Quiz 1 Solutions

- Do not open this quiz booklet until you are directed to do so. Read all the instructions first.
- When the quiz begins, write your name on every page of this quiz booklet.
- The quiz contains five multi-part problems. You have 80 minutes to earn 80 points.
- This quiz booklet contains **13** pages, including this one. An extra sheet of scratch paper is attached. Please detach it before turning in your quiz.
- 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	Points	Grade	Initials
1	11		
2	19		
3	10		
4	20		
5	20		
Total	80		

Name: Solutions

Circle your recitation letter and the name of your recitation instructor:

David A B Steve C D Hanson E F

Problem -1. Recurrences [11 points]

Solve the following recurrences. Give tight, i.e. $\Theta(\cdot)$, bounds.

(a) $T(n) = 81T(\frac{n}{9}) + n^4 \lg n$ **[2 points]**

Solution: $T(n) = \Theta(n^4 \lg n)$ by Case 3 of the Master Method. We have: $n^{\log_b a} = n^2$ and $f(n) = n^4 \lg n$. Thus, if $\epsilon = 1$, then $f(n) = \Omega(n^{2+\epsilon})$. The regularity condition, $a \cdot f(n/b) < c \cdot f(n)$ holds here for $c = .99$.
You received one point if you mentioned Case 3 of the Master Method and showed that it complied with the regularity condition and one point if you gave the correct answer.

(b) $T(n) = 2T(\frac{n}{2}) + \frac{n}{\lg n}$ **[2 points]**

Solution: Note that we can not use the Master Method to solve this recurrence. If we expand this recurrence, we obtain:

$$T(n) = \frac{n}{\lg n} + \frac{n}{\lg \frac{n}{2}} + \frac{n}{\lg \frac{n}{4}} \cdots \frac{n}{\lg \frac{n}{n}} = n \cdot \sum_{i=1}^{\lg n} \frac{1}{\lg \frac{n}{2^i}} =$$

$$n \cdot \sum_{i=1}^{\lg n} \frac{1}{\lg n - i} = n \cdot \sum_{i=1}^{\lg n} \frac{1}{i} = \Theta(n \lg \lg n).$$

(c) $T(n) = 4T(n/3) + n^{\log_3 4}$ **[2 points]**

Solution: $T(n) = \Theta(n^{\log_3 4} \lg n)$ by Case 2 of the Master Method.
You received one point if you mentioned the correct case of the Master Method and one point for the correct answer. A common error was to not cite which case of the Master Method you used.

(d) Write down and solve the recurrence for the running time of the DETERMINISTIC SELECT algorithm using groups of 3 (rather than 5) elements. The code is provided on the last page of the exam. **[5 points]**

Solution: $T(n) = T(n/3) + T(2n/3) + cn$. We solve this recurrence using a recursion tree, see Figure 1. The height of the tree is at least $\log_3 n$ and is at most $\log_{3/2} n$ and the sum of the costs in each level is n. Hence $T(n) = \Theta(n \lg n)$.
A correct recurrence received 3 points. One point was awarded for solving whichever recurrence you gave correctly and one point was awarded for justification. An incorrect recurrence received 1 point if it was almost correct.
A common error was omitting the $T(2n/3)$ term in the recurrence.

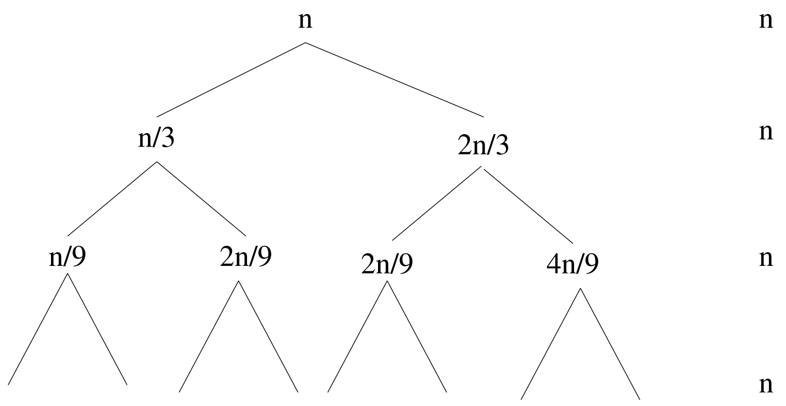

Figure 1: Recursion tree

Problem -2. True or False, and Justify [19 points]

Circle **T** or **F** for each of the following statements, and briefly explain why. The better your argument, the higher your grade, but be brief. Your justification is worth more points than your true-or-false designation.

(a) T F Given a set of n integers, the number of comparisons necessary to find the maximum element is $n - 1$ and the number of comparisons necessary to find the minimum element is $n - 1$. Therefore the number of comparisons necessary to simultaneously find the smallest and largest elements is $2n - 2$. **[3 points]**

Solution: False. In recitation, we gave an algorithm that uses $3n/2$ comparisons to simultaneously find the maximum and minimum elements.
A common error was to say, ”You can just keep track of min and max at the same time”, which does not say how many comparisons that would take or how it would work.

(b) T F There are n people born between the year 0 A.D. and the year 2003 A.D. It is possible to sort all of them by birthdate in $o(n \lg n)$ time.

Solution: True. We can represent a birthdate using c digits. This database can then be sorted in $O(c \cdot n) = O(n)$ time using RADIX SORT.
Common errors included saying, ”Use Mergesort” or ”Use Radix Sort” (with no explanation).

(c) T F There is some input for which RANDOMIZED QUICKSORT always runs in $\Theta(n^2)$ time. **[3 points]**

Solution: False. The *expected* running time of RANDOMIZED QUICKSORT is $\Theta(n \lg n)$. This applies to *any* input.
A common error was to say, "RANDOMIZED QUICKSORT will take $O(n^2)$ only if it's very unlucky", but you needed to say what it's expected runtime is.

(d) T F The following array A is a Min Heap:

$$2 \quad 5 \quad 9 \quad 8 \quad 10 \quad 13 \quad 12 \quad 22 \quad 50$$

[3 points]
Solution: True.

(e) T F If $f(n) = \Omega(g(n))$ and $g(n) = O(f(n))$ then $f(n) = \Theta(g(n))$. **[3 points]**

Solution: False. For example $f(n) = n^2$ and $g(n) = n$.
A common error was to mix up f and g or O and Ω.

(f) T F Let k, i, j be integers, where $k > 3$ and $1 \leq i, j \leq k$. Let h_{ij}^k be the hash function mapping a k-bit integer $b_1 b_2 \ldots b_k$ to the 2-bit value $b_i b_j$. For example, $h_{31}^8(\mathbf{0}0\mathbf{1}01011) = 10$.
The set $\{h_{ij}^k : 1 \leq i, j \leq k\}$ is a universal family of hash functions from k-bit integers into $\{00, 01, 10, 11\}$. **[4 points]**

Solution: False. Take $x = 0, y = 1$. Then all of x's binary digits are the same as y's, except for the least significant bit. Thus, if we choose one of the k hash functions at random, $\Pr[h(x) = h(y)] = (k^2 - k)/k^2 = (k-1)/k > 1/4$. If this were a universal class of hash functions, then this probability should be at most $1/4$.
A common error was to show a single counter-example without explanation.

Problem -3. Short Answer [10 points]

Give *brief*, but complete, answers to the following questions.

(a) Explain the differences between average-case running time analysis and expected running time analysis. For each type of running time analysis, name an algorithm we studied to which that analysis was applied.

[5 points]

Solution: The average-case running time does not provide a guarantee for the worst-case, i.e. it only applies to a specific input distribution, while the expected running time provides a guarantee (in expectation) for every input.

In the average-case running time, the probability is taken over the random choices over an input distribution. In the expected running time, the probability is taken over the random choices made by the algorithm.

The average-case running time of BUCKET SORT is $\Theta(n)$ when the input is chosen at random from the uniform distribution. The expected running time of QUICK SORT is $\Theta(n \lg n)$.

(b) Suppose we have a hash table with $2n$ slots with collisions resolved by chaining, and suppose that $n/8$ keys are inserted into the table. Assume each key is equally likely to be hashed into each slot (**simple uniform hashing**). What is the expected number of keys for each slot? Justify your answer. **[5 points]**

Solution: Define X_j (for $j = 1, \ldots, n$) to be the indicator which is 1 if element j hashes to slot i and 0 otherwise. Then $E[X_j] = Pr[X_j = 1] = 1/(2n)$. Then the expected number of elements in slot i is $E[\sum_{j=1}^{n/8} X_j] = \sum_{j=1}^{n/8} E[X_j] = n/(8(2n)) = 1/16$ by linearity of expectation.

Problem -4. Checking Properties of Sets [20 points]

In this problem, more efficient algorithms will be given more credit. Let S be a finite set of n positive integers, $S \subset \mathbb{Z}^+$. You may assume that all basic arithmetic operations, i.e. addition, multiplication, and comparisons, can be done in unit time. In this problem, partial credit will be given for correct but inefficient algorithms.

(a) Design an $O(n \lg n)$ algorithm to verify that:

$$\forall\, T \subseteq S, \quad \sum_{t \in T} t \geq |T|^3.$$

In other words, if there is some subset $T \subseteq S$ such that the sum of the elements in T is less than $|T|^3$, the your algorithm should output "no". Otherwise, it should output "yes". Argue (informally) that your algorithm is correct and analyze its running time. **[10 points]**

Solution: Sort the set S in time $O(n \lg n)$ using $O(n \lg n)$ sorting algorithm such as MERGE SORT or HEAP SORT. For each k between 1 and n, verify that $(\sum_{i=0}^{k} s_i) \geq k^3$. By maintaining a running sum, checking this sum for each value of k requires only one addition and one comparison operation. Thus, the total running time is $O(n \lg n) + O(n) = O(n \lg n)$.

Correctness: The key observation is that *every* set of k elements is at least k^3 iff the sum of the smallest k elements is at least k^3. Thus, if we sort the elements, the sum of the first k elements, namely the k smallest elements, will be at least k^3 iff every set of size k has sum at least k^3.

The following procedure CHECKALLSUMS takes as input an array A containing S and an integer n indicating the size of S.

CHECKALLSETS(A, n)

Sort A using MERGESORT

sum $\leftarrow 0$

from $i \leftarrow 1$ to n

    sum $\leftarrow$ sum + $A[i]$

    if sum $< k^3$

        return "no"

return "yes"

A common error was to use COUNTING SORT, which would not necessarily be efficient for an arbitrary set of n integers.

Many people got this problem backwards, i.e. they tried to find *some* set such that its sum was at least its size cubed. This was not heavily penalized.

(b) In addition to S and n, you are given an integer $k, 1 \leq k \leq n,$. Design a more efficient (than in part **(a)**) algorithm to verify that:

$$\forall\, T \subseteq S,\ |T| = k, \quad \sum_{t \in T} t \geq k^3.$$

In other words, if there is some subset $T \subseteq S$ such that T contains exactly k elements and the sum of the elements in T is less than k^3, then your algorithm should output "no". Otherwise, it should output "yes". Argue (informally) that your algorithm is correct and analyze its running time. **[10 points]**

Solution: Find the kth smallest element using the linear time DETERMINISTICSELECT algorithm. Then find the k smallest elements using the PARTITION procedure. Check that the sum of these k smallest elements is greater than k^3. The total runtime is $O(n) + O(n) + O(k) = O(n)$.

CHECKSETSOFSIZEK(A, n, k)
Run SELECT(A, n, k) to find k^{th} smallest element
Run PARTITION(A, n, k) to put k smallest elements in $A[1 \ldots k]$
sum $\leftarrow 0$
from $i \leftarrow 1$ to k
    sum $\leftarrow$ sum + $A[i]$
if sum $< k^3$
    return "no"
return "yes"

Correctness: Again, as in part **(a)**, the key observation is that *every* set of k elements has sum at least k^3 iff the sum of the k smallest elements is at least k^3.

Problem -5. Finding the Missing Number [20 points]

Suppose you are given an unsorted array A of all integers in the range 0 to n except for one integer, denoted the *missing number*. Assume $n = 2^k - 1$.

(a) Design a $O(n)$ Divide and Conquer algorithm to find the missing number. Partial credit will be given for non Divide and Conquer algorithms. Argue (informally) that your algorithm is correct and analyze its running time. **[12 points]**

Solution: We can use SELECT to find the median element and check to see if it is in the array. If it is not, then it is the missing number. Otherwise, we PARTITION the array around the median element x into elements $\leq x$ and $> x$. If the first one has size less than $x + 1$, then we recurse on this subarray. Otherwise we recurse on the other subarray.

The procedure MISSINGINTEGER$(A, n, [i, j])$ takes as input an array A and a range $[i, j]$ in which the missing number lies.

MISSINGINTEGER$(A, [i, j])$
Determine median element x in range $i \ldots j$
Check to see if x is in A
PARTITION A into B, elements $< x$, and C, elements $\geq x$
If SIZE$(B) < x + 1$
    MISSINGINTEGER$(B, [i, x])$
Else MISSINGINTEGER$(C, [x + 1, j])$

The running time is $O(n)$ because the recurrence for this algorithm is $T(n) = T(n/2) + n$, which is $O(n)$ by the Master Method.

Common errors included using a randomized, instead of deterministic, partitioning scheme and using COUNTING SORT and then stepping through the array to find adjacent pairs that differ by two, which is not a Divide and Conquer approach.

(b) Suppose the integers in A are stored as k-bit binary numbers, i.e. each bit is 0 or 1. For example, if $k = 2$ and the array $A = [01, 00, 11]$, then the missing number is 10.

Now the only operation to examine the integers is BIT-LOOKUP(i, j), which returns the jth bit of number $A[i]$ and costs unit time. Design an $O(n)$ algorithm to find the missing number. Argue (informally) that your algorithm is correct and analyze its running time. **[8 points]**

Solution: We shall examine bit by bit starting from the least significant bit to the most significant bit. Make a count of the number of 1's and 0's in each bit position, we can find whether the missing number has a 0 or 1 at the bit position being examined. Having done this, we have reduced the problem space by half as we have to search only among the numbers with that bit in that position. We continue in this manner till we have exhausted all the bit-positions (ie., k to 1).
The algorithm is as follows. For convenience, we have 3 sets S, S_0, S_1 which are maintained as link-lists. S contains the indices of the elements in A which we are going to examine while S_0 (S_1) contains the indices of the elements in A which have 0 (1) in the bit position being examined.

```
MISSING-INTEGER(A, n)
1  k ← lg n        ▷ n = 2^k − 1
2  S ← {1, 2, . . . , n}
3  S0 ← S1 ← {}         ▷ Initialized to Empty List
4  count0 ← count1 ← 0
5  for posn ← k downto 1 do
6     for each i ∈ S do
7           bit ← BIT-LOOKUP(i, posn)
8           if bit = 0
9              then count0 ← count0 + 1
10                  Add i to S0
11             else count1 ← count1 + 1
12                  Add i to S1
13    if count0 > count1
14          then missing[posn] ← 1
15                  S ← S1
16          else missing[posn] ← 0
17                  S ← S0
18    S0 ← S1 ← {}
19    count0 ← count1 ← 0
20 return missing
```

It can be noted that the following invariant holds at the end of the loop in Step 5-19.

- The bits of the missing integer in the bit position (*posn* to k) is given by *missing*[*posn*] . . . *missing*[k].
- $S = \{i : j^{th}$ bit of $A[i] =$ *missing*$[j]$ for *posn* $\leq j \leq k\}$

This loop invariant ensures the correctness of the algorithm.
Each loop iteration (Step 5-19) makes $|S|$ BIT-LOOKUP operations. And the size of S is halved in each iteration. Hence total number of BIT-LOOKUP operations is $\sum_{i=0}^{k-1} \frac{n}{2^k}$, which is $O(n)$.
The grading for this problem was (-6) points for an algorithm that runs in $\Theta(nk)$, like RADIX SORT (or any of a number of built-from-scratch radix-sort-like approaches). Another point was deducted for for not recognizing that $k = Theta(logn) => Theta(nk) = Theta(nlogn)$ and thus that the above solutions are *not* $O(n)$ time.

DETERMINISTICSELECT(A, n, i)

Divide the elements of the input array A into groups of 3 elements.
Find median of each group of 3 elements and put them in array B.
Call DETERMINISTICSELECT$(B, n/3, n/6)$ to find median of the medians, x.
Partition the input array around x into A_1 containing k elements $\leq x$
    and A_2 containing $n - k - 1$ elements $\geq x$.
If $i = k + 1$, then return x.
    Else if $i \leq k$, DETERMINISTICSELECT(A_1, k, i).
    Else if $i > k$, DETERMINISTICSELECT$(A_2, n - k - 1, i - (k + 1))$.

SCRATCH PAPER

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