Notes CS:5360 Randomized Algorithms
Week 3: Sep 4, 2018
Scribe: D M Hasibul Hasan

1 Deterministic QuickSort

Here is the usual deterministic version of quickSort.

Algorithm 1 Deterministic QuickSort

```
1: function QuickSort(L[1...n])
2:     if n ≤ c then
3:         BubbleSort(L)
4:         return L
5:     else
6:         p ←index chosen arbitrarily from [1...n]
7:         L_1, L_2 ← ∅
8:         for i ← 1 to n do
9:             if L[i] < L[p] then
10:                L_1 ← L_1 · L[i]
11:            else if L[i] > L[p] then
12:                L_2 ← L_2 · L[i]
13:            end if
14:        end for
15:        QuickSort(L_1)
16:        QuickSort(L_2)
17:        return L_1 · L[p] · L_2
18:    end if
19: end function
```

2 A Non-Standard Randomized Version of QuickSort

Here we consider a randomized version of the quickSort algorithm that is usually not discussed in classes. Replace Steps (6)-(14) from above algorithm by,

$$(L_1, L_2) \leftarrow \text{randomizedParitionLV}(L).$$

Since randomizedPartitionLV is a Las Vegas algorithm, when we exit this algorithm we are guaranteed to get a partition (L_1, L_2) that satisfies $|L|/3 \leq |L_1| \leq 2|L|/3$. Let $T(L)$ be the random variable that denotes the running time of randomizedQuickSort on input L. Let $\overline{T}(n)$ denote the worst case expected running time of randomizedQuickSort on input of size n. We get the following recurrence:

$$T(L) \leq T(L_1) + T(L_2) + R(L)$$

where $R(L)$ is the random variable denoting the running time of randomizedParitionLV on input L. By taking expectation on both sides we get,

$$E[T(L)] \leq E[T[L_1] + T[L_2] + R[L]].$$

By linearity of expectation we get

$$E[T[L]] \leq E[T[L_1]] + E[T[L_2]] + E[R[L]].$$

Therefore,

$$\overline{T}(n) \leq \overline{T}(\lceil n/3 \rceil) + \overline{T}(\lceil 2n/3 \rceil) + O(n).$$

Note that this uses the fact that earlier we showed that $E[R[L]] = O(n)$ for any list L of length n. Solving this recurrence gives us $\overline{T}(n) = O(n \log n)$.

3 The Standard Randomized Version of QuickSort

The "standard" version of randomized quickSort is one in which Step (6) is replaced by a randomized step:

$$p \leftarrow \text{ index chosen } \textit{uniformly} \text{ at random from } [1...n].$$

It is easy to see that the running time of this algorithm is proportional to the number of comparisons made by the algorithm. So we focus on counting the number of comparisons and show the following bound on the expected number of comparisons made by the algorithm.

Theorem 1. *The expected number of comparison made by* RANDOMIZEDQUICKSORT *is* $2n \ln n + \Theta(n)$.

Proof. Let $L = x_1, x_2, \ldots, x_n$ and let $y_1, y_2, \ldots, y_n$ be a sorted version of $L = x_1, x_2, \ldots, x_n$. Also let, X be the random variable denoting the total number of comparisons. So our goal is to upper bound $E[x]$. Let X_ij be the random variable denoting the number of comparisons of y_i and y_j. Note that $X_{ij} \in \{0, 1\}$. X_{ij} cannot be larger than 1 because when the algorithm compares y_i and y_j, it does so because one of them is the pivot. The pivot will not participate in any further comparisons. Further, note that

$$X = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} x_{ij}.$$

Therefore by linearity of expectation,

$$E[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} E[X_{ij}].$$

Since $X_{ij} \in \{0, 1\}$, $E[X_{ij}] = Pr(X_{ij} = 1)$. To calculate $Pr(X_{ij} = 1)$, note that y_i and y_j are compared iff the first element chosen as pivot from the set $Y^{ij} = \{y_i, y_{i+1}, \ldots, y_j\}$ is either y_i or y_j. Since this happens with probability $2/(j - i + 1)$, we get $Pr(X_{ij} = 1) = 2/(j - i + 1)$. Therefore,

$$E[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j - i + 1}.$$

To simplify the expression we do a "change of variable" and let $k = j - i + 1$. Then,

$$E[X] = \sum_{i=1}^{n-1} \sum_{k=2}^{n-i+1} \frac{2}{k}.$$

We now want to switch the two summations and for this we need to understand all pairs of values that i and k can take in this summation. The table below shows this.

After exchanging the position of i and k we get,

$$\begin{aligned}
E[X] &= \sum_{k=2}^{n} \sum_{i=1}^{n-k+1} \frac{2}{k}. \\
&= \sum_{k=2}^{n} \frac{2}{k} \cdot (n - k + 1) \\
&= 2n \sum_{k=2}^{n} \frac{1}{k} - 2(n-1) + 2 \sum_{k=2}^{n} \frac{1}{k} \\
&= 2n(H_n - 1) - 2(n-1) + 2(H_n - 1) \\
&= 2n(\ln n + \Theta(1)) - 2n - 2n + 2(\ln n + \Theta(1)) \\
&= 2n \ln \Theta + \Theta(n)
\end{aligned}$$

□

Table 1: Shows pairs of feasible values of i and k in the summation above.

i	$k=2$	3	4	.	.	.	n
1	✓	✓	✓	.	.	.	✓
2	✓	✓	✓	.	.	.	×
3	✓	✓	✓	.	.	×	×
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
$n-1$	✓	×	×	.	.	×	×

As an aside it is worth noting that a randomized algorithm very similar to the one analyzed above, works for the SELECTION problem.
SELECTION
INPUT: A list $L[1 \dots n]$ and an index $k \in [1 \dots n]$
OUTPUT: k^{th} smallest element

It can be shown (see Homework 2) that SELECTION can be solved in expected $O(n)$time using a randomized algorithm similar to the randomized quickSort algorithm described above. The key change we need to make to the algorithm is simple. After we have partitioned L into L_1, $L[p]$, and L_2, we first check if $L[p]$ has rank k. Otherwise, we recurse into either L_1 or L_2 (but not both) with an appropriate value of k.

4 The Union Bound and Some Applications

Let $A_1, A_2, \dots, A_n$ be arbitrary events. Then,

$$P_r(\bigcup_{i=1}^{n} A_i) \leq \sum_{i=1}^{n} P_r(A_i).$$

The typical usage of the union bound in the analysis of randomized algorithms is as follows. Suppose an algorithm makes errors in K different ways, we are interested in bounding the probability that the algorithm makes an error. This is equal to

$$Pr(\bigcup_{i=1}^{K} \text{algo makes an error of type } i).$$

Using union bound, this is upper bounded by,

$$\sum_{i=1}^{K} Pr(\text{algorithm makes error of type } i\ ).$$

Now suppose we have a total budget of "ϵ" for the error probability. Then, one way to meet this budget is to ensure that the probability of error of each of the k types is ϵ/K.

4.1 Monte Carlo RandomizedQuickSort

One example of using the Union Bound in this manner is as follows. Let us consider yet another version of the randomized quickSort algorith in which we call the Monte Carlo version of the randomized partitioning subroutine. So in other words,

$$(L_1, L_2) \leftarrow \text{RANDOMIZEDPARTITIONMC}(L).$$

Recall that this subroutine call runs in $O(nk)$ time and does not return a balanced partition with probability $(2/3)^k$. Each call to RANDOMIZEDPARTITIONMC is prone to error.

Let $L = x_1, x_2, \ldots, x_n$ and let $y_1, y_2, \ldots, y_n$ be the sorted version of L. Note that every call to RANDOMIZEDPARTITIONMC takes as input some contiguous subsequence of $y_1, y_2, \ldots, y_n$. Let us use Y^{ij} to denote $(y_i, y_{i+1}, \ldots, y_j)$. Let B_{ij} be the random variable denoting the event that the call to RANDOMIZEDPARTITIONMC on input Y^{ij} fails. Let B denote the event that quickSort fails. Since the only way quickSort can fail is if textscrandomizedPartitionMC on input Y^{ij} faisl for some $1 \leq i < j \leq n$, we have $B = \bigcup_{i,j} B_{ij}$. By the union bound,

$$Pr(B) \leq \sum_{i,j} P_r(B_{ij}) = \binom{n}{2}(2/3)^k.$$

Suppose we want the error probability to be at most $1/n$ or correctness probability $\geq 1 - 1/n$. We will need to pick k such that

$$\binom{n}{2}(2/3)^k \leq \frac{1}{n}.$$

It will therefore suffice to pick k such that

$$n^2(2/3)^k = \frac{1}{n} \Rightarrow n^3 = (3/2)^k \Rightarrow 3\log_{3/2} n = k.$$

4.2 Balls in Bins Problem

The *Balls in Bins* problem is a simple abstraction of a situation that arises in many settings, e.g., *load balancing*, and *hashing*. In the load balancing applications we have n machines and m jobs. The jobs have been adversarially distributed among the n machines and so a few machines may have been given many jobs to process. We assume that machines can communicate with each other and it is cheaper to redistribute jobs and balance the load before executing the jobs than it is to execute the jobs where they are. The question then is how to quickly redistribute the jobs so as to balance the load on the machines? A simple answer is that each machine i sends each of its jobs to a machine chosen from $\{1, 2, \ldots, n\}$ uniformly at random. To understand how good this scheme is, we need to understand the *maximum load* (i.e., the maximum number of jobs assigned to a machine) after redistribution. In this application, machines are bins and jobs are balls that are thrown into the bins.

To simplify calculations, let $m = n$, i.e., total numner of balls equals total number of bins. Also note that the expected number of balls in a bin is 1. Our goal is to find a function $k(n)$ such that $Pr(maxload > k(n)) < 1/n$.

Analysis: Suppose some bin i has k or more balls in it. Then, for some size-k subset S of balls, all balls in S fall into bin i. For any size-k subset S of balls, let B_S denote the event that all balls in S fall in the same bin. Let B denote the event that some bin has at least k balls. Then $B = \cup_S B_S$, where the union is taken over all size-k subsets S of balls. By the Union Bound,

$$Pr(B) \leq \sum_{S:|S|=k} Pr(B_S).$$

Noting that $Pr(B_S) = 1/n^{k-1}$ and there are $\binom{n}{k}$ size-k subsets we get that

$$\begin{aligned} Pr(B) &\leq \binom{n}{k} n^{-k+1} \\ &= \frac{n!}{(n-k)!k!} \cdot n^{-k+1} \\ &= \frac{n(n-1)(n-3)....(n-k+1)}{k!} \cdot n^{-k+1} \\ &\leq \frac{n^k}{k!} \cdot n^{-k+1} \\ &= \frac{n}{k!}. \end{aligned} \tag{1}$$

We can get a different form of this upper bound by using *Stirling's Formula*:

$$k! \geq \sqrt{2\pi k}\left(\frac{k}{e}\right)^k \geq \left(\frac{k}{e}\right)^k.$$

Therefore,

$$Pr(B) \leq \frac{n}{(k/e)^k} = \frac{n \cdot e^k}{k^k} \tag{2}$$

Now that we have two forms of upper bounds on $Pr(B)$, we can try different values of k (as a function of n) to see if we can drive the probability of B to $1/n$ or less.

1. First, let us try $k = \ln n$ and plug it into (2).

$$\begin{aligned} Pr(B) &= \frac{n^2}{(\ln n)^{\ln n}} \\ &= \frac{n^2}{(e^{\ln \ln n})^{\ln n}} \\ &= \frac{n^2}{(e^{\ln n})^{\ln \ln n}} \\ &= \frac{n^2}{n^{\ln \ln n}} \\ &= \frac{1}{n^{\ln \ln n - 2}} \end{aligned}$$

   Note that this probabilty is quite a bit less than our target of $1/n$. So we could probably try a smaller value of k.

2. So we try $k = \frac{c \ln n}{\ln \ln n}$ for a yet-to-be determined constant c. From Stirling formula, we know that $\ln(k!) \geq k \ln k - k$ Plugging, $k = \frac{c \cdot \ln n}{\ln \ln n}$ into this inequality, we get

$$\begin{aligned} \ln(k!) &\geq \frac{c \ln n}{\ln \ln n} \cdot (\ln c + \ln \ln n - \ln \ln \ln n - 1) \\ &= c \ln n (\frac{\ln c}{\ln \ln n} + 1 - \frac{\ln \ln \ln n}{\ln \ln n} - \frac{1}{\ln \ln n}) \\ &\geq \frac{c \ln n}{2.} \end{aligned}$$

   The last inequality holds when n is larger than a constant. Thus, for n at least some constant, $k! \geq e^{c \ln n/2} = n^{c/2}$. Therefore, using inequality (1),

$$Pr(B) \leq \frac{n}{n^{c/2}} = \frac{1}{n^{c/2-1}}.$$

   Therefore, if we pick $c = 4$, we get $Pr(B) \leq 1/n$.