COT 6405 Introduction to Theory of Algorithms

Topic 9. Randomized Quicksort

Worst case quicksort

- What will happen if the array is already sorted?
  - The partitioning routine produces n-1 elements and one with 0 elements.
  - How about the running time?
  - T(n) = O(n^2)

Improving quicksort

- The real liability of quicksort is that it runs in $O(n^2)$ on an already-sorted input
- How to avoid this?
- Two solutions
  - Randomize the input array
  - Pick a random pivot element
- How will these solve the problem?
  - By insuring that no particular input can be chosen to make quicksort run in $O(n^2)$ time

Randomized version of quicksort

- We add randomization to quicksort.
  – We could randomly permute the input array: very costly
  – Instead, we use **random sampling** to pick one element at random as the pivot
    - Don’t always use $A[r]$ as the pivot.

Randomized version of quicksort

RANDOMIZED-PARTITION*(A, p, r)*

i ←RANDOM*(p, r)*

exchange *A*[*r*] ↔ *A*[*i*]

return PARTITION*(A, p, r)*

Randomization of quicksort stops any specific type of array from causing the worst case behavior

– E.g., an already-sorted array causes worst-case behavior in non-randomized QUICKSORT, but not in RANDOMIZED-QUICKSORT.

Randomized version of quicksort

RANDOMIZED-QUICKSORT(A, p, r)

if $p < r$

then $q \leftarrow$ RANDOMIZED-PARTITION(A, p, r)

RANDOMIZED-QUICKSORT$(A, p, q - 1)$

RANDOMIZED-QUICKSORT$(A, q + 1, r)$

Analysis of quicksort

- We will analyze
  - the worst-case running time of QUICKSORT and RANDOMIZED-QUICKSORT
  - the expected (average-case) running time of QUICKSORT and RANDOMIZED-QUICKSORT

Worst-case analysis

- We saw a worst-case split (0:n-1) at every level of recursion in quicksort produces a $\Theta(n^2)$ running time, which,
  - Intuitively, is the worst-case running time
- We now prove this assertion

Worst-case analysis (cont'd)

- Let T (n) be the worst-case time for the procedure QUICKSORT on an input of size n, we have the recurrence
- $T(n) = \max_{0 \le q \le n-1} (T(q) + T(n-q-1)) + \Theta(n)$
  - q ranges between 0 and n-1, because the procedure PARTITION produces two subproblems with total size n-1
- We guess that T(n) $\le cn^2$ for some constant c

Worst-case analysis (cont'd)

- Substitution this guess into the recurrence, we obtain

$$T(n) \leq \max_{0 \leq q \leq n-1} (cq^2 + \mathrm{c}(n-q-1)^2) + \Theta(n)$$

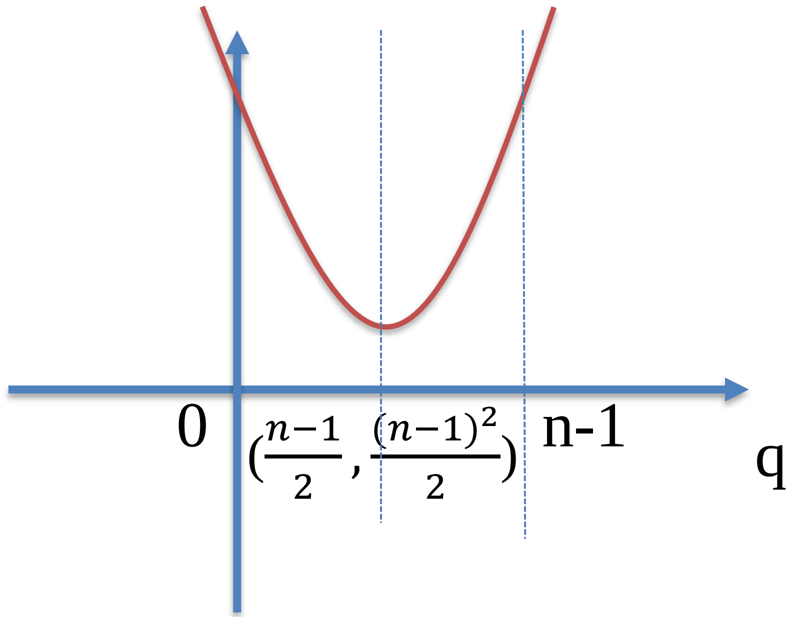
$$= c \max_{0 \leq q \leq n-1} (q^2 + (n-q-1)^2) + \Theta(n)$$

Exercise

- What values of q can enable the expression $q^2 + (n - q - 1)^2$ to achieve the maximum value?

Worst-case analysis (cont'd)

- $q^2 + (n-q-1)^2 = 2q^2 - 2(n-1)q + (n-1)^2$
- What's the shape of this function?
  - A cup-shaped parabola



Worst-case analysis (cont'd)

- $T(n) \leq \max_{0 \leq q \leq n-1} c\,(q^2 + (n-q-1)^2) + \Theta(n)$

The expression $q^2 + (n-q-1)^2$ achieves the maximum value when q is either 0 or n-1.

Worst-case analysis (cont'd)

- This observation gives us the bound
  - $\max_{0 \le q \le n-1} (q^2 + (n-q-1)^2) = (n-1)^2$

    $= n^2 - 2n + 1$
- Continuing with our bounding of T(n), we obtain

$$T(n) \le c \max_{0 \le q \le n-1} (q^2 + (n-q-1)^2) + \Theta(n)$$

$$= c\, n^2 - c(2n-1) + \Theta(n)$$

$$\le c\, n^2$$

Since we can pick *c* large enough so that *c*(2*n*-1) dominates $\Theta(n)$, T(*n*) = O (n^2)

Exercise

- Let T (n) be the worst-case time for the procedure QUICKSORT on an input of size n. Prove T(n) = $\Omega\ (n^2)$

Worst-case analysis (cont'd)

- $T(n) = \max_{0 \le q \le n-1} (T(q) + T(n-q-1)) + \Theta(n)$
- We guess that T(n) $\ge dn^2$ for some constant d Substitution this guess into the recurrence, we obtain

$$T(n) \ge \max_{0 \le q \le n-1} (dq^2 + d(n-q-1)^2) + \Theta(n)$$

$$= d \max_{0 \le q \le n-1} (q^2 + (n-q-1)^2) + \Theta(n)$$

$$= \mathrm{d}\, n^2 - \mathrm{d}(2\mathrm{n}-1) + \Theta(n)$$

$$\ge \mathrm{d}\, n^2$$

Since we can pick a small *d* so that $\Theta(n)$dominates *d*(2*n*-1) , T(*n*) = Ω (n^2)

Average case analysis

- The dominant cost of the algorithm is partitioning.
- What is the maximum number of calls to the function PARTITION?
  - Hint: PARTITION removes the pivot element from future consideration each time.
  - Thus, PARTITION is called at most n times.

Partition array $A[p..r]$

```
PARTITION(A, p, r)
        x ← A[r]           // select the pivot
        i ← p − 1
        for j ← p to r − 1
                if A[ j ] ≤ x
                        i ← i + 1
                        exchange A[i ] ↔ A[ j ]
        // move the pivot between the two subarraies
        exchange A[i + 1] ↔ A[r]
        // return the pivot
        return i + 1
```

Average case analysis (cont'd)

Lemma 7.1: Let X be the number of comparisons performed in line 4 of PARTITION over the entire execution of QUICKSORT on an n-element array. Then the running time of QUICKSORT is $O(n + X)$.

The amount of work of each call to PARTITION is a constant plus the number of comparisons performed in its for loop

Find the number of comparisons

- For ease of analysis:
  - Rename the elements of A as $z_1, z_2, \ldots, z_n$, with z_i being the i-th smallest element.
- Each pair of elements is compared at most once. Why?
- Because elements are compared only to the pivot element, and then the pivot element is never in any later call to PARTITION.

Cont'd

- Our analysis uses indicator random variables
- Let $X_{i,j}$ = I{z_i is compared to z_j }.

$$= \begin{cases} 1 \text{ if } z_i \text{ is compared to } z_j \\ 0 \text{ if } z_i \text{ is not compared to } z_j \end{cases}$$

- Considering whether z_i is compared to z_j at any time during the entire quicksort algorithm, not just during one call of PARTITION.

Cont’d

- Since each pair is compared at most once, the total number of comparisons performed by the algorithm is

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

Take expectations of both sides, use Lemma 5.1 and linearity of expectation:

$$\begin{aligned} \mathrm{E}[X] &= \mathrm{E}\left[\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}\right] \\ &= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \mathrm{E}[X_{ij}] \\ &= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \Pr\{z_i \text{ is compared to } z_j\} . \end{aligned}$$

Exercise

- Prove E[X_{ij}] = Pr (z_i is compared to z_j)

Cont'd

- $E[X_{ij}] = 1 \cdot Pr(X_{ij} = 1) + 0 \cdot Pr(X_{ij} = 0)$

  $= Pr(X_{ij} = 1)$

  $=$ *Pr* (z_i is compared to z_j)

Cont’d

- Now we need to find the probability that two elements are compared.
- Think about when two elements are *not* compared.
  - numbers in separate partitions will not be compared.
  - {8, 1, 6, 4, 0, 3, 9, 5} and the pivot is 5, so that none of the set {1, 4, 0, 3} will be compared to any of the set {8, 6, 9}

Cont'd

- Once a pivot x is chosen, such that z_i< x < z_j, then z_i and z_j will never be compared at any later time
- If either $z_i$or z_j is chosen as a pivot before any other element of Z_{ij}, then it will be compared to all the elements of Z_{ij} , except itself.
- The probability that z_i is compared to z_j is the probability that either z_i or z_j is the first element chosen to be the pivot

Cont'd

- Assume pivots are chosen randomly and independently.
- z_i and z_j must be in the same set after partition, otherwise they will never be compared
- Thus, the probability that any particular one of them is the first one chosen is $1/ n_{ij}$,
  where n_{ij} is the size of this set

Cont'd

- Therefore
- *Pr* (z_i is compared to z_j) = *Pr* (z_i or z_j is the first pivot chosen from the set) = *Pr* (z_i is the first pivot chosen from the set) + *Pr* (z_j is the first pivot chosen from the set)

= 1/ n_{ij} + 1/ n_{ij} = 2/ n_{ij}

Cont'd

- E(X) = $\sum_{i=1}^{i=n-1} \sum_{j=i+1}^{n} Pr$ (z_i is compared to z_j)

  $= \sum_{i=1}^{i=n-1} \sum_{j=i+1}^{n} \frac{2}{n_{ij}}$

When *i* = 1 and *j* = 2, n_{ij} reaches the smallest value of 2, and when *i* = 1 and *j* = *n*, n_{ij} reaches the largest value of *n*. Thus, n_{ij} ranges between 2 and *n,* and by changing variable (let *k* = n_{ij}), we have

$$E(X) = \sum_{i=1}^{i=n-1} \sum_{j=i+1}^{n} \frac{2}{n_{ij}} = \sum_{i=1}^{i=n-1} \sum_{k=2}^{n} \frac{2}{k}$$

Cont'd

- E(X)= $\sum_{i=1}^{i=n-1}\sum_{k=2}^{n}\frac{2}{k} = \sum_{i=1}^{i=n-1} \mathrm{O}(lgn)$
  = O(nlgn)

Harmonic Series:
$\sum_{k=1}^{n}\frac{2}{k} = 2\sum_{k=1}^{n}\frac{1}{k} < 2\ln n+1 = \mathrm{O}(\lg n)$