Collaborators

Ben Nelson, Corbin Baldwin and I collaborated for this assignment.

Question 1: Balls and bins . . . . **[12]**

Consider the process we saw in class, of distributing n balls into n bins, independently and uniformly at random.

(a) [**4**] What is the expected number of empty bins?

(b) [**2**] Show that the probability that 80% of the bins are empty is $\leq 1/2$.

(c) [**2**] Let X_i be the random variable that is 1 if bin i is empty and is 0 otherwise. Are X_1 and X_2 independent? Give an intuitive reasoning.

(d) [**4**] Let Z be the random variable defined by $Z = \sum_i X_i$ (i.e., the number of empty bins). Suppose you are told that the variance of Z is $\leq n$. Use this to prove that the probability that 80% of the bins are empty is $< 8/n$. [Note that this is a much better bound than the one in part (b) when n is large.]

(a)

$$\text{Let random variable } X_i = \begin{cases} 1, & \text{when slot } i \text{ is empty} \\ 0, & \text{otherwise} \end{cases}$$

$$\text{Number of empty bins} = \sum_{i=1}^{n} X_i$$

$$\begin{aligned}
\mathbb{E}\left[\text{Number of empty bins}\right] &= \mathbb{E}\left[\sum_{i=1}^{n} X_i\right] \\
&= \sum_{i=1}^{n} \mathbb{E}\left[X_i\right], \text{ due to linearity of expectation} \\
&= \sum_{i=1}^{n} \left(1 \times \Pr\left[X_i = 1\right] + 0 \times \Pr\left[X_i = 0\right]\right) \\
&= \sum_{i=1}^{n} \Pr\left[X_i = 1\right] \\
&= \sum_{i=1}^{n} \frac{(n-1)^n}{n^n}
\end{aligned}$$

since n balls can be put into $n-1$ slots in $(n-1)^n$ ways and n balls can be put into n slots in n^ndifferent ways

$$\begin{aligned}
&= n\frac{(n-1)^n}{n^n} \\
&= \frac{(n-1)^n}{n^{n-1}}
\end{aligned}$$

(b)

$$\begin{aligned}
\Pr\left[80\% \text{ of the bins are empty}\right] &= \Pr\left[\text{Number of empty bins } \geq 0.8n\right] \\
&= \Pr\left[X_e \geq 0.8n\right], \text{ where } X_e = \text{ number of empty bins} \\
&= \Pr\left[X_e \geq \frac{0.8n}{\mathbb{E}\left[X_e\right]}\mathbb{E}\left[X_e\right]\right] \\
&\leq \frac{\mathbb{E}\left[X_e\right]}{0.8n}, \text{ by Markov's Inequality} \\
&= \frac{(n-1)^n}{n^{n-1}}\frac{1}{0.8n} \\
&= \frac{1.25(n-1)^n}{n^n} \\
\text{Let } y &= \frac{1.25(n-1)^n}{n^n} \\
&= 1.25(n-1)^n n^{-n} \\
\frac{dy}{dn} &= 1.25n(n-1)^{n-1}n^{-n} - 1.25n(n-1)^n n^{-n-1} \\
&= 1.25n(n-1)^{n-1}n^{-n}\left(1-(n-1)n^{-1}\right) \\
&= 1.25n(n-1)^{n-1}n^{-n}\left(1-\left(1-\frac{1}{n}\right)\right) \\
&= 1.25n(n-1)^{n-1}n^{-n}\frac{1}{n} \\
&\geq 0, \text{ for } n > 1 \\
\lim_{n\to\infty}\frac{1.25(n-1)^n}{n^n} &= \lim_{n\to\infty} 1.25\left(\frac{n-1}{n}\right)^n \\
&= 1.25\lim_{n\to\infty}\left(1-\frac{1}{n}\right)^n \\
&= 1.25\frac{1}{e} \\
&= 0.4598 \\
&< \frac{1}{2}
\end{aligned}$$

Since $\frac{dy}{dn} > 0$ for $n > 1$ and $\lim_{n\to\infty} y < \frac{1}{2}$, $\Pr\left[80\% \text{ of the bins are empty}\right] < \frac{1}{2}$.

(c) If bin 1 is known to be empty, then there is a higher chance that bin 2 is not empty, since all of the n balls would have been assigned to the $n-1$ remaining buckets. This means that X_1 and X_2 are not independent random variables.

(d)

$$
\begin{aligned}
\mathbb{E}\left[Z\right] &= \frac{(n-1)^n}{n^{n-1}}, \text{ from part a above} \\
&= n\frac{(n-1)^n}{n^n} \\
&= n\left(1-\frac{1}{n}\right)^n \\
&< \frac{n}{e}, \text{ since } \lim_{n\to\infty}\left(1-\frac{1}{n}\right)^n = \frac{1}{e} \\
\Pr\left[80\% \text{ of the bins are empty}\right] &= \Pr\left[\text{Number of empty bins } \geq 0.8n\right] \\
&= \Pr\left[Z \geq 0.8n\right] \\
&= \Pr\left[Z - \mathbb{E}\left[Z\right] \geq 0.8n - \mathbb{E}\left[Z\right]\right] \\
&= \Pr\left[Z - \mathbb{E}\left[Z\right] \geq \frac{0.8n - \mathbb{E}\left[Z\right]}{\sigma}\sigma\right], \text{ where } \sigma^2 \text{ is the variance of } Z \\
&\leq \frac{1}{\left(\frac{0.8n-\mathbb{E}[Z]}{\sigma}\right)^2}, \text{ by Chebychev's inequality} \\
&= \frac{\sigma^2}{\left(0.8n - \mathbb{E}\left[Z\right]\right)^2} \\
&\leq \frac{n}{\left(0.8n - \mathbb{E}\left[Z\right]\right)^2} \\
&\leq \frac{n}{\left(0.8n - \frac{n}{e}\right)^2} \\
&= \frac{ne^2}{n^2\left(0.8e-1\right)^2} \\
&= \frac{1}{n}\frac{e^2}{\left(0.8e-1\right)^2} \\
&= \frac{5.355}{n} \\
&< \frac{8}{n} \quad \square
\end{aligned}
$$

Question 2: Median/order finding... **[10]**

Consider the following simple algorithm for finding the kth smallest element in a given unsorted array $A[1], A[2], \dots, A[n]$ (assume that all the $A[i]$ are distinct):

1. pick a uniformly random index $i \in \{1, 2, \dots, n\}$
2. divide the array $A[]$ into two sub-arrays – $B[]$ and $C[]$, where the elements of $B[]$ are all $\leq A[i]$ and the elements of $C[]$ are $> A[i]$.
3. if $\text{length}(B) \geq k$, we recursively find the kth smallest element in $B[]$; else we find the $k-\text{length}(B)$'th smallest element in $C[]$.

(a) **[2]** Prove that the running time on the array A of length n can be bounded as

$$T(n) \leq \max\{T(\text{length}(B)), T(\text{length}(C))\} + O(n).$$

(b) **[3]** Give an input A, k, and an unlucky choice of indices i that leads to a running time larger than $n^2/4$.

(c) **[5]** Let $f(n)$ denote the *expected running time* on an input array of length n. Derive a "probabilistic recurrence" analogous to the one we saw in class for quicksort, and show that $f(n) = O(n)$. [*Hint:* use part (a).]

(a) To divide the array into two sub-arrays, we would need to compare each element with the one corresponding to the position of the randomly chosen index. This will take $O(n)$ time. In the worst case, the element corresponding to the k^{th} smallest element in the original array, would lie in the larger of the two sub-arrays. This means that the running time on the array A of length n can be bounded as $T(n) \leq \max\{T(\text{length}(B)), T(\text{length}(C))\} + O(n)$.

(b) Let $k = 7$ and $A = \{6, 2, 5, 4, 3, 1, 9\}$
$B = \{1\}$ and $C = \{6, 2, 5, 4, 3, 9\}$ when $i = 6$
Since length(B)$= 1 < k$, we need to find the $7 - 1 = 6^{th}$ smallest element in C
$\{2\}$ and $\{6, 5, 4, 3, 9\}$ are the two sub-arrays when $i = 2$
Since length(first sub-array)$= 1 < 6$, we need to find the $6 - 1 = 5^{th}$ smallest element in the second sub-array
$\{3\}$ and $\{6, 5, 4, 9\}$ are the two sub-arrays when $i = 4$
Since length(first sub-array)$= 1 < 5$, we need to find the $5 - 1 = 4^{th}$ smallest element in the second sub-array
$\{4\}$ and $\{6, 5, 9\}$ are the two sub-arrays when $i = 3$
Since length(first sub-array)$= 1 < 4$, we need to find the $4 - 1 = 3^{rd}$ smallest element in the second sub-array
$\{5\}$ and $\{6, 9\}$ are the two sub-arrays when $i = 2$
Since length(first sub-array)$= 1 < 3$, we need to find the $3 - 1 = 2^{nd}$ smallest element in the second sub-array
$\{6\}$ and $\{9\}$ are the two sub-arrays when $i = 1$
Since length(first sub-array)$= 1 < 2$, we need to find the $2 - 1 = 1^{st}$ smallest element in the second sub-array
We get 9 as the answer, which is the $k = 7^{th}$ smallest element in the original array $A = \{6, 2, 5, 4, 3, 1, 9\}$

If we add up the lengths of the arrays worked on, each in $O(n)$ time, we get $7 + 6 + 5 + 4 + 3 + 2 = 27 > \frac{n^2}{4} = \frac{7^2}{4} = \frac{49}{4} = 12.25$.

(c) Consider the base case of an array of length 0. $T(1) = O(1)$ since it takes constant time to kind the 1^{st} smallest element in the array. Since the recurrence is bounded as

$$T(n) \leq \max\{T(\text{length}(B)), T(\text{length}(C))\} + O(n).$$

Consider the case where array B has a length of $n - r$ and array C has a length on r. The recurrence will be

$$T(n) \leq \max\{T(n - r), T(r)\} + O(n).$$

$$\begin{aligned}
\mathbb{E}\left[T(n)\right] &\leq \sum_{r=0}^{n} \Pr\left[r\right]\left(\max\{T(n-r), T(r)\} + O(n)\right), (r \text{ starts at 0 when largest element is chosen})\\
&= \sum_{r=0}^{n} \frac{1}{n}\left(\max\{T(n-r), T(r)\} + O(n)\right), \text{ since the probability of any } r \text{ is } \frac{1}{n}\\
&= \sum_{r=0}^{n} \frac{1}{n}\left(\max\{O(n-r), T(r)\} + O(n)\right), \text{ assuming } T(n-r) = O(n-r) \text{ and } T(r) = O(r)\\
&= \sum_{r=0}^{n} \frac{1}{n}\left(\max\{(n-r), r\} + n\right), \text{ taking the constant to be 1}\\
&= \frac{1}{n}\left(2\left(\frac{n(n+1)}{2} - \frac{\frac{n}{2}(\frac{n}{2}+1)}{2}\right) + n^2\right), \text{ as we take the larger sub-array for worst case}\\
&= \frac{1}{n}\left(n^2 + n - \frac{n^2}{4} - \frac{n}{2} + n^2\right)\\
&= \frac{1}{n}\left(\frac{7}{4}n^2 + \frac{n}{2}\right)\\
&= \frac{7}{4}n + \frac{1}{2}\\
&= O(n) \quad \square
\end{aligned}$$

Question 3: Quicksort revisited .... **[6]**

Recall the quicksort procedure we saw in class, where we pick a random element of the input array as the *pivot*, recursively sort the elements less than and larger than the pivot, and concatenate the solutions. We saw that the expected running time is $O(n \log n)$.

(a) [**2**] Using a basic implementation (base case being a singleton), find the *constant* in the $O()$ notation for the algorithm above. (You may do this by picking any array, repeatedly running the procedure above, and averaging the values of running time divided by $n \log n$.)

(b) [**2**] Now, consider the following procedure: for $k = 1, 2, 3$, first pick $(2k+1)$ random indices, and choose their *median* as the pivot. Now report the constant in the $O()$ notation.

(c) [**2**] Explain your observations intuitively.

(a) The constant that was computed for various random integer array sizes for runtimes averaged over 20 iterations is shown below

Array Size	Constant
10^5	5.21×10^{-6}
10^6	3.47×10^{-6}
10^7	2.74×10^{-6}
10^8	2.69×10^{-6}

(b) The value of the constant for various array sizes is shown below

Array Size	$k = 1$	$k = 2$	$k = 3$
10^5	6.08×10^{-6}	6.08×10^{-6}	6.95×10^{-6}
10^6	4.85×10^{-6}	5.14×10^{-6}	5.57×10^{-6}
10^7	4.20×10^{-6}	4.52×10^{-6}	4.98×10^{-6}
10^8	3.63×10^{-6}	4.17×10^{-6}	4.43×10^{-6}

(c) The value of the constant is least for the case of the single random pivot and higher for the cases where a median of random elements is used as a pivot. It looks like the work involved in computing the median of elements negated the expected boost in performance from choosing a supposedly better pivot. The more the elements that a pivot is chosen from, the higher the value of the constant and the run time. The best performance was from choosing a single random pivot as it involved minimum extra work and performance was good as it was improbable that many pivots would be bad and adversely impact run time.

Question 4: Checking matrix multiplication .... **[12]**

Matrix multiplication is one of the classic algorithmic problems. Consider the problem of multiplying two $n \times n$ matrices over the field $\mathbf{F}_2$ (i.e., we have matrices with entries 0/1, and we perform all computations modulo 2; e.g., 0*0 + 1*1 + 1*1 + 1*0 = 0).

The best known algorithms here are messy and take time $O(n^{2.36\cdots})$. However, the point of this exercise is to prove a simpler statement. Suppose someone gives a matrix C and claims that $C = AB$, can we *quickly verify* if the claim is true?

(a) [**4**] First prove a warm-up statement: suppose a and b are any two 0/1 vectors of length n, and suppose that $a \neq b$. Then, for a random binary vector $x \in \{0,1\}^n$ (one in which each coordinate is chosen uniformly at random), prove that $\Pr[\langle a, x\rangle = \langle b, x\rangle \pmod 2] = 1/2$.

(b) [**6**] Now, design an $O(n^2)$ time algorithm that tests if $C = AB$ and has a success probability $\geq 1/2$. (You need to prove the probability bound.)

(c) [**2**] Show how to improve the success probability to 7/8 while still having running time $O(n^2)$.

Question 5: Coloring graphs .... **[10]**

One of the popular problems in graph theory is that of graph coloring: we are given an undirected graph $G = (V, E)$, and the aim is to assign each vertex u a *color* $c(u)$, with the guarantee that for any edge u, v, $c(u) \neq c(v)$. (i.e., end-points of edges must receive different colors.) The goal is to color the graph using the fewest total number of colors, subject to the above condition.

Coloring is known to be a very hard problem. Suppose we are OK with something weaker: suppose that we consider an assignment of colors *acceptable* if $c(u) \neq c(v)$ holds for $\geq 90\%$ of the edges.

(a) [**7**] Suppose that we *randomly* assign a color in the range $\{1, 2, \ldots, 20\}$ to the vertices. Prove that we obtain an acceptable assignment with probability $\geq 1/2$. (Note that the algorithm is quite remarkable – it doesn't even look at the edges of the graph!)

(b) [**3**] What happens above when we randomly assign colors in the range $\{1, 2, \ldots, 11\}$? Can we still obtain an algorithm that succeeds with probability $\geq 1/2$?

(a)

$$\text{Let random variable } E_i = \begin{cases} 1, & \text{when vertices at the two ends of edge } E_i \text{ are of different colors} \\ 0, & \text{otherwise} \end{cases}$$

Let E_T be the number of edges where vertices at the two ends are of different colors

$$\begin{aligned} \mathbb{E}\left[E_T\right] &= \mathbb{E}\left[\sum_{i=1}^{n} E_i\right], \text{ where } n \text{ is the number of vertices} \\ &= \sum_{i=1}^{n} \mathbb{E}\left[E_i\right], \text{ by linearity of Expectation} \\ &= \sum_{i=1}^{n} \left(1 \times \Pr\left[E_i = 1\right] + 0 \times \Pr\left[E_i = 0\right]\right) \\ &= \sum_{i=1}^{n} \Pr\left[E_i = 1\right] \\ &= \sum_{i=1}^{n} \frac{19}{20}, \text{ as once a color is assigned to a vertex, one of 19 can be assigned to the other} \\ &= \frac{19}{20} n \\ &= 0.95n \end{aligned}$$

Since the expected value or mean of the number of edges where vertices at the two ends are of different colors is $0.95n$, the probability that the number of edges where vertices at the two ends are of different colors is $\geq 0.95n$ is $\frac{1}{2}$. This means that the probability that the number of edges where vertices at the two ends are of different colors is the $\geq 0.90n$ is definitely $\geq \frac{1}{2}$ since this covers more than half the probability mass.

(b) In the case where we randomly assign colors in the range $\{1, 2, \ldots, 11\}$, $\mathbb{E}\left[E_T\right] = \frac{10}{11}n = 0.91n$ based on a computation similar to the one shown above. So the the probability that the number of edges where vertices at the two ends are of different colors is the $\geq 0.90n$ is definitely $\geq \frac{1}{2}$ since this covers more than half the probability mass.

References

[1] "Freivalds' Algorithm." *Wikipedia*, Wikimedia Foundation, 30 Sept. 2018, `https://en.wikipedia.org/wiki/Freivalds\%27\_algorithm`.