Topic 4: Quicksort

(CLRS 7)

CPS 230, Fall 2001

1 Quicksort

- We previously saw how divide-and-conquer can be used to design sorting algorithm—Mergesort
  - Partition n elements array A into two subarrays of $n/2$ elements each
  - Sort the two subarrays recursively
  - Merge the two subarrays

  Running time: $T(n) = 2T(n/2) + \Theta(n) \Longrightarrow T(n) = \Theta(n \log n)$

- Another possibility is to use the "opposite" version of divide-and-conquer—Quicksort
  - Partition $A[1..n]$ into subarrays $A' = A[1..q]$ and $A'' = A[q+1..n]$ such that all elements in A'' are larger than all elements in A'.
  - Recursively sort A' and A''.
  - (nothing to combine/merge. A already sorted after sorting A' and A'')

  If $q = n/2$ and we divide in $\Theta(n)$ time, we again get the recurrence $T(n) = 2T(n/2) + \Theta(n)$ for the running time $\Longrightarrow T(n) = \Theta(n \log n)$

  The problem is that it is hard to develop partition algorithm which always divide A in two halves

- Pseudo code for Quicksort:

```
QUICKSORT(A, p, r)
IF p < r THEN
    q = PARTITION(A, p, r)
    QUICKSORT(A, p, q − 1)
    QUICKSORT(A, q + 1, r)
FI
```

  Sort using QUICKSORT$(A, 1, n)$

```
PARTITION(A, p, r)
x = A[r]
i = p − 1
FOR j = p TO r − 1 DO
    IF A[j] ≤ x THEN
        i = i + 1
        Exchange A[i] and A[j]
    FI
OD
Exchange A[i + 1] and A[r]
RETURN i + 1
```

- Correctness:
  - Clear if PARTITION divides correctly
  - PARTITION can be proved correct (by induction) using the following invariants at the beginning of each FOR loop execution (immediately after j is incremented):
    1. $A[p\,..\,i] \leq x$
    2. $A[i+1\,..\,j-1] > x$
    3. $A[j\,..\,r-1]$ is unrestricted at the moment
    4. $x = A[r]$ is the partitioning element
- PARTITION runs in time $\Theta(n)$
- How well does PARTITION divide A?
  - Example (where PARTITION does reasonably well): see Figure 7.1 in book
  - In the worst case q is always p and the running time becomes $T(n) = \Theta(n) + T(1) + T(n-1) \Longrightarrow T(n) = \Theta(n^2)$.
    * and what is maybe even worse, the worst case is when A is already sorted or reverse-sorted.
- So why is it called "quick" sort? Because it "often" performs very well—can we theoretically justify this?
  - Even if all the splits are relatively bad, we get $\Theta(n \log n)$ time:
    * Example: Split is $\frac{9}{10}n$, $\frac{1}{10}n$.
      $T(n) = T(\frac{9}{10}n) + T(\frac{1}{10}n) + n$
      Solution?
      Guess: $T(n) \leq cn \log n$
    * We prove that the guess works (i.e., we find some values of c that it works for) by using induction. We leave proving the base case (i.e., for the smaller values of n) to the reader. Let's look at the inductive step first. We assume that our guess works for all arguments of T less than n and now we check to see if it works for n. Therefore, we start with the recurrence, and by the inductive hypothesis, we can substitute our guess for the T terms on the RHS:

$$T(n) = T\left(\frac{9}{10}n\right) + T\left(\frac{1}{10}n\right) + n$$

$$
\begin{aligned}
&\leq \frac{9cn}{10}\log\frac{9n}{10} + \frac{cn}{10}\log\frac{n}{10} + n \\
&\leq \frac{9cn}{10}\log n + \frac{9cn}{10}\log\frac{9}{10} + \frac{cn}{10}\log n + \frac{cn}{10}\log\frac{1}{10} + n \\
&\leq cn\log n + \frac{9cn}{10}\log 9 - \frac{9cn}{10}\log 10 - \frac{cn}{10}\log 10 + n \\
&\leq cn\log n - n(c\log 10 - \frac{9c}{10}\log 9 - 1)
\end{aligned}
$$

  * $T(n) \leq cn\log n$ if $c\log 10 - \frac{9c}{10}\log 9 - 1 > 0$ which is definitely true if $c > \frac{10}{\log 10}$
- In other words, if the splits happen at a constant fraction of n, we get $T(n) = \Theta(n\lg n)$.

- Next we will further justify the good practical performance by looking more rigorously at the average-case running time.

2 Average-case analysis of Quicksort

- When doing average-case analysis, we need to be careful about what we are averaging over.
- To simplify the analysis, we assume all inputs are distinct—in fact, we will assume that we are sorting the numbers 1 through n
  - our analysis can be extended to general non-distinct inputs.
- We assume that there are $n!$ distinct and *equally likely* input configurations.
- Analysis:
  - q returned by PARTITION is the *rank* of $x = A[r]$ (i.e., the number of elements $\leq x$).
  - Probability that $rank(x) = q$ is $1/n$, for each $1 \leq q \leq n$.
  - $T_a(n)$ = average running time

    We can express the expected running time $T_a(n)$ as the weighted combination of n conditional expected running times, where each conditional expectation is conditioned on a different rank q for the partitioning element:

$$
\begin{aligned}
T_a(n) &= \Theta(n) + \sum_{rank(x)=q} \text{Prob}[rank(x) = q] \cdot (\text{expected running time given that } rank(x) = q) \\
&= \Theta(n) + \frac{1}{n} \cdot \sum_{rank(x)=q} (\text{expected running time given that } rank(x) = q) \\
&= \Theta(n) + \frac{1}{n}\sum_{q=1}^{n}(T_a(q-1) + T_a(n-q))) \\
&= \Theta(n) + \frac{2}{n}\sum_{q=0}^{n-1} T_a(q)
\end{aligned}
$$

We guess that $T_a(n) \leq an\log n + b$. (For simplicity, when $n = 0$, we define $an\log n = \lim_{n\to 0} an\log n = 0$. We prove that the guess is correct by induction We leave the base case for the smaller values of n as an exercise for the reader. Induction step:

$$
T_a(n) = \Theta(n) + \frac{2}{n}\sum_{q=0}^{n-1} T_a(q)
$$

$$
\begin{aligned}
&\leq \Theta(n) + \frac{2}{n}\sum_{q=0}^{n-1}(aq\log q + b) \\
&\leq \Theta(n) + 2b + \frac{2a}{n}\sum_{q=0}^{n-1} q\log q
\end{aligned}
$$

We use $\boxed{\sum_{q=0}^{n-1} q\log q \leq \frac{1}{2}n^2\log n - \frac{\log e}{4}n^2}$

$$
\begin{aligned}
T_a(n) &\leq \Theta(n) + 2b + \frac{2a}{n}\left(\frac{1}{2}n^2\log n - \frac{\log e}{4}n^2\right) \\
&\leq an\log n - \frac{\log e}{2}an + 2b - \frac{2b}{n} + \Theta(n) \\
&\leq an\log n - \frac{\log e}{2}an + 2b + \Theta(n) \\
&\leq (an\log n + b) + \left(b - \frac{\log e}{2}an + \Theta(n)\right)
\end{aligned}
$$

$T_a(n) \leq an\log n + b$ as we can choose a such that $(\log e)an/2$ dominates $b + \Theta(n)$ (For example : $\frac{(\log e)an}{2} \geq n + b \Longleftrightarrow n(\frac{(\log e)a}{2} - 1) > b \Longleftrightarrow \frac{(\log e)a}{2} - 1 > b$)

- Note: If we had just used $\sum_{q=0}^{n-1} q\log q \leq n^2\log n$, the proof would not have worked. We need both the $1/2$ coefficient (less than 1) for the $n^2\log n$ term and a negative coefficient for the n^2 term.
- Proof that $\sum_{q=0}^{n-1} q\log q \leq \frac{1}{2}n^2\log n - \frac{\log e}{4}n^2$ using integral method:

$$
\sum_{q=0}^{n-1} q\log q \leq \int_0^n x\log x\,dx.
$$

The antiderivative of $x\log x$ is $\frac{1}{2}x^2\log x - \frac{\log e}{4}x^2$. Therefore, we get

$$
\sum_{q=0}^{n-1} q\log q \leq \frac{1}{2}n^2\log n - \frac{\log e}{4}n^2.
$$

- Alternate proof, which is a little sloppier but more basic in some sense. It proves a slightly weaker bound that $\sum_{q=0}^{n-1} q\log q \leq \frac{1}{2}n^2\log n - \frac{1}{8}n^2$:

$$
\begin{aligned}
\sum_{q=0}^{n-1} q\log q &= \sum_{q=0}^{\lceil n/2\rceil - 1} q\log q + \sum_{\lceil n/2\rceil}^{n-1} q\log q \\
&\leq \sum_{q=0}^{\lceil n/2\rceil - 1} q\log\frac{n}{2} + \sum_{\lceil n/2\rceil}^{n-1} q\log n \\
&= (\log n - 1)\sum_{q=0}^{\lceil n/2\rceil - 1} q + \log n\sum_{\lceil n/2\rceil}^{n-1} q \\
&= (\log n)\sum_{q=0}^{n-1} q - \sum_{q=0}^{\lceil n/2\rceil - 1} q
\end{aligned}
$$

$$\begin{aligned}
&\le \frac{1}{2}(\log n)(n-1)n - \frac{1}{2}\left(\frac{n}{2}-1\right)\frac{n}{2} \\
&= \frac{1}{2}n^2\log n - \frac{1}{2}n\log n - \frac{1}{8}n^2 + \frac{1}{4}n \\
&= \frac{1}{2}n^2\log n - \frac{1}{8}n^2 - \left(\frac{1}{2}n\log n - \frac{1}{4}n\right) \\
&\le \frac{1}{2}n^2\log n - \frac{1}{8}n^2
\end{aligned}$$

- A more clever proof by manipulating the summation into one with a constant number of terms: We start with the quicksort-type recurrence (assuming for simplicity that the coefficient of the linear term is 1):

$$T_a(n) = n + \frac{2}{n}\sum_{q=0}^{n-1} T_a(q).$$

  - The key point is that the recurrence for $T_a(n)$ and $T_a(n-1)$ share almost the same summation, so by proper manipulation we should be able to get almost all the terms to cancel out. The only obstacle we have to overcome is the $2/n$ factor in front of the summation, which we can overcome by multiplying the recurrence on both sides by n:

$$nT_a(n) = n^2 + 2\sum_{q=0}^{n-1} T_a(q).$$

    The recurrence involving $T_a(n-1)$ is correspondingly

$$(n-1)T_a(n-1) = (n-1)^2 + 2\sum_{q=0}^{n-2} T_a(q).$$

  - Now we can subtract the two recurrences from one another to get rid of the summation:

$$\begin{aligned}
nT_a(n) - (n-1)T_a(n-1) &= n^2 - (n-1)^2 + 2T_a(n-1) \\
nT_a(n) - (n+1)T_a(n-1) &= 2n - 1 \\
T_a(n) &= \frac{n+1}{n}T_a(n-1) + 2 - \frac{1}{n} \\
\frac{T_a(n)}{n+1} &= \frac{T_a(n-1)}{n} + \frac{2}{n} + \Theta\left(\frac{1}{n^2}\right)
\end{aligned}$$

  - Why did we do the very last step? Well, if we think of $T_a(n)/(n+1)$ as a separate term, let's call it $S(n)$, then the recurrence in terms of $S(n)$ is pretty trivial:

$$S(n) = S(n-1) + \frac{2}{n} + \Theta\left(\frac{1}{n^2}\right)$$

    Using the fact that $S(1) = T_a(1)/2 = 1/2$, we get

$$\begin{aligned}
S(n) &= \sum_{1\le i\le n}\frac{2}{i} + \Theta(1) && (1) \\
&= 2H_n + \Theta(1) && (2) \\
&= 2\ln n + \Theta(1), && (3)
\end{aligned}$$

    which we know from the expansion of the harmonic series. Therefore, $T_a(n) = (n+1)S(n) = 2(n+1)\ln n + \Theta(n) = 2n\ln n + \Theta(n)$. Pretty neat, huh?

- So now we saw that both in the best case *and* in the average case, quicksort runs in $\Theta(n\log n)$ time but in the worst case it runs in $\Theta(n^2)$ time $\Longrightarrow$ worst-case cannot happen too often!

3 Randomized Quicksort

- In average-case analysis we assumed that all inputs are equally likely
  - In practice not all inputs are equally likely, in fact, the input is often already somewhat sorted which as we saw results in bad performance
  - We can remedy this problem using "randomization"
- We can enforce that all $n!$ permutations are equally likely by randomly permuting the input before the algorithm
  - Most computers have pseudo-random number generator $random(1,n)$ returning "random" number between 1 and n
  - Using pseudo-random number generator we can generate random permutation (all $n!$ permutations equally likely) in $O(n)$ time:

    Choose element in $A[1]$ randomly among elements in $A[1..n]$, choose element in $A[2]$ randomly among elements in $A[2..n]$, choose element in $A[3]$ randomly among elements in $A[3..n]$, and so on.

    (Note: Just choosing $A[i]$ randomly among elements $A[1..n]$ for all i will not give random permutation!)

  $\Longrightarrow$ Running time $\Theta(n \log n)$ on the average
  - no particular input results in the worst-case behavior (depend only on outcome of random choices)!
- Alternatively we can modify PARTITION slightly and exchange the rth element in A with a random element in A before partitioning
  - The only thing we used in average-case analysis was that $rank(q)$ was i with probability $1/n$.
- In general *randomized algorithms* use random choices independent of input, such that running time depends upon these random choices and not the input.

4 Quicksort Review

- Basic ideas of Quicksort:
  - Divide $A[1..r]$ (using PARTITION) into subarrays $A' = A[1..q-1]$, the partitioning element x, and $A'' = A[q+1..n]$ such that all elements in A' are less than or equal to x, and all elements in A'' are larger than x.
  - Recursively sort A' and A''.
- The split point q produced by PARTITION only depends upon the last element in A:
- Analysis:
  - Best case ($q = n/2$): $T(n) \approx 2T(n/2) + \Theta(n) \Longrightarrow T(n) = \Theta(n \log n)$.
  - Worst case ($q = 1$ or $q = n$): $T(n) = T(n-1) + \Theta(n) \Longrightarrow T(n) = \Theta(n^2)$.
  - Average case: $\Theta(n \log n)$