Chapter 15

Randomized Algorithms: QuickSort and QuickSelect

OLD CS 473: Fundamental Algorithms, Spring 2015
March 12, 2015

15.1 Slick analysis of QuickSort

15.1.0.1 A Slick Analysis of QuickSort

(A) Let $Q(A)$ be number of comparisons done on input array A:
    (A) R_{ij}: event that rank i element is compared with rank j element, for $1 \le i < j \le n$.
    (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.
(B) $Q(A) = \sum_{1 \le i < j \le n} X_{ij}$.
(C) By linearity of expectation,

$$\mathbf{E}\Big[Q(A)\Big] = \mathbf{E}\left[\sum_{1 \le i < j \le n} X_{ij}\right] = \sum_{1 \le i < j \le n} \mathbf{E}\Big[X_{ij}\Big]$$

$$= \sum_{1 \le i < j \le n} \mathbf{Pr}\Big[R_{ij}\Big].$$

15.1.0.2 A Slick Analysis of QuickSort

R_{ij} = rank i element is compared with rank j element.

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

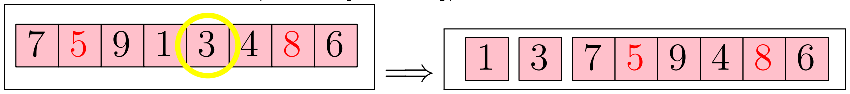
7	5	9	1	3	4	8	6

With ranks:

7	5	9	1	3	4	8	6
6	4	8	1	2	3	7	5

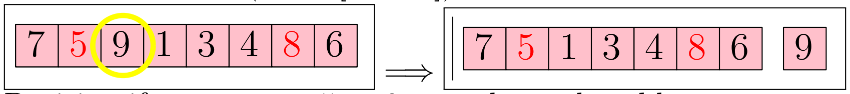
As such, probability of comparing 5 to 8 is $\mathbf{Pr}[R_{4,7}]$.

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



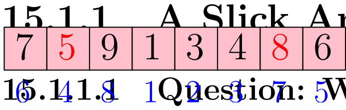
Decision if to compare 5 to 8 is moved to subproblem.

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



Decision if to compare 5 to 8 moved to subproblem.

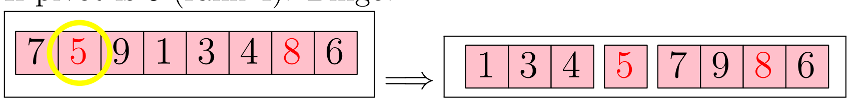
15.1.1 A Slick Analysis of QuickSort



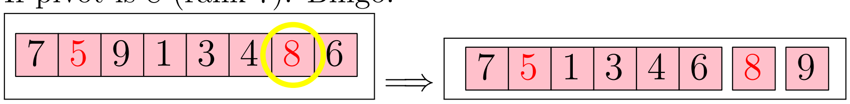
15.1.1.1 Question: What is $\mathbf{Pr}[R_{i,j}]$?

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

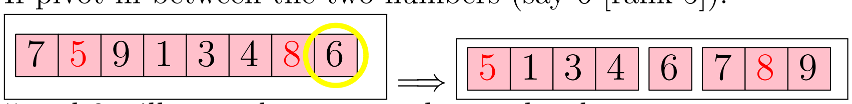
(A) If pivot is 5 (rank 4). Bingo!



(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.

15.1.2 A Slick Analysis of QuickSort

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

Conclusion:

$R_{i,j}$ happens $\iff$:

ith or jth ranked element is the first pivot out of the elements of rank $i, i+1, i+2, \ldots, j$

How to analyze this? Thinking acrobatics!

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

15.1.3 A Slick Analysis of QuickSort

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

(A) Choosing a pivot using priorities
  (A) Assign every element in array is a random priority (in $[0,1]$).
  (B) pivot = the element with lowest priority in subproblem.
(B) $\implies R_{i,j}$ happens if either i or j have lowest priority out of elements in rank $i \ldots j$,
(C) There are $k = j - i + 1$ relevant elements.
(D) $\mathbf{Pr}\Big[R_{i,j}\Big] = \dfrac{2}{k} = \dfrac{2}{j-i+1}$.

15.1.3.2 A Slick Analysis of QuickSort

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

Lemma 15.1.1. $\mathbf{Pr}\Big[R_{ij}\Big] = \frac{2}{j-i+1}$.

Proof
(A) $a_1, \ldots, a_i, \ldots, a_j, \ldots, a_n$: elements of A in sorted order. Let $S = \{a_i, a_{i+1}, \ldots, a_j\}$
(B) **Observation:** If pivot is chosen outside S then all of S either in left or right recursive subproblem.
(C) **Observation:** a_i and a_j separated when a pivot is chosen from S for the first time. Once separated never to meet again. $\implies a_i$ and a_j will not be compared.

15.1.4 A Slick Analysis of QuickSort

15.1.4.1 Continued...

Lemma 15.1.2. $\mathbf{Pr}\Big[R_{ij}\Big] = \frac{2}{j-i+1}$.

Proof:
(A) Let $a_1, \ldots, a_i, \ldots, a_j, \ldots, a_n$ be sort of A.
(B) Let $S = \{a_i, a_{i+1}, \ldots, a_j\}$
(C) **Observation:** a_i is compared with $a_j \iff$ either a_i or a_j is chosen as a pivot from S at separation.
(D) **Observation:** Given: Pivot 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.

■

15.1.5 A Slick Analysis of QuickSort

15.1.5.1 Continued...

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

Lemma 15.1.3. $\mathbf{Pr}[R_{ij}] = \frac{2}{j-i+1}$.

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

15.2 Quick sort with high probability

15.2.1 Yet another analysis of QuickSort

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

(A) Consider element e in the array.
(B) $S_1, S_2, \ldots, S_k$: subproblems e participates in during **QuickSort** execution:
(C) **Definition**
e is lucky in the jth iteration if $|S_j| \le (3/4)\,|S_{j-1}|$.
(D) **Key observation**: The event that e is lucky in jth iteration...
(E) ... is independent of the event that e is lucky in kth iteration,
(If $j \ne k$)
(F) $X_j = 1 \iff e$ is lucky in the jth iteration.

15.2.2 Yet another analysis of QuickSort

15.2.2.1 Continued...

Claim

$\mathbf{Pr}[X_j = 1] = 1/2$.

Proof:
(A) X_j determined by j recursive subproblem.
(B) Subproblem has $n_{j-1} = |X_{j-1}|$ elements.
(C) jth pivot rank $\in [n_{j-1}/4, (3/4)n_{j-1}] \implies e$ lucky in jth iter.
(D) Prob. e is lucky $\ge |[n_{j-1}/4, (3/4)n_{j-1}]|\,/n_{j-1} = 1/2$.

■

Observation

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

15.2.3 Yet another analysis of QuickSort

15.2.3.1 Continued...

Observation

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

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

Conclusion

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

15.3 Randomized Selection

15.3.0.2 Randomized Quick Selection

Input Unsorted array A of n integers

Goal Find the jth 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.

15.3.0.3 Algorithm for Randomized Selection

Assume for simplicity that A has distinct elements.

```
QuickSelect(A, j):
    Pick pivot x uniformly at random fr
    Partition A into A_less, x, and A_grea
    if (|A_less| = j − 1) then
        return x
    if (|A_less| ≥ j) then
        return QuickSelect(A_less, j)
    else
        return QuickSelect(A_greater, j −
```

15.3.0.4 QuickSelect analysis

(A) $S_1, S_2, \ldots, S_k$ 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 ith successful iteration.
(F) Running time is $O(\sum_i n_i Y_i)$.

15.3.0.5 QuickSelect analysis

Example

S_i = subarray used in ith 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 ith successful iteration have size $\leq n_i$.
(B) Total work: $O(\sum_i n_i Y_i)$.

15.3.0.6 QuickSelect analysis

(A) Total work: $O(\sum_i n_i Y_i)$.
(B) $n_i \leq (3/4) n_{i-1} \leq (3/4)^{i-1} n$.
(C) Y_i is a random variable with geometric distribution
Probability of $Y_i = k$ is $1/2^i$.
(D) $\mathbf{E}[Y_i] = 2$.
(E) As such, expected work is proportional to

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

15.3.0.7 QuickSelect analysis

Theorem 15.3.1. *The expected running time of* **QuickSelect** *is* $O(n)$.

15.3.1 QuickSelect analysis via recurrence

15.3.1.1 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^i_{less} and A^i_{greater} be the left and right arrays obtained if pivot is rank i element of A.
(D) Algorithm recurses on A^i_{less} if $j < i$ and recurses on A^i_{greater} if $j > i$ and terminates if $j = i$.

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

15.3.1.2 Analyzing the Recurrence

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

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

Theorem 15.3.2. *$T(n) = O(n)$.*

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

15.3.1.3 Analyzing the recurrence

Theorem 15.3.3. *$T(n) = O(n)$.*

Prove by induction that $T(n) \le \alpha n$ for some constant $\alpha \ge 1$ to be fixed later.
Base case: $n = 1$, we have $T(1) = 0$ since no comparisons needed and hence $T(1) \le \alpha$.
Induction step: Assume $T(k) \le \alpha k$ for $1 \le k < n$ and prove it for $T(n)$. We have by the recurrence:

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

15.3.1.4 Analyzing the recurrence

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

15.3.1.5 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.