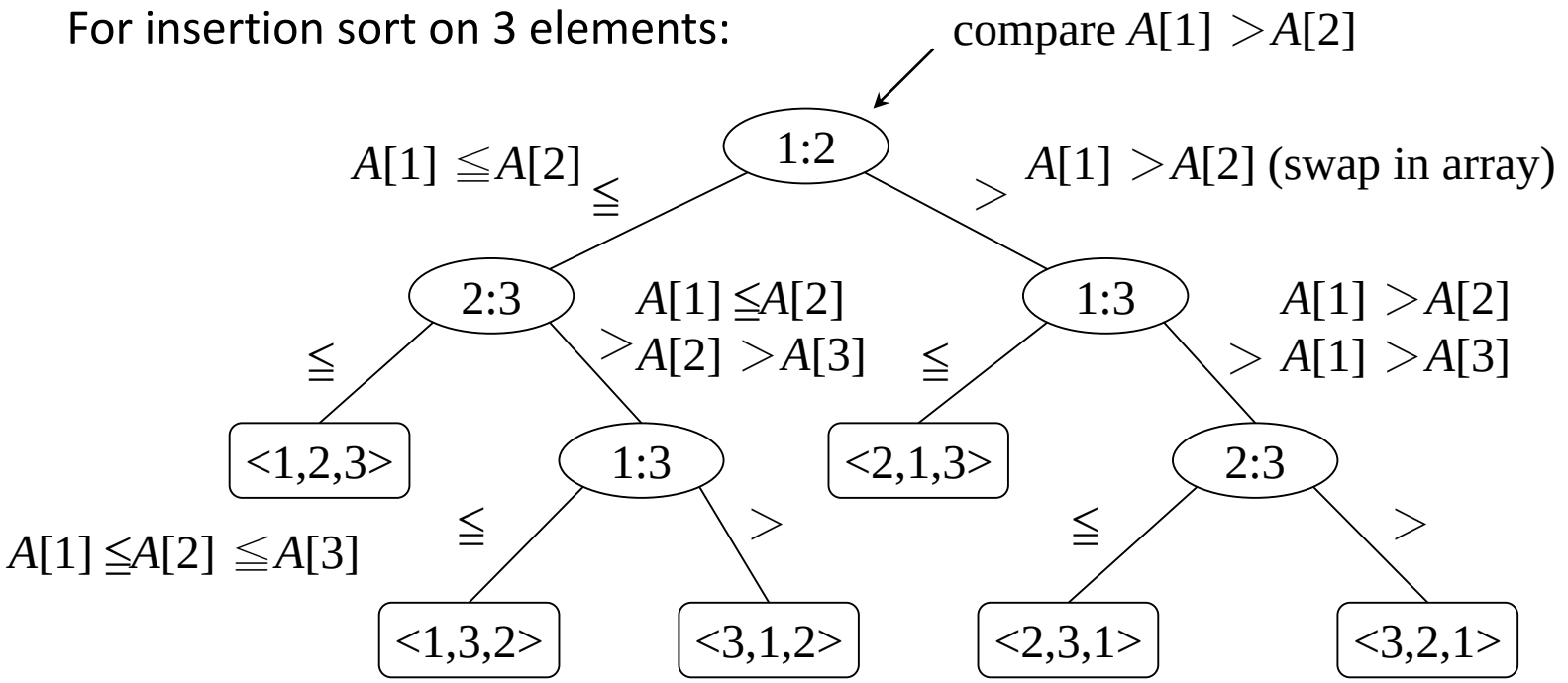
So,

$$O(n \sum_{h=0}^{\infty} \frac{h}{2^h}) = O(2n) = O(n)$$

The running time of BUILD-MAX-HEAP is $O(n)$.

6. (10pts) Show that any comparison sort algorithm requires $\Omega(n \log n)$ comparisons in the worst case.

For insertion sort on 3 elements:



[Each internal node is labeled by indices of array elements **from their original positions**. Each leaf is labeled by the permutation of orders that the algorithm determines.]

6. (10pts) Show that any comparison sort algorithm requires $\Omega(n \log n)$ comparisons in the worst case.

From the preceding discussion, it suffices to determine the height of a decision tree in which each permutation appears as a reachable leaf. Consider a decision tree of height h with l reachable leaves corresponding to a comparison sort on n elements. Because each of the n! permutation of the input appears as some leaf, we have $n! \leqq l$.

Since a binary tree of height h has no more than 2^h leaves, we have

$n! \leqq l \leqq 2^h$

which, by taking algorithms, implies

$h \geqq \lg(n!) = \Omega(n \lg n)$ (since the lg function is monotonically increasing)

6. (10pts) Show that any comparison sort algorithm requires $\Omega(n \log n)$ comparisons in the worst case.

- 若個別舉例說明(詳述原因 而非只有時間複雜度) 一個5分 兩個8分 三個10分
- 只有寫comparison sort跟時間複雜度給3分(若上課有講的寫三個以上給5分)

7.
(1) Explain what is a stable sorting algorithm, that is, what does it mean for a sorting algorithm to be “stable”?
(2) Explain how come Counting-Sort is a stable sorting algorithm.
(3) Is the modified algorithm stable? Please answer “Yes” or “No”, and explain your reason.

- (a) 如果鍵值相同之資料，在排序後相對位置與排序前相同時，即為stable sort algorithm。
- (b) 因C array中存放的是該數值的最後一個在B array中出現的位置，所以要從A的最後一個element開始放入B，放入後將C陣列數值-1，下個有相同數值的element就會擺在前一位。
- (c) No，承(b)的解釋，若改為順向for loop會使相對位置相反，因從A的頭開始拿，原本在前面的會被放到後面。

評分標準:
- 3題都對，+10分
  - 有在(2)或(3)講到C陣列的作用，和要從A的末端開始放
- 錯一小題，+5分
- 錯兩小題，+3分
- (2)說明不夠，+8分
- 小錯，+9分

8. (10pts) What is the running time of Quick Sort when all elements of array A have the same value?

- The running time of QUICKSORT when all elements of array A have the same value will be equivalent to the worst case running of QUICKSORT since no matter what pivot is picked, QUICKSORT will have to go through all the values in A. And since all values are the same, each recursive call will lead to unbalanced partitioning.

- Thus the recurrence will be: $T(n) = T(n-1) + \Theta(n)$
- The above recurrence has the solution: $T(n) = \Theta(n^2)$
- Hence the running time of QUICKSORT in this case is $\Theta(n^2)$

評分標準:

- 只要寫出$\Theta(n^2)$，就給對

9. (10pts) An algorithm is *asymptotically optimal* if the time complexity of the best case of the algorithm is equal to the time complexity of the worst case of the algorithm. Determine the following algorithms are *asymptotically optimal* or not, and briefly

(1) **Quick Sort**

Not asymptotically optimal, O(n^2)≠ O(*n* log *n*)

(2) **Merge Sort**

Asymptotically optimal, O(*n* log *n*)= O(*n* log *n*)

(3) **Insertion Sort**

Not asymptotically optimal, O(n^2) ≠ O(*n*)

(4) **Heap Sort**

Asymptotically optimal, O(*n* log *n*)= O(*n* log *n*)

9. (10pts) An algorithm is *asymptotically optimal* if the time complexity of the best case of the algorithm is equal to the time complexity of the worst case of the algorithm. Determine the following algorithms are *asymptotically optimal* or not, and briefly

- 分數隨答對題數增加：0→3 →5 →8 →10
- 若best或worst的時間複雜度寫錯等等，扣1分
- 沒寫原因會扣分(可能扣至該小題沒分)