Quiz 1 Solution

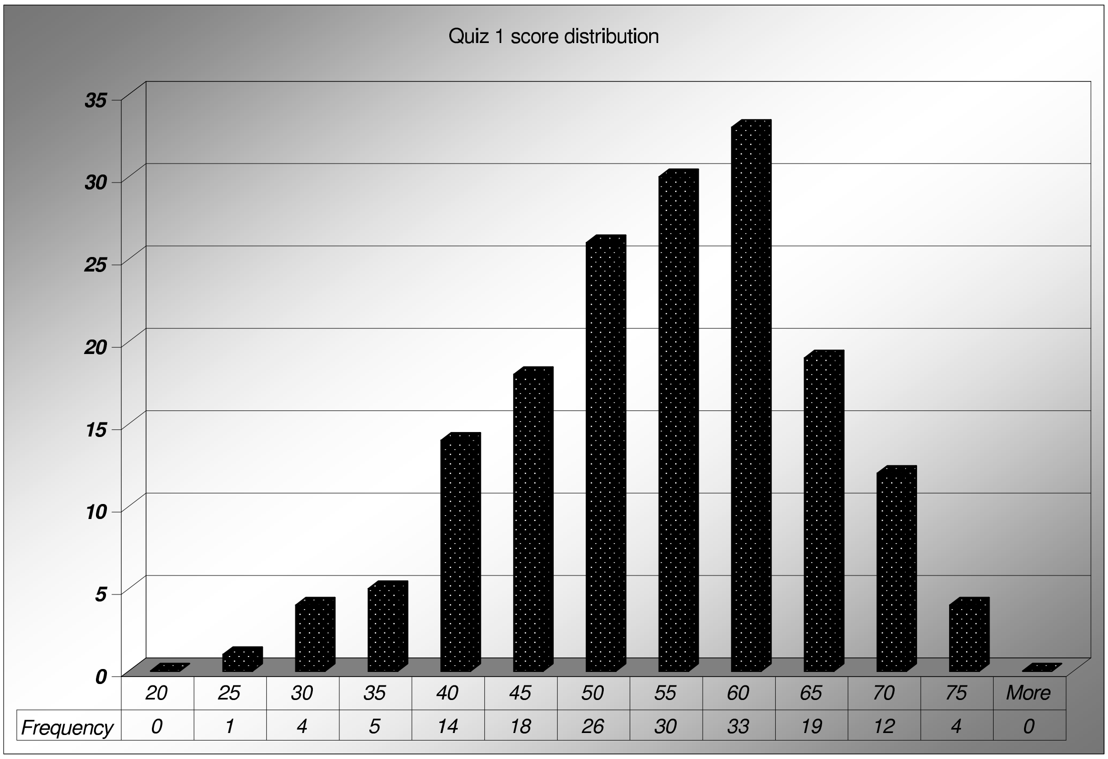

Figure 1: grade distribution

Problem	Points	Grade	Initials
1	16		
2	19		
3	45		
Total	80		

Name: ____

MIT students: Circle the name of your recitation instructor:

Bob Chong George Jennifer Rachel

Problem 1. Match-up [16 points]

Fill in the following table by specifying, for each algorithm, the letter of the recurrence for its running time $T(n)$ and the numeral of the solution to that recurrence. Points will be deducted for wrong answers, so do not guess unless you are reasonably sure.

Algorithm	**Recurrence**	**Solution**
Merge sort (worst case)	d	6
Binary search (worst case)	b	2
Randomized quicksort (expected)	g	6
Strassen's algorithm (worst case)	a	3
Selection (worst case)	c	1
Randomized selection (expected)	e	1

Letter	**Recurrence**
a	$T(n) = 7\,T(n/2) + \Theta(n^2)$
b	$T(n) = T(n/2) + \Theta(1)$
c	$T(n) = T(\lceil n/5 \rceil) + T(7n/10 + 6) + \Theta(n)$
d	$T(n) = 2\,T(n/2) + \Theta(n)$
e	$T(n) = \dfrac{2}{n} \displaystyle\sum_{k=\lfloor n/2 \rfloor}^{n} T(k) + \Theta(n)$
f	$T(n) = 2\,T(n/2) + \Theta(1)$
g	$T(n) = \dfrac{2}{n} \displaystyle\sum_{k=0}^{n-1} T(k) + \Theta(n)$
h	$T(n) = T(n-1) + \Theta(n)$

Numeral	**Solution**
1	$T(n) = \Theta(n)$
2	$T(n) = \Theta(\lg n)$
3	$T(n) = \Theta(n^{\lg 7})$
4	$T(n) = \Theta(1)$
5	$T(n) = \Theta(n^2)$
6	$T(n) = \Theta(n \lg n)$

Problem 2. Merging several sorted lists [19 points]

Professor Fiorina uses the following algorithm for merging k sorted lists, each having n/k elements. She takes the first list and merges it with the second list using a linear-time algorithm for merging two sorted lists, such as the merging algorithm used in merge sort. Then, she merges the resulting list of $2n/k$ elements with the third list, merges the list of $3n/k$ elements that results with the fourth list, and so forth, until she ends up with a single sorted list of all n elements.

(a) Analyze the worst-case running time of the professor's algorithm in terms of n and k.

Solution: Merging the first two lists,each of n/k elements, takes $2n/k$ time. Merging the resulting $2n/k$ elements with the third list of n/k elements takes $3n/k$ time, and so on. Thus for a total of k list, we have:

$$
\begin{aligned}
Time &= \frac{2n}{k} + \frac{3n}{k} + \cdots + \frac{kn}{k} \\
&= \sum_{i=2}^{k} \frac{in}{k} \\
&= \frac{n}{k} \sum_{i=2}^{k} i \\
&= \frac{n}{k} \frac{(k+2)(k-1)}{2} \\
&= \theta(nk)
\end{aligned}
$$

(b) Briefly describe an algorithm for merging k sorted lists, each of length n/k, whose worst-case running time is $O(n \lg k)$. Briefly justify the running time of your algorithm. (If you cannot achieve $O(n \lg k)$, do the best you can for partial credit.)

Solution: There are several possible answers (only one is required). One method is to repeatedly pair up the lists, and merge each pair. This method can also be seen as a tail component of the execution merge sort, where the analysis is clear. Finally, a conceptually simple method is to store a min priority queue of the minimum elements of each of the k lists. At each step, we output the extracted minimum of the priority queue, and using sattelite data determine from which of the k lists it came, and insert the next element from that list into the priority queue.

(c) Argue that there are

$$\frac{n!}{((n/k)!)^k}$$

different ways to interleave k lists, each of length n/k, into a single list of length n.

Solution: For the first list,

$$\text{number of ways of choosing } \frac{n}{k} \text{ elements from } n \text{ elements} = \frac{n!}{(n-\frac{n}{k})!(\frac{n}{k})!}$$

Note that of the $(\frac{n}{k})!$ possible permutation of the elements in the list, only 1 permutation is valid because the elements in the list is already sorted. Hence the $(\frac{n}{k})!$ term in the denominator.

For the second list,

$$\text{number of ways of choosing } \frac{n}{k} \text{ elements from the remaining } n-\frac{n}{k} \text{ elements} = \frac{(n-\frac{n}{k})!}{(n-\frac{2n}{k})!(\frac{n}{k})!}$$

And so on. Therefore, total number of ways of forming the k lists, each of size n/k, from n elements is:

$$\begin{aligned}
&= \frac{n!}{(n-\frac{n}{k})!(\frac{n}{k})!}\frac{(n-\frac{n}{k})!}{(n-\frac{2n}{k})!(\frac{n}{k})!}\cdots\frac{(n-\frac{(k-2)n}{k})!}{(n-\frac{(k-1)n}{k})!(\frac{n}{k})!}\frac{(n-\frac{(k-1)n}{k})!}{(n-\frac{kn}{k})!(\frac{n}{k})!} \\
&= \frac{n!}{((n/k)!)^k}
\end{aligned}$$

This is exactly the number of different ways of interleaving the k list, into the single list of n elements

(d) Prove a lower bound of $\Omega(n \lg k)$ on the worst-case running time of any comparison algorithm for merging k sorted lists each of length n/k. (*Hint:* Use the fact that $(n/e)^n \leq n! \leq n^n$.)

Solution: Consider the decision tree used to make the comparisons. The worst-case running time corresponds to the height of the tree:

$$
\begin{aligned}
Time &= \lg \frac{n!}{((n/k)!)^k} \\
&= \lg n! - \lg((\frac{n}{k})!)^k \\
&\geq \lg n! - \lg((\frac{n}{k})^{\frac{n}{k}})^k \\
&= \lg n! - n \lg \frac{n}{k} \\
&= \lg n! - n \lg n + n \lg k \\
&\geq \lg(\frac{n}{e})^n - n \lg n + n \lg k \\
&= n \lg n - n \lg e - n\; lgn + n \lg k \\
&= n \lg k - n \lg e \\
&= \Omega(n \lg k)
\end{aligned}
$$

Problem 3. True or False, and Justify [45 points]

Circle **T** or **F** for each of the following statements, and briefly 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.

(a) T F A constant $n_0 \geq 1$ exists such that for any $n \geq n_0$, there is an array of n elements such that insertion sort runs faster than merge sort on that input.

Solution: true.
If the n elements are already in sorted order, the running time for insertion sort is $O(n)$, whereas that of merge sort is $O(nlgn)$.

(b) T F Consider the algorithm from the textbook for building a max-heap:

BUILD-MAX-HEAP(A)
1 *heap-size*$[A] \leftarrow$ *length*$[A]$
2 **for** $i \leftarrow \lfloor$ *length*$[A]/2 \rfloor$ **downto** 1
3 **do** MAX-HEAPIFY(A, i)

On an array of n elements, this code runs in $\Theta(n \lg n)$ time in the worst case, because there are $\Theta(n)$ calls to MAX-HEAPIFY, and the worst-case time for any call is $\Theta(\lg n)$.

Solution: false.
$n \lg n$ is not a tight bound. In fact, Build-Max-Heap runs in $\Theta(n)$ time, as proven during recitation.

(c) T F Given a number a and a positive integer n, the value $a^{2^n} = a^{(2^n)}$ can be computed in $O(n \lg n)$ time by repeated squaring. (Assume that multiplying two numbers takes constant time.)

Solution: true.

$$a^{2^n} = a^{2^{n-1}} a^{2^{n-1}}$$

Letting $T(n) = a^{2^n}$, we have:

$$\begin{aligned} T(n) &= 2T(n-1) + \Theta(1) \\ &= \Theta(n) \\ &= O(n \lg n) \end{aligned}$$

Note: it is important to understand that $\Theta(n) = O(n \lg n)$.

(d) T F An adversary can force randomized quicksort to run in $\Omega(n^2)$ time by providing as input an already sorted or reverse-sorted array of size n.

Solution: false.
The running time is not dependent on the input that the adversary provides. This is because radomized quicksort will randomly choose a pivot to partition, independent of the input.

(e) T F For any integer-valued random variable $X \geq 0$, we have

$$\Pr\{X = 0\} \geq 1 - \mathrm{E}[X] \ .$$

Solution: true

$$\begin{aligned} \Pr\{X \geq t\} &\leq \frac{\mathrm{E}[x]}{t} \\ \Pr\{X \geq 1\} &\leq \mathrm{E}[x] \end{aligned}$$

$$\begin{aligned} \Pr\{X = 0\} &= 1 - \Pr\{X \geq 1\} \\ &\geq 1 - \mathrm{E}[X] \end{aligned}$$

(f) T F Bucket sort is a suitable auxiliary sort for radix sort.

Solution: true.
An auxilliary sort for radix sort must be stable. Bucket sort is stable and hence suitable for use as an auxilliary sort for radix sort.

(g) T F Consider two implementations of a hash table with m slots storing n keys, where $n < m$. Let $T_c(m, n)$ be the expected time for an unsuccessful search in the table if collisions are resolved by chaining, using the assumption of simple uniform hashing. Let $T_o(m, n)$ be the expected time for an unsuccessful search in the table if collisions are resolved by open addressing, using the assumption of uniform hashing. Then, we have $T_c(m, n) = \Theta(T_o(m, n))$.

Solution: false.

$$
\begin{aligned}
T_c(m,n) &= \Theta(1 + \frac{n}{m}) \\
T_o(m,n) &= \Theta(\frac{1}{1 - \frac{n}{m}})
\end{aligned}
$$

In the worst case, in the chaining version, we need to search through all the values in one particular slot. On the other hand, in the open addressing version, we need to search through every slot. Hence $T_c(m, n) \neq \Theta(T_o(m, n))$.

(h) T F Let $\mathcal{H}$ be a class of universal hash functions for a hash table of size $m = n^3$. Then, if we use a random $h \in \mathcal{H}$ to hash n keys into the table, the expected number of collisions is at most $1/n$.

Solution: true.
Number of ways to choose 2 keys out of n keys is $\frac{n(n-1)}{2}$
Therefore, the expected number of collisions,

$$\begin{aligned} \text{E}\left[\text{number of collisions}\right] &= \frac{n(n-1)}{2}\frac{1}{m} \\ &= \frac{n(n-1)}{2}\frac{1}{n^3} \\ &\leq \frac{1}{n} \end{aligned}$$

(i) T F Given a set $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$ of points in the plane, the $\sqrt{n}$ points closest to the origin $(0, 0)$ (using normal Euclidean distance) can be found in $O(n)$ time in the worst case.

Solution: true.

Finding the distance of every point from the origin takes $O(n)$. We can then use Select to get the $\sqrt{n}th$ point. That again takes $O(n)$. Finally, partitioning around the $\sqrt{n}th$ point requies $O(n)$. Each of the steps takes $O(n)$. Therefore, total time is $O(n)$