Lecture 6: Expected Running Time of Quick-Sort

(CLRS 7.3-7.4, (C.2))

May 23rd, 2002

1 Quick-sort review

- Last time we discussed quick-sort.
  - Quick-Sort is "opposite" of merge-sort
  - Obtained using divide-and-conquer
- Abstract algorithm
  - Divide $A[1...n]$ into subarrays $A' = A[1..q-1]$ and $A" = A[q+1...n]$ such that all elements in $A"$ are larger than $A[q]$ and all elements in A' are smaller than $A[q]$.
  - Recursively sort A' and $A"$.
  - (nothing to combine/merge. A already sorted after sorting A' and $A"$)
- Pseudo code:

```
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
```

```
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$(A, 1, n)$

- Analysis :
  - PARTITION runs in $\Theta(r-p)$ time.
  - If array is always partitioned nicely in two halves (partition returns $q = \frac{r-p}{2}$), we have the recurrence $T(n) = 2T(n/2) + \Theta(n) \Rightarrow T(n) = \Theta(n \lg n)$.
  - But in the worst case, PARTITION always returns $q = p$ (when input is sorted) and in this case we get the recurrence $T(n) = T(n-1) + T(1) + \Theta(n) \Rightarrow T(n) = \Theta(n^2)$

    What's maybe even worse is that the worst-case happens when the data is already sorted.
- Quick-sort "often" perform well in practice and last time we started trying to justify this theoretically.
  - We saw that even if all the splits are relatively bad (we looked at the case $\frac{9}{10}n$, $\frac{1}{10}n$) we still get worst-case running time $O(n \log n)$.
  - To justify it further we define *average* and *expected* running time.

2 Average and Expected Running Time (Randomized Algorithms)

- We are normally interested in *worst-case* running time of an algorithm, that is, the maximal running time over all input of size n

$$T(n) = \max_{|X|=n} T(X)$$

- We are sometimes interested in analyzing the *average-case* running time of an algorithm, that is, the *expected* value for the running time, over all input of size n

$$T_a(n) = E_{|X|=n}[T(n)] = \sum_{|X|=n} T(X) \cdot Pr[X]$$

- The problem is that we often don't know the probability $Pr[X]$ of getting a particular input X.
  - Sometime we assume that all possible inputs are equally likely, but thats often not very realistic in practice.
- Instead of using average case running time we therefore consider what we call *randomized algorithms*, that is, algorithms that make some random choices during their execution
  - Running time of normal *deterministic* algorithm only depend on he input configuration.
  - Running time of randomized algorithm depend not only on input configuration but also on the random choices made by the algorithm.
  - Running time of a randomized algorithm is not fixed for a given input!
- We are often interested in analyzing the *worst-case expected* running time of a randomized algorithm, that is, the maximal of the average running times for all inputs of size n

$$T_e(n) = \max_{|X|=n} E[T(X)]$$

3 Randomized Quick-Sort

- We could analyze quick-sort assuming that we are sorting numbers 1 through n and that all $n!$ different input configurations are equally likely.
  - Average running time would be $T_a(n) = O(n \log n)$.
- The assumption that all inputs are equally likely are not very realistic (data tend to be somewhat sorted).
- 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!)
- Alternatively we can modify Partition sightly and exchange last element in A with random element in A before partitioning

```
RandPartition(A, p, r)
i=Random(p, r)
Exchange A[r] and A[i]
RETURN Partition(A, p, r)
```

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

4 Expected Running Time of Randomized Quick-Sort

- Running time of RANDQUICKSORT is dominated by the time spent in PARTITION procedure.
- PARTITION is called n times
  - The pivot element x is not included in any recursive calls.
- One call of PARTITION takes $O(1)$ time plus time proportional to the number of iterations of FOR-loop.
  - In each iteration of FOR-loop we compare an element with the pivot element.

  $\Downarrow$

  If X is the number of comparisons $A[j] \leq x$ performed in PARTITION over the entire execution of RANDQUICKSORT then the running time is $O(n + X)$.
- To analyze the expected running time we need to compute $E[X]$
  - To compute X we use $z_1, z_2, \ldots, z_n$ to denote the elements in A where z_i is the ith smallest element. We also use Z_{ij} to denote $\{z_i, z_{i+1}, \ldots, z_j\}$.
  - Each pair of elements z_i and z_j are compared at most ones (when either of them is the pivot)

    $\Downarrow$

    $X = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}$ where

    $$X_{ij} = \begin{cases} 1 & \text{If } z_i \text{ compared to } z_i \\ 0 & \text{If } z_i \text{ not compared to } \mathrm{z_i} \end{cases}$$

    $\Downarrow$

    $$\begin{aligned} E[X] &= E\left[\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}\right] \\ &= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} E[X_{ij}] \\ &= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} Pr[z_i \text{ compared to } z_j] \end{aligned}$$
  - To compute $Pr[z_i \text{ compared to } z_j]$ it is useful to consider when two elements are *not* compared.

    > Example: Consider an input consisting of numbers 1 through n.
    > Assume first pivot it 7 $\Rightarrow$ first partition separates the numbers into sets $\{1, 2, 3, 4, 5, 6\}$ and $\{8, 9, 10\}$.
    > In partitioning, 7 is compared to all numbers. No number from the first set will ever be compared to a number from the second set.

    In general, once a pivot x, $z_i < x < z_j$, is chosen, we know that z_i and z_j cannot later be compared.

    On the other hand, if z_i is chosen as pivot before any other element in Z_{ij} then it is compared to each element in Z_{ij}. Similar for z_j.

    > In example: 7 and 9 are compared because 7 is first item from $Z_{7,9}$ to be chosen as pivot, and 2 and 9 are not compared because the first pivot in $Z_{2,9}$ is 7.

    Prior to an element in Z_{ij} being chosen as pivot, the set Z_{ij} is together in the same partition $\Rightarrow$ any element in Z_{ij} is equally likely to be first element chosen as pivot $\Rightarrow$ the probability that z_i or z_j is chosen first in Z_{ij} is $\frac{1}{j-i+1}$

    $\Downarrow$

    $Pr[z_i \text{ compared to } z_j] = \frac{2}{j-i+1}$

- We now have:

$$\begin{aligned} E[X] &= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} Pr[z_i \text{ compared to } z_j] \\ &= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1} \\ &= \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{2}{k+1} \\ &< \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{2}{k} \\ &= \sum_{i=1}^{n-1} O(\log n) \\ &= O(n \log n) \end{aligned}$$

- Next time we will see how to make quick-sort run in worst-case $O(n \log n)$ time.