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,

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

14.1.0.2 A Slick Analysis of QuickSort

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

With ranks: 6 4 8 1 2 3 7 5

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:

$$\begin{array}{cccccccc}
7 & 5 & 9 & 1 & 3 & 4 & 8 & 6 \\
\end{array}$$

$\Rightarrow$

$$\begin{array}{cccccccc}
1 & 3 & 7 & 5 & 9 & 4 & 8 & 6 \\
\end{array}$$
Decision if to compare 5 to 8 is moved to subproblem.

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

\[
\begin{array}{cccccccc}
7 & 5 & 9 & 1 & 3 & 4 & 8 & 6 \\
\end{array}
\rightarrow
\begin{array}{cccccccc}
7 & 5 & 1 & 3 & 4 & 8 & 6 & 9 \\
\end{array}
\]

Decision if to compare 5 to 8 moved to subproblem.

14.1.1 A Slick Analysis of QuickSort

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

Conclusion: \( R_{i,j} \) happens if and only if:

\[
\begin{array}{cccccccc}
i^\text{th} \text{ or } j^\text{th} \text{ ranked element is the first pivot out of } i^\text{th} \text{ to } j^\text{th} \text{ ranked elements}
\end{array}
\]

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).

14.1.2 A Slick Analysis of QuickSort

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

How to analyze this?

Thinking acrobatics!

14.1.3 A Slick Analysis of QuickSort

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

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.

\[ R_{i,j} \] 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 \( \Pr[R_{i,j}] \)?

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

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

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

**Proof:** Let \( a_1, \ldots, a_i, \ldots, a_j, \ldots, a_n \) be sort of \( A \). Let \( S = \{a_i, a_{i+1}, \ldots, a_j\} \)

**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...

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

**Lemma 14.1.3** \( \Pr[R_{i,j}] = \frac{2}{j - i + 1} \).
\[ \mathbb{E}[Q(A)] = \sum_{1 \leq i < j \leq n} \Pr[R_{ij}] = \sum_{1 \leq i < j \leq n} \frac{2}{j - i + 1} \]
\[ = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j - i + 1} = 2 \sum_{i=1}^{n-1} \sum_{i<j}^{n} \frac{1}{j - i + 1} \]
\[ = 2 \sum_{i=1}^{n-1} (H_{n-i+1} - 1) \leq 2 \sum_{1 \leq i < n} H_n \]
\[ \leq 2nH_n = O(n \log n) \]

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 \texttt{QuickSort} execution:
\( S_1, S_2, \ldots, S_k \).

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 \( X_1 + X_2 + \ldots X_k = \lceil \log_{4/3} n \rceil \) 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

$$\Pr[X_1 + X_2 + \ldots + X_k \leq \lceil \log_{4/3} n \rceil]$$

$$\leq \Pr[X_1 + X_2 + \ldots + X_k \leq k/4]$$

$$\leq 2 \cdot 0.68^{k/4} \leq 1/n^5.$$ 

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

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

Assume for simplicity that $A$ has distinct elements.
14.3.0.4 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 $i$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.

<table>
<thead>
<tr>
<th>Inst'</th>
<th>$S_1$</th>
<th>$S_2$</th>
<th>$S_3$</th>
<th>$S_4$</th>
<th>$S_5$</th>
<th>$S_6$</th>
<th>$S_7$</th>
<th>$S_8$</th>
<th>$S_9$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>S_i</td>
<td>$</td>
<td>100</td>
<td>70</td>
<td>60</td>
<td>50</td>
<td>40</td>
<td>30</td>
<td>25</td>
</tr>
<tr>
<td>Succ'</td>
<td>$Y_1 = 2$</td>
<td>$Y_2 = 4$</td>
<td>$Y_3 = 2$</td>
<td>$Y_4 = 1$</td>
<td>$Y_5 = 1$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$n_i =$</td>
<td>$n_1 = 100$</td>
<td>$n_2 = 60$</td>
<td>$n_3 = 25$</td>
<td>$n_4 = 2$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(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 $1/2^i$.

(C) $E[Y_i] = 2$.

As such, expected work is proportional to

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

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

14.3.0.7 QuickSelect analysis

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

6
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 $A_{\text{less}}^i$ and $A_{\text{greater}}^i$ be the left and right arrays obtained if pivot is rank $i$ element of $A$.

(D) Algorithm recurses on $A_{\text{less}}^i$ if $j < i$ and recurses on $A_{\text{greater}}^i$ if $j > i$ and terminates if $j = i$.

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

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:

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

\[
\leq n + \frac{\alpha}{n} \left( \sum_{i=1}^{j-1} (n-i) + \sum_{i=j}^{n} (i-1) \right)
\]

by applying induction
14.3.1.4 Analyzing the recurrence

\[
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)
\text{above expression maximized when } j = (n + 1)/2: \text{ calculus}
\leq n + \frac{\alpha}{2n} (3n^2/2 - n) \text{ substituting } (n + 1)/2 \text{ for } j
\leq n + 3\alpha n/4
\leq \alpha n \text{ for any constant } \alpha \geq 4

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

\textbf{John Von Neumann:}

\textit{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 \textbf{QuickSort} implies \(O(\log n)\) time per operation (with high probability).