

Randomized Algorithms: QuickSort and QuickSelect

Lecture 14

October 18, 2011

Part I

Slick analysis of QuickSort

A Slick Analysis of QuickSort

Let $Q(\mathbf{A})$ be number of comparisons done on input array \mathbf{A} :

- For $1 \leq i < j < n$ let R_{ij} be the event that rank i element is compared with rank j element.
- 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 .

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

and hence by linearity of expectation,

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

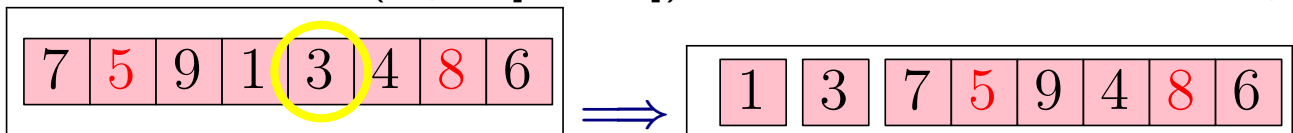
A Slick Analysis of QuickSort

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



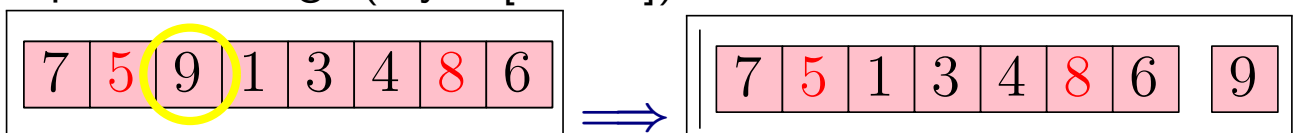
As such, probability of comparing **5** to **8** is $\Pr[R_{4,7}]$.

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



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

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



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

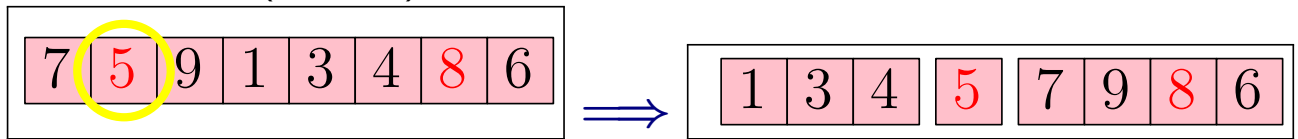
A Slick Analysis of QuickSort

7 5 9 1 3 4 8 6

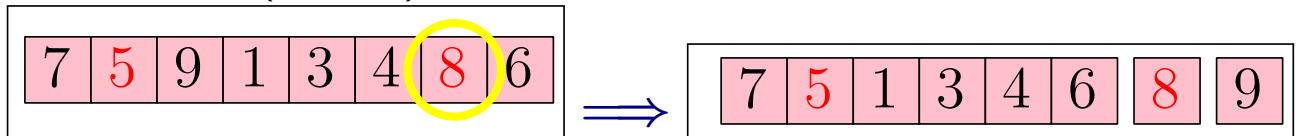
6 4 8 1 2 3 7 5

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

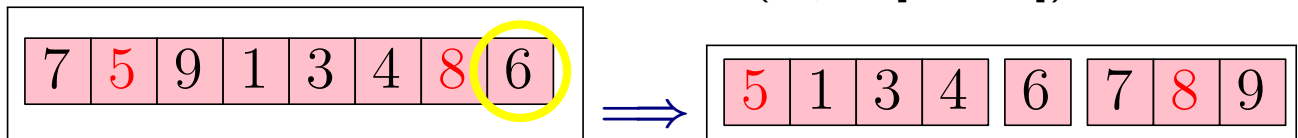
- If pivot is 5 (rank 4). Bingo!



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



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



5 and 8 will never be compared to each other.

A Slick Analysis of QuickSort

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!

- Assign every element in the array a random priority (say in $[0, 1]$).
- Choose pivot to be the element with lowest priority in subproblem.
- Equivalent to picking pivot uniformly at random (as **QuickSort** do).

A Slick Analysis of QuickSort

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

How to analyze this?

Thinking acrobatics!

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

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

A Slick Analysis of QuickSort

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

Lemma

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

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... \square

A Slick Analysis of QuickSort

Continued...

Lemma

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

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. \square

A Slick Analysis of QuickSort

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

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

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

Yet another analysis of QuickSort

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.

Yet another analysis of QuickSort

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!

Yet another analysis of QuickSort

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.

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

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

Algorithm for Randomized Selection

Assume for simplicity that A has distinct elements.

QuickSelect(A, j):

Pick pivot x uniformly at random from A

Partition A into A_{less} , x , and A_{greater} using x as pivot

if ($|A_{\text{less}}| = j - 1$) **then**

return x

if ($|A_{\text{less}}| \geq j$) **then**

return **QuickSelect**(A_{less}, j)

else

return **QuickSelect**($A_{\text{greater}}, j - |A_{\text{less}}| - 1$)

QuickSelect analysis

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

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'	$Y_1 = 2$		$Y_2 = 4$			$Y_3 = 2$		$Y_4 = 1$	
$n_i =$	$n_1 = 100$		$n_2 = 60$			$n_3 = 25$		$n_4 = 2$	

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

QuickSelect analysis

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

We have:

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

As such, expected work is proportional to

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

QuickSelect analysis

Theorem

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

QuickSelect analysis

Analysis via Recurrence

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

$$Q(\mathbf{A}) = n + \sum_{i=1}^{j-1} \Pr[\text{pivot has rank } i] Q(\mathbf{A}_{\text{greater}}^i) + \sum_{i=j+1}^n \Pr[\text{pivot has rank } i] Q(\mathbf{A}_{\text{less}}^i)$$

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

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

Proof.

(Guess and) Verify by induction (see next slide). □

Analyzing the recurrence

Theorem

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

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:

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

Analyzing the recurrence

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

Comments on analyzing the recurrence

- Algebra looks messy but intuition suggest that the median is the hardest case and hence can plug $j = n/2$ to simplify without calculus
- 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.

If there is time...

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