Quiz 1 Solutions

- Do not open this quiz booklet until you are directed to do so. Read all the instructions on this page.
- When the quiz begins, write your name on every page of this quiz booklet.
- This quiz contains 4 problems, some with multiple parts. You have 80 minutes to earn 80 points.
- This quiz booklet contains 13 pages, including this one. Two extra sheets of scratch paper are attached. Please detach them before turning in your quiz at the end of the examination period.
- This quiz is closed book. You may use one handwritten A4 or $8\frac{1}{2}'' \times 11''$ crib sheet. No calculators or programmable devices are permitted.
- Write your solutions in the space provided. If you need more space, write on the back of the sheet containing the problem. Do not put part of the answer to one problem on the back of the sheet for another problem, since the pages may be separated for grading.
- Do not waste time and paper rederiving facts that we have studied. It is sufficient to cite known results.
- Do not spend too much time on any one problem. Read them all through first, and attack them in the order that allows you to make the most progress.
- Show your work, as partial credit will be given. You will be graded not only on the correctness of your answer, but also on the clarity with which you express it. Be neat.
- Good luck!

Problem	Parts	Points	Grade	Grader
1	4	12		
2	1	7		
3	11	44		
4	3	17		
Total		80		

Name: ______________________________

Problem 1. Asymptotic Running Times [12 points] (4 parts)

For each algorithm listed below,

- give a recurrence that describes its worst-case running time, and
- give its worst-case running time using Θ-notation.

You need not justify your answers.

(a) Binary search

Solution: $T(n) = T(n/2) + \Theta(1) = \Theta(\lg n)$

(b) Insertion sort

Solution: $T(n) = T(n-1) + \Theta(n) = \Theta(n^2)$

(c) Strassen's algorithm

Solution: $T(n) = 7T(n/2) + \Theta(n^2) = \Theta(n^{\lg 7})$

(d) Merge sort

Solution: $T(n) = 2T(n/2) + \Theta(n) = \Theta(n \lg n)$

Problem 2. Substitution Method [7 points]

Consider the recurrence

$$\begin{aligned} T(n) &= T(n/2) + T(n/4) + n \ , \\ T(m) &= 1 \quad \text{for } m \le 5. \end{aligned}$$

Use the substitution method to give a tight upper bound on the solution to the recurrence using O-notation.

Solution: We guess $T(n) = O(n)$, which leads to the induction hypothesis $T(m) \le cm$ for all $m < n$. For $c \ge 1$, we have the base cases $T(n) = 1 \le cn$ for $n \le 5$. The induction hypothesis yields

$$T(n) = T(n/2) + T(n/4) + n \le cn/2 + cn/4 + n = (3c/4 + 1)n.$$

If we choose $c = 4$, then $T(n) \le (3 + 1)n = 4n = cn$. By induction on n, $T(n) \le cn$ for $c \ge 4$ and all $n \ge 1$.

Problem 3. True or False, and Justify [44 points] (11 parts)

Circle **T** or **F** for each of the following statements to indicate whether the statement is true or false, respectively. If the statement is correct, briefly state why. If the statement is wrong, explain why. The more content you provide in your justification, the higher your grade, but be brief. Your justification is worth more points than your true-or-false designation.

T F The solution to the recurrence $T(n) = 3T(n/3) + O(\lg n)$ is $T(n) = \Theta(n \lg n)$.

Solution: False. Case 3 of the master theorem applies: $f(n) = O(n^{\log_3 3}) = O(n)$ for $f(n) = O(\lg n)$, hence, $T(n) = O(n)$.

T F Let F_k denote the kth Fibonacci number. Then, the n^2th Fibonacci number F_{n^2} can be computed in $O(\lg n)$ time.

Solution: True. The n^2th Fibonacci number can be computed in $O(\lg n^2) = O(\lg n)$ time by using square and multiply method with matrix

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}.$$

T F Suppose that an array contains n numbers, each of which is -1, 0, or 1. Then, the array can be sorted in $O(n)$ time in the worst case.

Solution: **True.** We may use counting sort. We first add 1 to each of the elements in the input array such that the precondition of counting sort is satisfied. After running counting sort, we subtract 1 from each of the elements in the sorted output array.

A solution based on partitioning is as follows. Let $A[1\,..\,n]$ be the input array. We define the invariant

- $A[1\,..\,i]$ contains only -1,
- $A[i+1\,..\,j]$ contains only 0, and
- $A[h\,..\,n]$ contains only $+1$.

Initially, $i = 0$, $j = 0$, and $h = n + 1$. If $h = j + 1$, then we are done; the array is sorted. In the loop we examine $A[j+1]$. If $A[j+1] = -1$, then we exchange $A[j+1]$ and $A[i+1]$ and we increase both i and j with 1 (as in partition in quicksort). If $A[j+1] = 0$, then we increase j with 1. Finally, if $A[j+1] = +1$, then we exchange $A[j+1]$ and $A[h-1]$ and we decrease h by 1.

T F An adversary can provide randomized quicksort with an input array of length n that forces the algorithm to run in $\omega(n \lg n)$ time on that input.

Solution: **False.** As we saw in lecture, for *any* input, the expected running time of quicksort is $O(n \lg n)$, where the expectation is taken over the random choices made by quicksort, independent of the choice of the input.

T F The array

$$20 \quad 15 \quad 18 \quad 7 \quad 9 \quad 5 \quad 12 \quad 3 \quad 6 \quad 2$$

forms a max-heap.

Solution: True.

- $A[1] = 20$ has children $A[2] = 15 \leq 20$ and $A[3] = 18 \leq 20$.
- $A[2] = 15$ has children $A[4] = 7 \leq 15$ and $A[5] = 9 \leq 15$.
- $A[3] = 18$ has children $A[6] = 5 \leq 18$ and $A[7] = 12 \leq 18$.
- $A[4] = 7$ has children $A[8] = 3 \leq 7$ and $A[9] = 6 \leq 7$.
- $A[5] = 9$ has child $A[10] = 2$.
- $A[6], \ldots, A[10]$ have no children.

T F Heapsort can be used as the auxiliary sorting routine in radix sort, because it operates in place.

Solution: False. The auxiliary sorting routine in radix sort needs to be stable, meaning that numbers with the same value appear in the output array in the same order as they do appear in the input array. Heapsort is not stable. It does operate in place, meaning that only a constant number of elements of the input array are ever stored outside the array.

T F There exists a comparison sort of 5 numbers that uses at most 6 comparisons in the worst case.

Solution: **False.** The number of leaves of a decision tree which sorts 5 numbers is $5!$ and the height of the tree is at least $\lg(5!)$. Since $5! = 120$, $2^6 = 64$, and $2^7 = 128$, we have $6 < \lg(5!) < 7$. Thus at least 7 comparisons are required.

T F Suppose that a hash table with collisions resolved by chaining contains n items and has a load factor of $\alpha = 1/\lg n$. Assuming simple uniform hashing, the expected time to search for an item in the table is $O(1/\lg n)$.

Solution: **False.** The expected time to search for an item in the table is $O(1+\alpha) = O(1 + 1/\lg n) = O(1)$. At least a constant running time $O(1)$ is needed to search for an item; subconstant running time $O(1/\lg n)$ is not possible.

T F Let X be an indicator random variable such that $\mathrm{E}[X] = 1/2$. Then, we have $\mathrm{E}\left[\sqrt{X}\right] = 1/\sqrt{2}$.

Solution: **False.** Since X is an indicator random variable, $X = 0$ or $X = 1$. For both possible values $\sqrt{X} = X$, which implies that $\mathrm{E}\left[\sqrt{X}\right] = \mathrm{E}[X] = 1/2$.

T F Suppose that a hash table of m slots contains a single element with key k and the rest of the slots are empty. Suppose further that we search r times in the table for various other keys not equal to k. Assuming simple uniform hashing, the probability is r/m that one of the r searches probes the slot containing the single element stored in the table.

Solution: **False.** The probability p that one of the r searches collides with the single element stored in the table is equal to 1 minus the probability that none of the r searches collides with the single element stored in the table. That is, $p = 1 - (1 - 1/m)^r$.

T F Let S be a set of n integers. One can create a data structure for S so that determining whether an integer x belongs to S can be performed in $O(1)$ time in the worst case.

Solution: **True.** Perfect hashing.

Problem 4. Close Numbers [17 points] (3 parts)

Consider a set S of $n \geq 2$ distinct numbers. For simplicity, assume that $n = 2^k + 1$ for some $k \geq 0$. Call a pair of distinct numbers $x, y \in S$ ***close*** in S if

$$|x - y| \leq \frac{1}{n-1}\left(\max_{z \in S} z - \min_{z \in S} z\right) ,$$

that is, if the distance between x and y is at most the average distance between consecutive numbers in the sorted order.

(a) Explain briefly why every set S of $n \geq 2$ distinct numbers contains a close pair of numbers.

Solution: Without loss of generality, assume $S = \{z_1, z_2, \ldots, z_n\}$, with $z_i \leq z_{i+1}$. The average distance between two consecutive numbers z_i and z_{i+1} is

$$\frac{1}{n-1}\sum_{i=1}^{n-1}(z_{i+1} - z_i) = \frac{1}{n-1}(z_n - z_1).$$

There exists at least one pair of consecutive numbers x and y whose distance between them is less than or equal to the avearge. The result then follows from the definition of the *close* pair.

(b) Suppose that we partition S around a pivot element $p \in S$, organizing the result into two subsets of S: $S_1 = \{x \in S \mid x \leq p\}$ and $S_2 = \{x \in S \mid x \geq p\}$. Prove that either

1. every pair $x, y \in S_1$ of numbers that is close in S_1 is also close in S, or
2. every pair $x, y \in S_2$ of numbers that is close in S_2 is also close in S.

Show how to determine, in $O(n)$ time, a value $k \in \{1, 2\}$ such that every pair $x, y \in S_k$ of numbers that is close in S_k is also close in S.

Solution: Without loss of generality, assume that the elements in S_i are in sorted order. For $k = 1, 2$, let a_k be the average distance between two consecutive numbers in S_k, and let n_k the number of elements in S_k. Using the result from Part (a), we have

$$a_1 = \frac{1}{n_1 - 1}\left(\max_{z \in S_1} z - \min_{z \in S_1} z\right) = \frac{1}{n_1 - 1}\left(p - \min_{z \in S} z\right),$$

and

$$a_2 = \frac{1}{n_2 - 1}\left(\max_{z \in S_2} z - \min_{z \in S_2} z\right) = \frac{1}{n_2 - 1}\left(\max_{z \in S} z - p\right).$$

The average distance a between two consecutive numbers in S in sorted order is then given by

$$\begin{aligned} a &= \frac{1}{n-1}\left(\max_{z \in S} z - \min_{z \in S} z\right) \\ &= \frac{1}{n-1}\left(p - \min_{z \in S} z\right) + \frac{1}{n-1}\left(\max_{z \in S} z - p\right) \\ &= \frac{n_1 - 1}{n-1} a_1 + \frac{n_2 - 1}{n-1} a_2. \end{aligned}$$

Note that $n_1 + n_2 = n + 1$, because p is included in both S_1 and S_2. So, a is a weighted average of a_1 and a_2:

$$a = (1 - \alpha)a_1 + \alpha a_2,$$

where $\alpha = (n_2 - 1)/(n - 1)$.

Suppose that $a_1 \leq a_2$. If x and y are a close pair in S_1, then

$$|x - y| \leq a_1 = (1 - \alpha)a_1 + \alpha a_1 \leq (1 - \alpha)a_1 + \alpha a_2 = a.$$

This implies that every close pair in S_1 is also a close pair in S. Similarly, if $a_2 \leq a_1$, then every close pair in S_2 is a close pair in S.

The average distance a_k can be computed in $O(n)$ time, by searching for the minimum or the maximum number in S_k. Therefore, the subset S_k with the specified property can be computed in $O(n)$ time.

(c) Describe an $O(n)$-time algorithm to find a close pair of numbers in S. Explain briefly why your algorithm is correct, and analyze its running time. (*Hint:* Use divide and conquer.)

Solution: The idea is to partition S recursively until we find a close pair.

1. Determine the median of S and use it to partion S into S_1 and S_2.
2. Use the result from Part (b) to determine the set S_k that contains a close pair of S.
3. Recurse on S_k until S_k contains 2 elements.

Since each recursive step reduces the cardinality of the set by roughly a half, the recursion is guaranteed to terminate. After each recursive step, the remaining set contains a close pair of S.

Step 1 takes $O(n)$ time in the worst case, if we use the deterministic median-finding algorithm. Step 2 takes $O(n)$ time based on the result from Part (b). Therefore, the running time of the algorithm is given by the following recurrence:

$$T(n) = T(n/2) + O(n),$$

with the solution $T(n) = O(n)$ according to the master theorem.

SCRATCH PAPER — Please detach this page before handing in your exam.

SCRATCH PAPER — Please detach this page before handing in your exam.