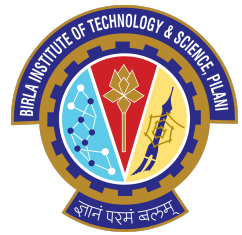
CS F364: Design & Analysis of Algorithms

Lecture-kt05: Randomized Quick Sort



Dr. Kamlesh Tiwari,
Assistant Professor,
Department of Computer Science and Information Systems,
BITS Pilani, Rajasthan-333031 INDIA

Jan 31, 2017 (Campus @ BITS-Pilani Jan-May 2017)

Recap: Master method

When $T(n) = aT(n/b) + f(n)$ $\quad a \geq 1, b > 1$

Let $\epsilon > 0$ be a constant

1. If $f(n) = O(n^{log_b a - \epsilon})$ then $T(n) = \Theta(n^{log_b a})$
2. If $f(n) = \Theta(n^{log_b a})$ then $T(n) = \Theta(n^{log_b a} \log n)$
3. If $f(n) = \Omega(n^{log_b a + \epsilon})$ then $T(n) = \Theta(f(n))$
   provided if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n. **Regularity condition must be checked in case-3.**

Recap: Master method

When $T(n) = aT(n/b) + f(n)$ $\quad a \geq 1, b > 1$

Let $\epsilon > 0$ be a constant

1. If $f(n) = O(n^{log_b a - \epsilon})$ then $T(n) = \Theta(n^{log_b a})$
2. If $f(n) = \Theta(n^{log_b a})$ then $T(n) = \Theta(n^{log_b a} \log n)$
3. If $f(n) = \Omega(n^{log_b a + \epsilon})$ then $T(n) = \Theta(f(n))$
   provided if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n. **Regularity condition must be checked in case-3.**

Recurrence	Solution with Master Method
$T(n) = 9T(n/3) + n$	Case-1: $T(n) = \Theta(n^2)$

Recap: Master method

When $T(n) = aT(n/b) + f(n)$ $\quad a \geq 1, b > 1$

Let $\epsilon > 0$ be a constant

1. If $f(n) = O(n^{log_b a - \epsilon})$ then $T(n) = \Theta(n^{log_b a})$
2. If $f(n) = \Theta(n^{log_b a})$ then $T(n) = \Theta(n^{log_b a} \log n)$
3. If $f(n) = \Omega(n^{log_b a + \epsilon})$ then $T(n) = \Theta(f(n))$
   provided if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n. **Regularity condition must be checked in case-3.**

Recurrence	Solution with Master Method
$T(n) = 9T(n/3) + n$	Case-1: $T(n) = \Theta(n^2)$
$T(n) = T(2n/3) + 1$	Case-2: $T(n) = \Theta(\log n)$

Recap: Master method

When $T(n) = aT(n/b) + f(n)$ $\quad a \geq 1, b > 1$

Let $\epsilon > 0$ be a constant

1. If $f(n) = O(n^{\log_b a - \epsilon})$ then $T(n) = \Theta(n^{\log_b a})$
2. If $f(n) = \Theta(n^{\log_b a})$ then $T(n) = \Theta(n^{\log_b a} \log n)$
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ then $T(n) = \Theta(f(n))$
   provided if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n. **Regularity condition must be checked in case-3.**

Recurrence	Solution with Master Method
$T(n) = 9T(n/3) + n$	Case-1: $T(n) = \Theta(n^2)$
$T(n) = T(2n/3) + 1$	Case-2: $T(n) = \Theta(\log n)$
$T(n) = 3T(n/4) + n \log n$	Case-3: $T(n) = \Theta(n \log n)$

Recap: Master method

When $T(n) = aT(n/b) + f(n)$ $\quad a \geq 1, b > 1$

Let $\epsilon > 0$ be a constant

1. If $f(n) = O(n^{\log_b a - \epsilon})$ then $T(n) = \Theta(n^{\log_b a})$
2. If $f(n) = \Theta(n^{\log_b a})$ then $T(n) = \Theta(n^{\log_b a} \log n)$
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ then $T(n) = \Theta(f(n))$
   provided if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n. **Regularity condition must be checked in case-3.**

Recurrence	Solution with Master Method
$T(n) = 9T(n/3) + n$	Case-1: $T(n) = \Theta(n^2)$
$T(n) = T(2n/3) + 1$	Case-2: $T(n) = \Theta(\log n)$
$T(n) = 3T(n/4) + n \log n$	Case-3: $T(n) = \Theta(n \log n)$
$T(n) = 2T(n/2) + n \log n$	Case-?: Falls in gap of case 2 & 3

Recap: Master method

When $T(n) = aT(n/b) + f(n)$ $\quad a \geq 1, b > 1$

Let $\epsilon > 0$ be a constant

1. If $f(n) = O(n^{\log_b a - \epsilon})$ then $T(n) = \Theta(n^{\log_b a})$
2. If $f(n) = \Theta(n^{\log_b a})$ then $T(n) = \Theta(n^{\log_b a} \log n)$
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ then $T(n) = \Theta(f(n))$
   provided if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n. **Regularity condition must be checked in case-3.**

Recurrence	Solution with Master Method
$T(n) = 9T(n/3) + n$	Case-1: $T(n) = \Theta(n^2)$
$T(n) = T(2n/3) + 1$	Case-2: $T(n) = \Theta(\log n)$
$T(n) = 3T(n/4) + n \log n$	Case-3: $T(n) = \Theta(n \log n)$
$T(n) = 2T(n/2) + n \log n$	Case-?: Falls in gap of case 2 & 3

Recap: Quick Sort

Which algorithm is better ?

	Best Case	Worst Case	Average Case
Algo-01	$n \log n$	$n \log n$	$n \log n$
Algo-02	$n \log n$	$n(n-1)$	$n \log n$

Recap: Quick Sort

Which algorithm is better ?

	Best Case	Worst Case	Average Case
Algo-01	$n \log n$	$n \log n$	$n \log n$
Algo-02	$n \log n$	$n(n-1)$	$n \log n$

- If I tell you Algo-01 is merge sort and Algo-02 is quick sort then?

Recap: Quick Sort

Which algorithm is better ?

	Best Case	Worst Case	Average Case
Algo-01	$n \log n$	$n \log n$	$n \log n$
Algo-02	$n \log n$	$n(n-1)$	$n \log n$

- If I tell you Algo-01 is merge sort and Algo-02 is quick sort then?
- Why quick sort is popular? it always behaves like average case as the number of items to be sorted sort increases

Recap: Quick Sort

Which algorithm is better ?

	Best Case	Worst Case	Average Case
Algo-01	$n \log n$	$n \log n$	$n \log n$
Algo-02	$n \log n$	$n(n-1)$	$n \log n$

- If I tell you Algo-01 is merge sort and Algo-02 is quick sort then?
- Why quick sort is popular? it always behaves like average case as the number of items to be sorted sort increases
- 1000 time execution of randomized quick sort on randomly selected items

	Number of items $n =$				
Number of times the runtime exceed average behavior	10^2	10^3	10^4	10^5	10^6
10%	190	49	22	10	3
20%	28	17	12	3	0
50%	2	1	1	0	0
100%	0	0	0	0	0

Execution of Quick Sort

```
QUICKSORT(A, p, r)
1  if p < r
2        q = PARTITION(A, p, r)
3        QUICKSORT(A, p, q − 1)
4        QUICKSORT(A, q + 1, r)
```

```
PARTITION(A, p, r)
1  x = A[r]
2  i = p − 1
3  for j = p to r − 1
4       if A[j] ≤ x
5            i = i + 1
6            exchange A[i] with A[j]
7  exchange A[i + 1] with A[r]
8  return i + 1
```

Randomized Quick Sort

```
QUICKSORT(A, p, r)
1  if p < r
2      q = PARTITION(A, p, r)
3      QUICKSORT(A, p, q − 1)
4      QUICKSORT(A, q + 1, r)
```

```
PARTITION(A, p, r)
1  x = A[r]
2  i = p − 1
3  for j = p to r − 1
4      if A[j] ≤ x
5          i = i + 1
6          exchange A[i] with A[j]
7  exchange A[i + 1] with A[r]
8  return i + 1
```

```
RANDOMIZED-QUICKSORT(A, p, r)
1  if p < r
2      q = RANDOMIZED-PARTITION(A, p, r)
3      RANDOMIZED-QUICKSORT(A, p, q − 1)
4      RANDOMIZED-QUICKSORT(A, q + 1, r)
```

```
RANDOMIZED-PARTITION(A, p, r)
1  i = RANDOM(p, r)
2  exchange A[r] with A[i]
3  return PARTITION(A, p, r)
```

Analysis of Randomized Quick Sort

- We need to estimate number of comparisons performed during its execution

Analysis of Randomized Quick Sort

- We need to estimate number of comparisons performed during its execution
- Let the sorted list of items be $< S_1, S_2, S_3, ..., S_n >$ with S_i being i^{th} smallest element

Analysis of Randomized Quick Sort

- We need to estimate number of comparisons performed during its execution
- Let the sorted list of items be $< S_1, S_2, S_3, ..., S_n >$ with S_i being i^{th} smallest element
- Define a random variable X_{ij} to be the number of comparisons between S_i and S_j

Analysis of Randomized Quick Sort

- We need to estimate number of comparisons performed during its execution
- Let the sorted list of items be $< S_1, S_2, S_3, ..., S_n >$ with S_i being i^{th} smallest element
- Define a random variable X_{ij} to be the number of comparisons between S_i and S_j
- X_{ij} can either take a value 0 or 1

Analysis of Randomized Quick Sort

- We need to estimate number of comparisons performed during its execution
- Let the sorted list of items be $< S_1, S_2, S_3, ..., S_n >$ with S_i being i^{th} smallest element
- Define a random variable X_{ij} to be the number of comparisons between S_i and S_j
- X_{ij} can either take a value 0 or 1
- Expected number of comparison is

$$E[\sum_{i=1}^{n}\sum_{j>i} X_{ij}] = \sum_{i=1}^{n}\sum_{j>i} E[X_{ij}]$$

Randomized Quick Sort

- Let p_{ij} denote the probability of comparison between S_i and S_j. Then,

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

Randomized Quick Sort

- Let p_{ij} denote the probability of comparison between S_i and S_j. Then,

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

- Pivot element can either be
  1. S_i or S_j: its probability is $\frac{2}{j-i+1}$

Randomized Quick Sort

- Let p_{ij} denote the probability of comparison between S_i and S_j. Then,

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

- Pivot element can either be
  1. S_i or S_j: its probability is $\frac{2}{j-i+1}$
  2. S_q from $i < q < j$

Randomized Quick Sort

- Let p_{ij} denote the probability of comparison between S_i and S_j. Then,

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

- Pivot element can either be
  1. S_i or S_j: its probability is $\frac{2}{j-i+1}$
  2. S_q from $i < q < j$
  3. S_r from $r < i$ or $j < r$

Randomized Quick Sort

- Let p_{ij} denote the probability of comparison between S_i and S_j. Then,

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

$$\sum_{i=1}^{n} \sum_{j>i} E[X_{ij}] = \sum_{i=1}^{n} \sum_{j>i} p_{ij}$$

- Pivot element can either be
  1. S_i or S_j: its probability is $\frac{2}{j-i+1}$
  2. S_q from $i < q < j$
  3. S_r from $r < i$ or $j < r$

Randomized Quick Sort

- Let p_{ij} denote the probability of comparison between S_i and S_j. Then,

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

$$\begin{aligned} \sum_{i=1}^{n} \sum_{j>i} E[X_{ij}] &= \sum_{i=1}^{n} \sum_{j>i} p_{ij} \\ &= \sum_{i=1}^{n} \sum_{j>i} \frac{2}{j-i+1} \end{aligned}$$

- Pivot element can either be
  1. S_i or S_j: its probability is $\frac{2}{j-i+1}$
  2. S_q from $i < q < j$
  3. S_r from $r < i$ or $j < r$

Randomized Quick Sort

- Let p_{ij} denote the probability of comparison between S_i and S_j. Then,

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

$$\begin{aligned}\sum_{i=1}^{n}\sum_{j>i} E[X_{ij}] &= \sum_{i=1}^{n}\sum_{j>i} p_{ij} \\ &= \sum_{i=1}^{n}\sum_{j>i} \frac{2}{j-i+1} \\ &= 2\sum_{i=1}^{n}\sum_{k=1}^{n-i+1} \frac{1}{k}\end{aligned}$$

- Pivot element can either be
  1. S_i or S_j: its probability is $\frac{2}{j-i+1}$
  2. S_q from $i < q < j$
  3. S_r from $r < i$ or $j < r$

Randomized Quick Sort

- Let p_{ij} denote the probability of comparison between S_i and S_j. Then,

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

- Pivot element can either be
  1. S_i or S_j: its probability is $\frac{2}{j-i+1}$
  2. S_q from $i < q < j$
  3. S_r from $r < i$ or $j < r$

$$\begin{aligned}
\sum_{i=1}^{n} \sum_{j>i} E[X_{ij}] &= \sum_{i=1}^{n} \sum_{j>i} p_{ij} \\
&= \sum_{i=1}^{n} \sum_{j>i} \frac{2}{j-i+1} \\
&= 2 \sum_{i=1}^{n} \sum_{k=1}^{n-i+1} \frac{1}{k} \\
&\leq 2 \sum_{i=1}^{n} \sum_{k=1}^{n} \frac{1}{k}
\end{aligned}$$

Randomized Quick Sort

- Let p_{ij} denote the probability of comparison between S_i and S_j. Then,

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

- Pivot element can either be
  1. S_i or S_j: its probability is $\frac{2}{j-i+1}$
  2. S_q from $i < q < j$
  3. S_r from $r < i$ or $j < r$

$$
\begin{aligned}
\sum_{i=1}^{n} \sum_{j>i} E[X_{ij}] &= \sum_{i=1}^{n} \sum_{j>i} p_{ij} \\
&= \sum_{i=1}^{n} \sum_{j>i} \frac{2}{j-i+1} \\
&= 2 \sum_{i=1}^{n} \sum_{k=1}^{n-i+1} \frac{1}{k} \\
&\leq 2 \sum_{i=1}^{n} \sum_{k=1}^{n} \frac{1}{k} \\
&= 2nH_n
\end{aligned}
$$

Randomized Quick Sort

- Let p_{ij} denote the probability of comparison between S_i and S_j. Then,

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

- Pivot element can either be
  1. S_i or S_j: its probability is $\frac{2}{j-i+1}$
  2. S_q from $i < q < j$
  3. S_r from $r < i$ or $j < r$

$$
\begin{aligned}
\sum_{i=1}^{n} \sum_{j>i} E[X_{ij}] &= \sum_{i=1}^{n} \sum_{j>i} p_{ij} \\
&= \sum_{i=1}^{n} \sum_{j>i} \frac{2}{j-i+1} \\
&= 2 \sum_{i=1}^{n} \sum_{k=1}^{n-i+1} \frac{1}{k} \\
&\leq 2 \sum_{i=1}^{n} \sum_{k=1}^{n} \frac{1}{k} \\
&= 2nH_n \\
&= O(n \ln n)
\end{aligned}
$$

as $H_n \sim \ln n + \Theta(1)$

Thank You!

Thank you very much for your attention!

Queries ?