

Chapter 14

Randomized Algorithms: QuickSort and QuickSelect

CS 473: Fundamental Algorithms, Fall 2011
October 18, 2011

14.1 Slick analysis of QuickSort

14.1.0.1 A Slick Analysis of QuickSort

Let $Q(A)$ be number of comparisons done on input array A :

- (A) For $1 \leq i < j < n$ let R_{ij} be the event that rank i element is compared with rank j element.
- (B) X_{ij} is the indicator random variable for R_{ij} . That is, $X_{ij} = 1$ if rank i is compared with rank j element, otherwise 0.

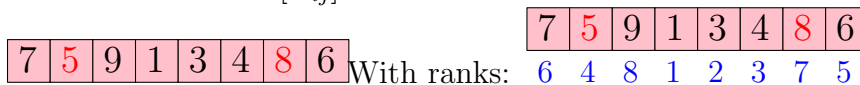
$$Q(A) = \sum_{1 \leq i < j \leq n} X_{ij}$$

and hence by linearity of expectation,

$$\mathbf{E}[Q(A)] = \sum_{1 \leq i < j \leq n} \mathbf{E}[X_{ij}] = \sum_{1 \leq i < j \leq n} \Pr[R_{ij}].$$

14.1.0.2 A Slick Analysis of QuickSort

Question: What is $\Pr[R_{ij}]$?



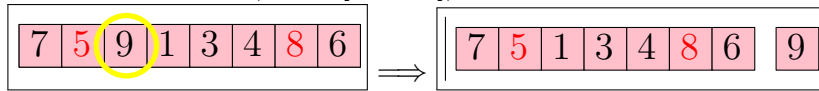
As such, probability of comparing 5 to 8 is $\Pr[R_{4,7}]$.

- (A) If pivot too small (say 3 [rank 2]). Partition and call recursively:



Decision if to compare 5 to 8 is moved to subproblem.

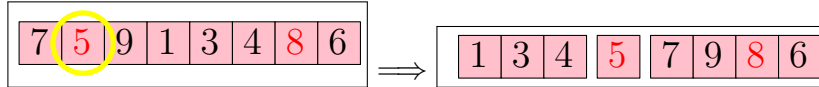
(B) If pivot too large (say 9 [rank 8]):



Decision if to compare 5 to 8 moved to subproblem.

14.1.1.1 Question: What is $\Pr[R_{i,j}]$?

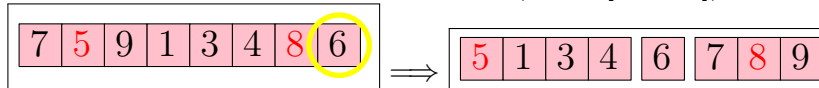
(A) If pivot is 5 (rank 4). Bingo!



(B) If pivot is 8 (rank 7). Bingo!



(C) If pivot in between the two numbers (say 6 [rank 5]):



5 and 8 will never be compared to each other.

14.1.2 A Slick Analysis of QuickSort

14.1.2.1 Question: What is $\Pr[R_{i,j}]$?

Conclusion:

$R_{i,j}$ happens if and only if:

i th or j th ranked element is the first pivot out of
 i th to j th ranked elements

How to analyze this?

Thinking acrobatics!

(A) Assign every element in the array a random priority (say in $[0, 1]$).

(B) Choose pivot to be the element with lowest priority in subproblem.

(C) Equivalent to picking pivot uniformly at random
(as **QuickSort** do).

14.1.3 A Slick Analysis of QuickSort

14.1.3.1 Question: What is $\Pr[R_{i,j}]$?

How to analyze this?

Thinking acrobatics!

- (A) Assign every element in the array a random priority (say in $[0, 1]$).
- (B) Choose pivot to be the element with lowest priority in subproblem.
 $\implies R_{i,j}$ happens if either i or j have lowest priority out of elements rank i to j ,
 As such

$$\Pr[R_{i,j}] = \frac{2}{j-i+1}.$$

14.1.3.2 A Slick Analysis of QuickSort

Question: What is $\Pr[R_{ij}]$?

Lemma 14.1.1 $\Pr[R_{ij}] = \frac{2}{(j-i+1)}.$

Proof: Let $a_1, \dots, a_i, \dots, a_j, \dots, a_n$ be elements of A in sorted order. Let $S = \{a_i, a_{i+1}, \dots, a_j\}$

Observation: If pivot is chosen outside S then all of S either in left array or right array.

Observation: a_i and a_j separated when a pivot is chosen from S for the first time. Once separated no comparison.

Observation: a_i is compared with a_j if and only if either a_i or a_j is chosen as a pivot from S at separation... ■

14.1.4 A Slick Analysis of QuickSort

14.1.4.1 Continued...

Lemma 14.1.2 $\Pr[R_{ij}] = \frac{2}{(j-i+1)}.$

Proof: Let $a_1, \dots, a_i, \dots, a_j, \dots, a_n$ be sort of A . Let $S = \{a_i, a_{i+1}, \dots, a_j\}$

Observation: a_i is compared with a_j if and only if either a_i or a_j is chosen as a pivot from S at separation.

Observation: Given that pivot is chosen from S the probability that it is a_i or a_j is exactly $2/|S| = 2/(j-i+1)$ since the pivot is chosen uniformly at random from the array. ■

14.1.5 A Slick Analysis of QuickSort

14.1.5.1 Continued...

$$\mathbf{E}[Q(A)] = \sum_{1 \leq i < j \leq n} \mathbf{E}[X_{ij}] = \sum_{1 \leq i < j \leq n} \Pr[R_{ij}].$$

Lemma 14.1.3 $\Pr[R_{ij}] = \frac{2}{(j-i+1)}.$

$$\begin{aligned}
\mathbf{E}[Q(A)] &= \sum_{1 \leq i < j \leq n} \Pr[R_{ij}] = \sum_{1 \leq i < j \leq n} \frac{2}{j-i+1} \\
&= \sum_{i=1}^{n-1} \sum_{j=i+1}^n \frac{2}{j-i+1} = 2 \sum_{i=1}^{n-1} \sum_{i < j}^n \frac{1}{j-i+1} \\
&= 2 \sum_{i=1}^{n-1} (H_{n-i+1} - 1) \leq 2 \sum_{1 \leq i < n} H_n \\
&\leq 2nH_n = O(n \log n)
\end{aligned}$$

14.2 QuickSelect with high probability

14.2.1 Yet another analysis of QuickSort

14.2.1.1 You should never trust a man who has only one way to spell a word

Consider element e in the array.

Consider the subproblems it participates in during **QuickSort** execution:

S_1, S_2, \dots, S_k .

Definition

e is lucky in the j th iteration if $|S_j| \leq (3/4)|S_{j-1}|$.

Key observation

The event e is lucky in j th iteration

is independent of

the event that e is lucky in k th iteration,

(If $j \neq k$)

$X_j = 1$ **iff** e is lucky in the j th iteration.

14.2.2 Yet another analysis of QuickSort

14.2.2.1 Continued...

Observation

$\Pr[X_j = 1] = 1/2$.

Observation

If $X_1 + X_2 + \dots + X_k = \lceil \log_{4/3} n \rceil$ then e subproblem is of size one. Done!

14.2.3 Yet another analysis of QuickSort

14.2.3.1 Continued...

Observation

Probability e participates in $\geq k = 40 \lceil \log_{4/3} n \rceil$ subproblems. Is equal to

$$\begin{aligned} & \Pr[X_1 + X_2 + \dots + X_k \leq \lceil \log_{4/3} n \rceil] \\ & \leq \Pr[X_1 + X_2 + \dots + X_k \leq k/4] \\ & \leq 2 \cdot 0.68^{k/4} \leq 1/n^5. \end{aligned}$$

Conclusion

QuickSort takes $O(n \log n)$ time with high probability.

14.3 Randomized Selection

14.3.0.2 Randomized Quick Selection

Input Unsorted array A of n integers

Goal Find the j th smallest number in A (*rank j number*)

Randomized Quick Selection

- (A) Pick a pivot element *uniformly at random* from the array
- (B) Split array into 3 subarrays: those smaller than pivot, those larger than pivot, and the pivot itself.
- (C) Return pivot if rank of pivot is j
- (D) Otherwise recurse on one of the arrays depending on j and their sizes.

14.3.0.3 Algorithm for Randomized Selection

Assume for simplicity that A has distinct elements.

```
QuickSelect( $A, j$ ):
    Pick pivot  $x$  uniformly at random from
    Partition  $A$  into  $A_{\text{less}}, x$ , and  $A_{\text{greater}}$ 
    if ( $|A_{\text{less}}| = j - 1$ ) then
        return  $x$ 
    if ( $|A_{\text{less}}| \geq j$ ) then
        return QuickSelect( $A_{\text{less}}, j$ )
    else
        return QuickSelect( $A_{\text{greater}}, j - |A_{\text{less}}|$ )
```

14.3.0.4 QuickSelect analysis

- (A) S_1, S_2, \dots, S_k be the subproblems considered by the algorithm.
Here $|S_1| = n$.
- (B) S_i would be **successful** if $|S_i| \leq (3/4) |S_{i-1}|$
- (C) $Y_1 =$ number of recursive calls till first successful iteration.
Clearly, total work till this happens is $O(Y_1 n)$.
- (D) $n_i =$ size of the subproblem immediately after the $(i - 1)$ th successful iteration.
- (E) $Y_i =$ number of recursive calls after the $(i - 1)$ th successful call, till the i th successful iteration.
- (F) Running time is $O(\sum_i n_i Y_i)$.

14.3.0.5 QuickSelect analysis

Example

$S_i =$ subarray used in i th recursive call

$|S_i| =$ size of this subarray

Red indicates successful iteration.

| | | | | | | | | | |
|---------|-------------|-------|------------|-------|-------|------------|-------|-----------|-------|
| Inst' | S_1 | S_2 | S_3 | S_4 | S_5 | S_6 | S_7 | S_8 | S_9 |
| $ S_i $ | 100 | 70 | 60 | 50 | 40 | 30 | 25 | 5 | 2 |
| Succ' | $Y_1 = 2$ | | $Y_2 = 4$ | | | $Y_3 = 2$ | | $Y_4 = 1$ | |
| $n_i =$ | $n_1 = 100$ | | $n_2 = 60$ | | | $n_3 = 25$ | | $n_4 = 2$ | |

- (A) All the subproblems after $(i - 1)$ th successful iteration till i th successful iteration have size $\leq n_i$.
- (B) Total work: $O(\sum_i n_i Y_i)$.

14.3.0.6 QuickSelect analysis

Total work: $O(\sum_i n_i Y_i)$.

We have:

- (A) $n_i \leq (3/4)n_{i-1} \leq (3/4)^{i-1}n$.
- (B) Y_i is a random variable with geometric distribution
Probability of $Y_i = k$ is $1/2^k$.
- (C) $\mathbf{E}[Y_i] = 2$.

As such, expected work is proportional to

$$\begin{aligned} \mathbf{E}\left[\sum_i n_i Y_i\right] &= \sum_i \mathbf{E}[n_i Y_i] \leq \sum_i \mathbf{E}[(3/4)^{i-1} n Y_i] \\ &= n \sum_i (3/4)^{i-1} \mathbf{E}[Y_i] = n \sum_{i=1}^{\infty} (3/4)^{i-1} 2 \leq 8n. \end{aligned}$$

14.3.0.7 QuickSelect analysis

Theorem 14.3.1 *The expected running time of QuickSelect is $O(n)$.*

14.3.1 QuickSelect analysis

14.3.1.1 Analysis via Recurrence

- (A) Given array A of size n let $Q(A)$ be number of comparisons of randomized selection on A for selecting rank j element.
- (B) Note that $Q(A)$ is a random variable
- (C) Let A_{less}^i and A_{greater}^i be the left and right arrays obtained if pivot is rank i element of A .
- (D) Algorithm recurses on A_{less}^i if $j < i$ and recurses on A_{greater}^i if $j > i$ and terminates if $j = i$.

$$\begin{aligned} Q(A) &= n + \sum_{i=1}^{j-1} \Pr[\text{pivot has rank } i] Q(A_{\text{greater}}^i) \\ &\quad + \sum_{i=j+1}^n \Pr[\text{pivot has rank } i] Q(A_{\text{less}}^i) \end{aligned}$$

14.3.1.2 Analyzing the Recurrence

As in **QuickSort** we obtain the following recurrence where $T(n)$ is the worst-case expected time.

$$T(n) \leq n + \frac{1}{n} \left(\sum_{i=1}^{j-1} T(n-i) + \sum_{i=j}^n T(i-1) \right).$$

Theorem 14.3.2 $T(n) = O(n)$.

Proof: (Guess and) Verify by induction (see next slide). ■

14.3.1.3 Analyzing the recurrence

Theorem 14.3.3 $T(n) = O(n)$.

Prove by induction that $T(n) \leq \alpha n$ for some constant $\alpha \geq 1$ to be fixed later.

Base case: $n = 1$, we have $T(1) = 0$ since no comparisons needed and hence $T(1) \leq \alpha$.

Induction step: Assume $T(k) \leq \alpha k$ for $1 \leq k < n$ and prove it for $T(n)$. We have by the recurrence:

$$\begin{aligned} T(n) &\leq n + \frac{1}{n} \left(\sum_{i=1}^{j-1} T(n-i) + \sum_{i=j}^n T(i-1) \right) \\ &\leq n + \frac{\alpha}{n} \left(\sum_{i=1}^{j-1} (n-i) + \sum_{i=j}^n (i-1) \right) \quad \text{by applying induction} \end{aligned}$$

14.3.1.4 Analyzing the recurrence

$$\begin{aligned}T(n) &\leq n + \frac{\alpha}{n} \left(\sum_{i=1}^{j-1} (n-i) + \sum_{i=j}^n (i-1) \right) \\&\leq n + \frac{\alpha}{n} \left((j-1)(2n-j)/2 + (n-j+1)(n+j-2)/2 \right) \\&\leq n + \frac{\alpha}{2n} (n^2 + 2nj - 2j^2 - 3n + 4j - 2) \\&\quad \text{above expression maximized when } j = (n+1)/2: \text{ calculus} \\&\leq n + \frac{\alpha}{2n} (3n^2/2 - n) \quad \text{substituting } (n+1)/2 \text{ for } j \\&\leq n + 3\alpha n/4 \\&\leq \alpha n \quad \text{for any constant } \alpha \geq 4\end{aligned}$$

14.3.1.5 Comments on analyzing the recurrence

- (A) Algebra looks messy but intuition suggest that the median is the hardest case and hence can plug $j = n/2$ to simplify without calculus
- (B) Analyzing recurrences comes with practice and after a while one can see things more intuitively

John Von Neumann:

Young man, in mathematics you don't understand things. You just get used to them.

14.3.1.6 If there is time...

Sketch Treaps and how **QuickSort** implies $O(\log n)$ time per operation (with high probability).