8-1

8-1 Probabilistic lower bounds on comparison sorting

In this problem, we prove a probabilistic $\Omega(n \lg n)$ lower bound on the running time of any deterministic or randomized comparison sort on n distinct input elements. We begin by examining a deterministic comparison sort A with decision tree T_A. We assume that every permutation of A's inputs is equally likely.

a. Suppose that each leaf of T_A is labeled with the probability that it is reached given a random input. Prove that exactly $n!$ leaves are labeled $1/n!$ and that the rest are labeled 0.

b. Let $D(T)$ denote the external path length of a decision tree T; that is, $D(T)$ is the sum of the depths of all the leaves of T. Let T be a decision tree with $k > 1$ leaves, and let LT and RT be the left and right subtrees of T. Show that $D(T) = D(LT) + D(RT) + k$.

c. Let $d(k)$ be the minimum value of $D(T)$ over all decision trees T with $k > 1$ leaves. Show that $d(k) = \min_{1 \le i \le k-1} \{d(i) + d(k-i) + k\}$. (*Hint:* Consider a decision tree T with k leaves that achieves the minimum. Let i_0 be the number of leaves in LT and $k - i_0$ the number of leaves in RT.)

d. Prove that for a given value of $k > 1$ and i in the range $1 \le i \le k-1$, the function $i \lg i + (k-i) \lg (k-i)$ is minimized at $i = k/2$. Conclude that $d(k) = \Omega(k \lg k)$.

e. Prove that $D(T_A) = \Omega(n! \lg(n!))$, and conclude that the average-case time to sort n elements is $\Omega(n \lg n)$.

Now, consider a *randomized* comparison sort B. We can extend the decision-tree model to handle randomization by incorporating two kinds of nodes: ordinary comparison nodes and "randomization" nodes. A randomization node models a random choice of the form RANDOM$(1, r)$ made by algorithm B; the node has r children, each of which is equally likely to be chosen during an execution of the algorithm.

f. Show that for any randomized comparison sort B, there exists a deterministic comparison sort A whose expected number of comparisons is no more than those made by B.

sol)

a. For a comparison algorithm A to sort, no two input permutations can reach the same leaf of the decision tree, so there must be at least $n!$ leaves reached in T_A, one for each possible input permutation. Since A is a deterministic algorithm, it must always reach the same leaf when given a particular permutation as input, so at most $n!$ leaves are reached (one for each permutation). Therefore exactly $n!$ leaves are reached, one for each input permutation.

These $n!$ leaves will each have probability $1/n!$, since each of the $n!$ possible permutations is the input with the probability $1/n!$. Any remaining leaves will have probability 0, since they are not reached for any input.

Without loss of generality, we can assume for the rest of this problem that paths leading only to 0-probability leaves aren't in the tree, since they cannot affect the running time of the sort. That is, we can assume that T_A consists of only the $n!$ leaves labeled $1/n!$ and their ancestors.

b. If $k > 1$, then the root of T is not a leaf. This implies that all of T's leaves are leaves in LT and RT. Since every leaf at depth h in LT or RT has depth $h+1$ in T, $D(T)$ must be the sum of $D(LT)$, $D(RT)$, and k, the total number of leaves. To prove this last assertion, let $d_T(x) =$ depth of node x in tree T. Then,

$$\begin{aligned}
D(T) &= \sum_{x \in \text{leaves}(T)} d_T(x) \\
&= \sum_{x \in \text{leaves}(LT)} d_T(x) + \sum_{x \in \text{leaves}(RT)} d_T(x) \\
&= \sum_{x \in \text{leaves}(LT)} (d_{LT}(x) + 1) + \sum_{x \in \text{leaves}(RT)} (d_{RT}(x) + 1) \\
&= \sum_{x \in \text{leaves}(LT)} d_{LT}(x) + \sum_{x \in \text{leaves}(RT)} d_{RT}(x) + \sum_{x \in \text{leaves}(T)} 1 \\
&= D(LT) + D(RT) + k\,.
\end{aligned}$$

c. To show that $d(k) = \min_{1 \le i \le k-1} \{d(i) + d(k-i) + k\}$ we will show separately that

$$d(k) \le \min_{1 \le i \le k-1} \{d(i) + d(k-i) + k\}$$

and

$$d(k) \ge \min_{1 \le i \le k-1} \{d(i) + d(k-i) + k\}\,.$$

- To show that $d(k) \le \min_{1 \le i \le k-1} \{d(i) + d(k-i) + k\}$, we need only show that $d(k) \le d(i) + d(k-i) + k$, for $i = 1, 2, \ldots, k-1$. For any i from 1 to $k-1$ we can find trees RT with i leaves and LT with $k-i$ leaves such that $D(RT) = d(i)$ and $D(LT) = d(k-i)$. Construct T such that RT and LT are the right and left subtrees of T's root respectively. Then

$$\begin{aligned}
d(k) &\le D(T) && \text{(by definition of } d \text{ as min } D(T) \text{ value)} \\
&= D(RT) + D(LT) + k && \text{(by part (b))} \\
&= d(i) + d(k-i) + k && \text{(by choice of } RT \text{ and } LT)\,.
\end{aligned}$$

- To show that $d(k) \ge \min_{1 \le i \le k-1} \{d(i) + d(k-i) + k\}$, we need only show that $d(k) \ge d(i) + d(k-i) + k$, for some i in $\{1, 2, \ldots, k-1\}$. Take the tree T with k leaves such that $D(T) = d(k)$, let RT and LT be T's right and left subtree, respecitvely, and let i be the number of leaves in RT. Then $k-i$ is the number of leaves in LT and

$$\begin{aligned}
d(k) &= D(T) && \text{(by choice of } T) \\
&= D(RT) + D(LT) + k && \text{(by part (b))} \\
&\ge d(i) + d(k-i) + k && \text{(by defintion of } d \text{ as min } D(T) \text{ value)}\,.
\end{aligned}$$

Neither i nor $k-i$ can be 0 (and hence $1 \le i \le k-1$), since if one of these were 0, either RT or LT would contain all k leaves of T, and that k-leaf subtree would have a D equal to $D(T) - k$ (by part (b)), contradicting the choice of T as the k-leaf tree with the minimum D.

d. Let $f_k(i) = i \lg i + (k-i)\lg(k-i)$. To find the value of i that minimizes f_k, find the i for which the derivative of f_k with respect to i is 0:

$$\begin{aligned} f_k'(i) &= \frac{d}{di}\left(\frac{i \ln i + (k-i)\ln(k-i)}{\ln 2}\right) \\ &= \frac{\ln i + 1 - \ln(k-i) - 1}{\ln 2} \\ &= \frac{\ln i - \ln(k-i)}{\ln 2} \end{aligned}$$

is 0 at $i = k/2$. To verify this is indeed a minimum (not a maximum), check that the second derivative of f_k is positive at $i = k/2$:

$$\begin{aligned} f_k''(i) &= \frac{d}{di}\left(\frac{\ln i - \ln(k-i)}{\ln 2}\right) \\ &= \frac{1}{\ln 2}\left(\frac{1}{i} + \frac{1}{k-i}\right) . \\ f_k''(k/2) &= \frac{1}{\ln 2}\left(\frac{2}{k} + \frac{2}{k}\right) \\ &= \frac{1}{\ln 2} \cdot \frac{4}{k} \\ &> 0 \qquad\qquad \text{since } k > 1 . \end{aligned}$$

Now we use substitution to prove $d(k) = \Omega(k \lg k)$. The base case of the induction is satisfied because $d(1) \ge 0 = c \cdot 1 \cdot \lg 1$ for any constant c. For the inductive step we assume that $d(i) \ge ci \lg i$ for $1 \le i \le k-1$, where c is some constant to be determined.

$$\begin{aligned} d(k) &= \min_{1 \le i \le k-1} \{d(i) + d(k-i) + k\} \\ &\ge \min_{1 \le i \le k-1} \{c(i \lg i + (k-i)\lg(k-i)) + k\} \\ &= \min_{1 \le i \le k-1} \{c f_k(i) + k\} \\ &= c\left(\frac{k}{2}\lg\frac{k}{2} + \left(k - \frac{k}{2}\right)\lg\left(k - \frac{k}{2}\right)\right) + k \\ &= ck \lg\left(\frac{k}{2}\right) + k \\ &= c(k \lg k - k) + k \\ &= ck \lg k + (k - ck) \\ &\ge ck \lg k \qquad \text{if } c \le 1 , \end{aligned}$$

and so $d(k) = \Omega(k \lg k)$.

e. Using the result of part (d) and the fact that T_A (as modified in our solution to part (a)) has $n!$ leaves, we can conclude that

$$D(T_A) \ge d(n!) = \Omega(n! \lg(n!)) .$$

$D(T_A)$ is the sum of the decision-tree path lengths for sorting all input permutations, and the path lengths are proportional to the run time. Since the $n!$ permutations have equal probability $1/n!$, the expected time to sort n random elements (1 input permutation) is the total time for all permutations divided by $n!$:

$$\frac{\Omega(n!\lg(n!))}{n!} = \Omega(\lg(n!)) = \Omega(n \lg n) .$$

f. We will show how to modify a randomized decision tree (algorithm) to define a deterministic decision tree (algorithm) that is at least as good as the randomized one in terms of the average number of comparisons.

At each randomized node, pick the child with the smallest subtree (the subtree with the smallest average number of comparisons on a path to a leaf). Delete all the other children of the randomized node and splice out the randomized node itself.

The deterministic algorithm corresponding to this modified tree still works, because the randomized algorithm worked no matter which path was taken from each randomized node.

The average number of comparisons for the modified algorithm is no larger than the average number for the original randomized tree, since we discarded the higher-average subtrees in each case. In particular, each time we splice out a randomized node, we leave the overall average less than or equal to what it was, because

- the same set of input permutations reaches the modified subtree as before, but those inputs are handled in less than or equal to average time than before, and
- the rest of the tree is unmodified.

The randomized algorithm thus takes at least as much time on average as the corresponding deterministic one. (We've shown that the expected running time for a deterministic comparison sort is $\Omega(n \lg n)$, hence the expected time for a randomized comparison sort is also $\Omega(n \lg n)$.)

9-1

9-1 Largest i numbers in sorted order

Given a set of n numbers, we wish to find the i largest in sorted order using a comparison-based algorithm. Find the algorithm that implements each of the following methods with the best asymptotic worst-case running time, and analyze the running times of the algorithms in terms of n and i.

a. Sort the numbers, and list the i largest.

b. Build a max-priority queue from the numbers, and call EXTRACT-MAX i times.

c. Use an order-statistic algorithm to find the ith largest number, partition around that number, and sort the i largest numbers.

sol)

We assume that the numbers start out in an array.

a. Sort the numbers using merge sort or heapsort, which take $\Theta(n \lg n)$ worst-case time. (Don't use quicksort or insertion sort, which can take $\Theta(n^2)$ time.) Put the i largest elements (directly accessible in the sorted array) into the output array, taking $\Theta(i)$ time.

Total worst-case running time: $\Theta(n \lg n + i) = \Theta(n \lg n)$ (because $i \leq n$).

b. Implement the priority queue as a heap. Build the heap using BUILD-HEAP, which takes $\Theta(n)$ time, then call HEAP-EXTRACT-MAX i times to get the i largest elements, in $\Theta(i \lg n)$ worst-case time, and store them in reverse order of extraction in the output array. The worst-case extraction time is $\Theta(i \lg n)$ because

- i extractions from a heap with $O(n)$ elements takes $i \cdot O(\lg n) = O(i \lg n)$ time, and
- half of the i extractions are from a heap with $\geq n/2$ elements, so those $i/2$ extractions take $(i/2)\Omega(\lg(n/2)) = \Omega(i \lg n)$ time in the worst case.

Total worst-case running time: $\Theta(n + i \lg n)$.

c. Use the SELECT algorithm of Section 9.3 to find the ith largest number in $\Theta(n)$ time. Partition around that number in $\Theta(n)$ time. Sort the i largest numbers in $\Theta(i \lg i)$ worst-case time (with merge sort or heapsort).

Total worst-case running time: $\Theta(n + i \lg i)$.

Note that method (c) is always asymptotically at least as good as the other two methods, and that method (b) is asymptotically at least as good as (a). (Comparing (c) to (b) is easy, but it is less obvious how to compare (c) and (b) to (a). (c) and (b) are asymptotically at least as good as (a) because n, $i \lg i$, and $i \lg n$ are all $O(n \lg n)$. The sum of two things that are $O(n \lg n)$ is also $O(n \lg n)$.)

11-1

11-1 Longest-probe bound for hashing

Suppose that we use an open-addressed hash table of size m to store $n \leq m/2$ items.

a. Assuming uniform hashing, show that for $i = 1, 2, \ldots, n$, the probability is at most 2^{-k} that the ith insertion requires strictly more than k probes.

b. Show that for $i = 1, 2, \ldots, n$, the probability is $O(1/n^2)$ that the ith insertion requires more than $2 \lg n$ probes.

Let the random variable X_i denote the number of probes required by the ith insertion. You have shown in part (b) that $\Pr\{X_i > 2 \lg n\} = O(1/n^2)$. Let the random variable $X = \max_{1 \leq i \leq n} X_i$ denote the maximum number of probes required by any of the n insertions.

c. Show that $\Pr\{X > 2 \lg n\} = O(1/n)$.

d. Show that the expected length $\mathrm{E}[X]$ of the longest probe sequence is $O(\lg n)$.

sol)

a. Since we assume independent uniform permutation hashing, we can use the same observation as is used in Corollary 11.7: that inserting a key entails an unsuccessful search followed by placing the key into the first empty slot found. As in the proof of Theorem 11.6, if we let X be the random variable denoting the number of probes in an unsuccessful search, then $\Pr\{X \geq i\} \leq \alpha^{i-1}$. Since $n \leq m/2$, we have $\alpha \leq 1/2$. Letting $i = p + 1$, we have $\Pr\{X > p\} = \Pr\{X \geq p + 1\} \leq (1/2)^{(p+1)-1} = 2^{-p}$.

b. Substituting $p = 2\lg n$ into the statement of part (a) yields that the probability that the ith insertion requires more than $p = 2\lg n$ probes is at most $2^{-2\lg n} = (2^{\lg n})^{-2} = n^{-2} = 1/n^2$.

We must deal with the possibility that $2\lg n$ is not an integer, however. Then the event that the ith insertion requires more than $2\lg n$ probes is the same as the event that the ith insertion requires more than $\lfloor 2\lg n \rfloor$ probes. Since $\lfloor 2\lg n \rfloor > 2\lg n - 1$, we have that the probability of this event is at most $2^{-\lfloor 2\lg n \rfloor} < 2^{-(2\lg n - 1)} = 2/n^2 = O(1/n^2)$.

c. Let the event A be $X > 2\lg n$, and for $i = 1, 2, \ldots, n$, let the event A_i be $X_i > 2\lg n$. In part (b), we showed that $\Pr\{A_i\} = O(1/n^2)$ for $i = 1, 2, \ldots, n$. From how we defined these events, $A = A_1 \cup A_2 \cup \cdots \cup A_n$. Using Boole's inequality, (C.21), we have

$$\begin{aligned}
\Pr\{A\} &\leq \Pr\{A_1\} + \Pr\{A_2\} + \cdots + \Pr\{A_n\} \\
&\leq n \cdot O(1/n^2) \\
&= O(1/n) \, .
\end{aligned}$$

d. We use the definition of expectation and break the sum into two parts:

$$\begin{aligned}
\mathrm{E}[X] &= \sum_{k=1}^{n} k \cdot \Pr\{X = k\} \\
&= \sum_{k=1}^{\lceil 2\lg n \rceil} k \cdot \Pr\{X = k\} + \sum_{k=\lceil 2\lg n \rceil + 1}^{n} k \cdot \Pr\{X = k\} \\
&\leq \sum_{k=1}^{\lceil 2\lg n \rceil} \lceil 2\lg n \rceil \cdot \Pr\{X = k\} + \sum_{k=\lceil 2\lg n \rceil + 1}^{n} n \cdot \Pr\{X = k\} \\
&= \lceil 2\lg n \rceil \sum_{k=1}^{\lceil 2\lg n \rceil} \Pr\{X = k\} + n \sum_{k=\lceil 2\lg n \rceil + 1}^{n} \Pr\{X = k\} \, .
\end{aligned}$$

Since X takes on exactly one value, we have that $\sum_{k=1}^{\lceil 2\lg n \rceil} \Pr\{X = k\} = \Pr\{X \leq \lceil 2\lg n \rceil\} \leq 1$ and $\sum_{k=\lceil 2\lg n \rceil + 1}^{n} \Pr\{X = k\} \leq \Pr\{X > 2\lg n\} = O(1/n)$, by part (c). Therefore,

$$\begin{aligned}
\mathrm{E}[X] &\leq \lceil 2\lg n \rceil \cdot 1 + n \cdot O(1/n) \\
&= \lceil 2\lg n \rceil + O(1) \\
&= O(\lg n) \, .
\end{aligned}$$

11-2

11-2 Slot-size bound for chaining

Suppose that we have a hash table with n slots, with collisions resolved by chaining, and suppose that n keys are inserted into the table. Each key is equally likely to be hashed to each slot. Let M be the maximum number of keys in any slot after all the keys have been inserted. Your mission is to prove an $O(\lg n / \lg \lg n)$ upper bound on $\mathrm{E}[M]$, the expected value of M.

a. Argue that the probability Q_k that exactly k keys hash to a particular slot is given by

$$Q_k = \left(\frac{1}{n}\right)^k \left(1 - \frac{1}{n}\right)^{n-k} \binom{n}{k} .$$

b. Let P_k be the probability that $M = k$, that is, the probability that the slot containing the most keys contains k keys. Show that $P_k \le nQ_k$.

c. Use Stirling's approximation, equation (3.18), to show that $Q_k < e^k / k^k$.

d. Show that there exists a constant $c > 1$ such that $Q_{k_0} < 1/n^3$ for $k_0 = c \lg n / \lg \lg n$. Conclude that $P_k < 1/n^2$ for $k \ge k_0 = c \lg n / \lg \lg n$.

e. Argue that

$$\mathrm{E}[M] \le \Pr\left\{M > \frac{c \lg n}{\lg \lg n}\right\} \cdot n + \Pr\left\{M \le \frac{c \lg n}{\lg \lg n}\right\} \cdot \frac{c \lg n}{\lg \lg n} .$$

Conclude that $\mathrm{E}[M] = O(\lg n / \lg \lg n)$.

sol)

a. A particular key is hashed to a particular slot with probability $1/n$. Suppose we select a specific set of k keys. The probability that these k keys are inserted into the slot in question and that all other keys are inserted elsewhere is

$$\left(\frac{1}{n}\right)^k \left(1 - \frac{1}{n}\right)^{n-k} .$$

Since there are $\binom{n}{k}$ ways to choose our k keys, we get

$$Q_k = \left(\frac{1}{n}\right)^k \left(1 - \frac{1}{n}\right)^{n-k} \binom{n}{k} .$$

b. For $i = 1, 2, \ldots, n$, let X_i be a random variable denoting the number of keys that hash to slot i, and let A_i be the event that $X_i = k$, i.e., that exactly k keys hash to slot i. From part (a), we have $\Pr\{A\} = Q_k$. Then,

$$\begin{aligned}
P_k &= \Pr\{M = k\} \\
&= \Pr\left\{\left(\max_{1 \le i \le n} X_i\right) = k\right\} \\
&= \Pr\{\text{there exists } i \text{ such that } X_i = k \text{ and that } X_i \le k \text{ for } i = 1, 2, \ldots, n\} \\
&\le \Pr\{\text{there exists } i \text{ such that } X_i = k\} \\
&= \Pr\{A_1 \cup A_2 \cup \cdots \cup A_n\} \\
&\le \Pr\{A_1\} + \Pr\{A_2\} + \cdots + \Pr\{A_n\} \qquad \text{(by inequality (C.19))} \\
&= nQ_k .
\end{aligned}$$

c. We start by showing two facts. First, $1 - 1/n < 1$, which implies $(1 - 1/n)^{n-k} < 1$. Second, $n!/(n-k)! = n \cdot (n-1) \cdot (n-2) \cdots (n-k+1) < n^k$. Using these facts, along with the simplification $k! > (k/e)^k$ of equation (3.18), we have

$$\begin{aligned}
Q_k &= \left(\frac{1}{n}\right)^k \left(1 - \frac{1}{n}\right)^{n-k} \frac{n!}{k!(n-k)!} \\
&< \frac{n!}{n^k k!(n-k)!} && ((1 - 1/n)^{n-k} < 1) \\
&< \frac{1}{k!} && (n!/(n-k)! < n^k) \\
&< \frac{e^k}{k^k} && (k! > (k/e)^k) .
\end{aligned}$$

d. Notice that when $n = 2$, $\lg \lg n = 0$, so to be precise, we need to assume that $n \ge 3$.

In part (c), we showed that $Q_k < e^k/k^k$ for any k; in particular, this inequality holds for k_0. Thus, it suffices to show that $e^{k_0}/{k_0}^{k_0} < 1/n^3$ or, equivalently, that $n^3 < {k_0}^{k_0}/e^{k_0}$.

Taking logarithms of both sides gives an equivalent condition:

$$\begin{aligned}
3 \lg n &< k_0(\lg k_0 - \lg e) \\
&= \frac{c \lg n}{\lg \lg n}(\lg c + \lg \lg n - \lg \lg \lg n - \lg e) .
\end{aligned}$$

Dividing both sides by $\lg n$ gives the condition

$$\begin{aligned}
3 &< \frac{c}{\lg \lg n}(\lg c + \lg \lg n - \lg \lg \lg n - \lg e) \\
&= c\left(1 + \frac{\lg c - \lg e}{\lg \lg n} - \frac{\lg \lg \lg n}{\lg \lg n}\right) .
\end{aligned}$$

Let x be the last expression in parentheses:

$$x = \left(1 + \frac{\lg c - \lg e}{\lg\lg n} - \frac{\lg\lg\lg n}{\lg\lg n}\right) .$$

We need to show that there exists a constant $c > 1$ such that $3 < cx$.

Noting that $\lim_{n\to\infty} x = 1$, we see that there exists n_0 such that $x \geq 1/2$ for all $n \geq n_0$. Thus, any constant $c > 6$ works for $n \geq n_0$.

We handle smaller values of n—in particular, $3 \leq n < n_0$—as follows. Since n is constrained to be an integer, there are a finite number of n in the range $3 \leq n < n_0$. We can evaluate the expression x for each such value of n and determine a value of c for which $3 < cx$ for all values of n. The final value of c that we use is the larger of

- 6, which works for all $n \geq n_0$, and
- $\max_{3\leq n<n_0} \{c : 3 < cx\}$, i.e., the largest value of c that we chose for the range $3 \leq n < n_0$.

Thus, we have shown that $Q_{k_0} < 1/n^3$, as desired.

To see that $P_k < 1/n^2$ for $k \geq k_0$, we observe that by part (b), $P_k \leq nQ_k$ for all k. Choosing $k = k_0$ gives $P_{k_0} \leq nQ_{k_0} < n \cdot (1/n^3) = 1/n^2$. For $k > k_0$, we will show that we can pick the constant c such that $Q_k < 1/n^3$ for all $k \geq k_0$, and thus conclude that $P_k < 1/n^2$ for all $k \geq k_0$.

To pick c as required, we let c be large enough that $k_0 > 3 > e$. Then $e/k < 1$ for all $k \geq k_0$, and so e^k/k^k decreases as k increases. Thus,

$$\begin{aligned} Q_k &< e^k/k^k \\ &\leq e^{k_0}/k^{k_0} \\ &< 1/n^3 \end{aligned}$$

for $k \geq k_0$.

e. The expectation of M is

$$\begin{aligned} \mathrm{E}[M] &= \sum_{k=0}^{n} k \cdot \Pr\{M = k\} \\ &= \sum_{k=0}^{k_0} k \cdot \Pr\{M = k\} + \sum_{k=k_0+1}^{n} k \cdot \Pr\{M = k\} \\ &\leq \sum_{k=0}^{k_0} k_0 \cdot \Pr\{M = k\} + \sum_{k=k_0+1}^{n} n \cdot \Pr\{M = k\} \\ &\leq k_0 \sum_{k=0}^{k_0} \Pr\{M = k\} + n \sum_{k=k_0+1}^{n} \Pr\{M = k\} \\ &= k_0 \cdot \Pr\{M \leq k_0\} + n \cdot \Pr\{M > k_0\} , \end{aligned}$$

which is what we needed to show, since $k_0 = c \lg n / \lg\lg n$.

To show that $\mathrm{E}[M] = O(\lg n / \lg\lg n)$, note that $\Pr\{M \leq k_0\} \leq 1$ and

$$
\begin{aligned}
\Pr\{M > k_0\} &= \sum_{k=k_0+1}^{n} \Pr\{M = k\} \\
&= \sum_{k=k_0+1}^{n} P_k \\
&< \sum_{k=k_0+1}^{n} 1/n^2 \qquad \text{(by part (d))} \\
&< n \cdot (1/n^2) \\
&= 1/n \, .
\end{aligned}
$$

We conclude that

$$
\begin{aligned}
\mathrm{E}[M] &\leq k_0 \cdot 1 + n \cdot (1/n) \\
&= k_0 + 1 \\
&= O(\lg n / \lg \lg n) \, .
\end{aligned}
$$