

Part I: Divide and Conquer

Lecture 4: Randomized Partition and Randomized Selection

Ahmed Khademzadeh
<http://khademzadeh.mshdiau.ac.ir>
Azad University of Mashhad



The slides are borrowed from Ke (Kevin) Yi. Thanks to him for his benevolence.

Objective and Outline

Objective: Show two examples that use both (1) divide and conquer and (2) randomization

Reference: Chapter 7, 9, Appendix C (probability) of CLRS

Outline:

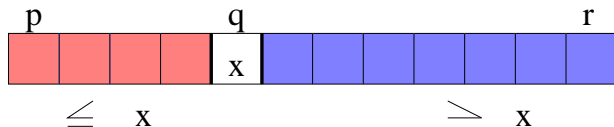
- Partition
 - Basic partition
 - Randomized partition and randomized quicksort
 - Analysis of the randomized quicksort
- Selection
 - The selection problem
 - First solution: Selection by sorting
 - A divide-and-conquer algorithm
 - Analysis of the divide-and-conquer algorithm

Partition

Given: An array of numbers

Partition: Rearrange the array $A[p..r]$ **in place** into two (possibly empty) subarrays $A[p..q-1]$ and $A[q+1..r]$ such that

$$A[u] < A[q] < A[v], \quad \text{for any } p \leq u \leq q-1 \text{ and } q+1 \leq v \leq r$$



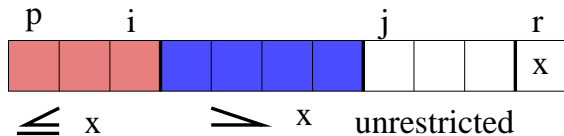
$$x = A[r]$$

x is called the **pivot**. Assume $x = A[r]$; if not, swap first
Quicksort works by:

- 1 calling partition first
- 2 recursively sorting $A[p..q-1]$ and $A[q+1..r]$

The Idea of Partition(A, p, r)

Use $A[r]$ as the **pivot**, and grow partition from left to right

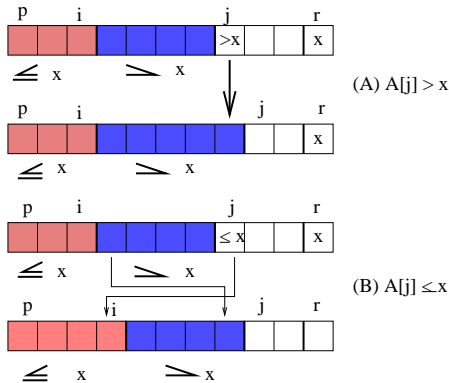


- 1 Initially $(i, j) = (p - 1, p)$
- 2 Increase j by 1 each time to find a place for $A[j]$
At the same time increase i when necessary
- 3 Stops when $j = r$

One Iteration of the Procedure Partition

Increase j by 1 each time to find a place for $A[j]$

At the same time increase i when necessary



- 1 Only increase j by 1
- 2 $i \leftarrow i + 1$. $A[i] \leftrightarrow A[j]$. $j \leftarrow j + 1$

Example: The Operation of Partition(A, p, r)

i p, j r

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

 (1)

p, i j r

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

 (2)

p, i j r

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

 (3)

p, i j r

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

 (4)

p i j r

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

 (5)

p i j r

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

 (6)

p i j r

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

 (7)

p i j, r

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

 (8)

p i j, r

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

 (9)

The Partition(A, p, r) Algorithm

Partition(A, p, r)

```
begin
  x = A[r]; // A[r] is the pivot element
  i = p-1;
  for j = p to r-1 do
    if A[j] ≤ x then
      i = i+1;
      exchange A[i] and A[j];
    end
  end
  exchange A[i+1] and A[r]; // put pivot in position
  return i+1 // q = i+1
end
```

Running Time of Partition(A, p, r)

Partition(A, p, r):

$x = A[r]$

$i = p - 1$

for $j = p$ to $r - 1$

if $A[j] \leq x$

$(r - p)$

$i = i + 1$

exchange $A[i] \leftrightarrow A[j]$

exchange $A[i + 1] \leftrightarrow A[r]$

return $i + 1$

Total: $(r - p)$

Running time is $O(r - p)$

- **linear** in the length of the array $A[p..r]$

Quicksort(A, p, r)

```
begin
  if  $p < r$  then
     $q = \text{Partition}(A, p, r)$ ;
    Quicksort(A, p,  $q-1$ );
    Quicksort(A,  $q+1$ , r);
  end
end
```

- If we could always partition the array into halves, then we have the recurrence $T(n) \leq 2T(n/2) + O(n)$, hence $T(n) = O(n \log n)$
- However, if we always get unlucky with very unbalanced partitions, then $T(n) \leq T(n-1) + O(n)$, hence $T(n) = O(n^2)$

Outline:

- Partition
- Randomized partition and randomized quicksort
- Analysis of the randomized quicksort
- The selection problem
- First solution: Selection by sorting
- A divide-and-conquer algorithm
- Analysis of the divide-and-conquer algorithm

Randomized-Partition(A, p, r)

Idea:

- In the algorithm Partition(A, p, r), $A[r]$ is always used as the pivot x to partition the array $A[p..r]$
- In the algorithm Randomized-Partition(A, p, r), we randomly choose an j , $p \leq j \leq r$, and use $A[j]$ as pivot
- Idea is that if we choose randomly, then the chance that we get unlucky every time is extremely low.



Randomized-Partition(A, p, r)...

Let $\text{random}(p,r)$ be a pseudorandom-number generator that returns a random number between p and r

$\text{Randomized-Partition}(A, p, r)$

```
begin  
  |  $j = \text{random}(p, r);$   
  |  $\text{exchange } A[r] \text{ and } A[j];$   
  |  $\text{Partition}(A, p, r);$   
end
```

Randomized-Quicksort Algorithm

We make use of the Randomized-Partition idea to develop a new version of quicksort

Randomized-Quicksort(A, p, r)

```
begin
  if  $p < r$  then
     $q = \text{Randomized-Partition}(A, p, r)$ ;
    Randomized-Quicksort( $A, p, q-1$ );
    Randomized-Quicksort( $A, q+1, r$ );
  end
end
```

Outline:

- Partition
- Randomized partition and randomized quicksort
- Analysis of the randomized quicksort
- The selection problem
- First solution: Selection by sorting
- A divide-and-conquer algorithm
- Analysis of the divide-and-conquer algorithm

Running Time of the Randomized-Quicksort

Measuring running time:

- The running time is dominated by the time spent in partition.
- The running time of the partition procedure can be measured by the **number of key comparisons**.

$T(n)$: running time on array of size n .

Recurrence: $T(n) = T(m) + T(n - m - 1) + O(n)$

Worst Case:

$$T(n) = T(0) + T(n - 1) + O(n)$$

$$T(n) = O(n^2)$$

What inputs give worst case performance?

- Whether performance is the worst is not determined by input.
- An important property of randomized algorithms.
- Worst case performance results only if the random number generator always produces the worst choice.

Analysis for Randomized Algorithms

- 1 Worst-case doesn't make sense: for any given input, the worst case is very unlikely to happen
- 2 Use **expected running time** analysis for randomized algorithms!

Average case analysis

- Used for deterministic algorithms
- Assume the input is chosen randomly from some distribution
- Depends on assumptions on the input, weaker
- Not required in this course

Expected case analysis

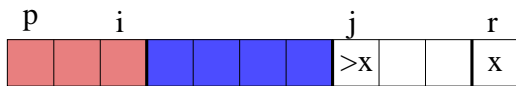
- Used for randomized algorithms
- Need to work for **any** given input
- Randomization is inherent within the algorithm, stronger
- Required in this course

Two methods to analyze the expected running time of a divide-and-conquer randomized algorithm:

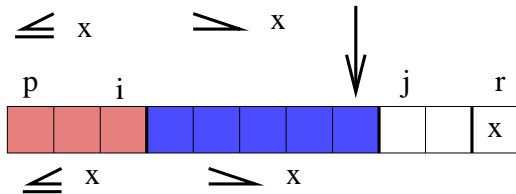
- 1 Old fashioned: Write out a recurrence on $T(n)$, where $T(n)$ is the **expected** running time of the algorithm on an input of size n , and solve it.
(Almost) always works but needs complicated maths.
- 2 New: Indicator variables.
Simple and elegant, but needs practice to master.

Two facts about key comparisons:

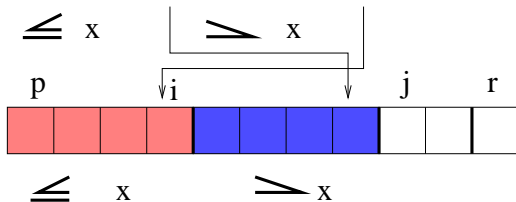
- 1 When a pivot is selected, the pivot is compared with **every** other elements, then the elements are partitioned into two parts accordingly
- 2 Elements in **different** partitions are **never** compared with each other in **all** operations



(A) $A[j] > x$



(B) $A[j] \leq x$



Expected Running Time of Randomized-Quicksort

- Let $z_1 < z_2 < \dots < z_n$ be the n elements in sorted order
- X : total number of comparisons performed in **all** calls to randomized-partition
- X_{ij} : number of comparisons between z_i and z_j
 - can only be **0 or 1**

$$\begin{aligned} E[X] &= E \left[\sum_{i=1}^{n-1} \sum_{j=i+1}^n X_{ij} \right] = \sum_{i=1}^{n-1} \sum_{j=i+1}^n E[X_{ij}] \\ &= \sum_{i=1}^{n-1} \sum_{j=i+1}^n [Pr\{z_i \text{ is compared with } z_j\} \times 1 \\ &\quad + Pr\{z_i \text{ is not compared with } z_j\} \times 0] \\ &= \sum_{i=1}^{n-1} \sum_{j=i+1}^n Pr\{z_i \text{ is compared with } z_j\} \end{aligned}$$

Key Observations on Finding $Pr\{z_i \text{ is compared with } z_j\}$

For $1 \leq i \leq j \leq n$, let $Z_{ij} = \{z_i, z_{i+1}, \dots, z_j\}$

- remember $z_i < z_{i+1} < \dots < z_j$

Observations:

- 1 Partition divides an array into three segments, left, pivot, and right.
- 2 When z_i and z_j are first placed in DIFFERENT segments of the array by partitioning, the pivot is one of the elements in Z_{ij}
- 3 If the pivot is either z_i or z_j
 - z_i and z_j will be compared
- 4 If the pivot is any element in Z_{ij} other than z_i or z_j
 - z_i and z_j are **not** compared with each other in **all** randomized-partition calls

How to Find $Pr\{z_i \text{ is compared with } z_j\}$?

$Pr\{z_i \text{ is compared with } z_j\}$

$$= Pr\{z_i \text{ or } z_j \text{ is the first pivot chosen from } Z_{ij}\}$$

$$= Pr\{z_i \text{ is the first pivot chosen from } Z_{ij}\}$$

$$+ Pr\{z_j \text{ is the first pivot chosen from } Z_{ij}\}$$

$$= \frac{1}{j-i+1} + \frac{1}{j-i+1} = \frac{2}{j-i+1}$$

$$\begin{aligned} E[X] &= \sum_{i=1}^{n-1} \sum_{j=i+1}^n Pr\{z_i \text{ is compared with } z_j\} = \sum_{i=1}^{n-1} \sum_{j=i+1}^n \frac{2}{j-i+1} \\ &= \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{2}{k+1} < \sum_{i=1}^{n-1} \sum_{k=1}^n \frac{2}{k} = \sum_{i=1}^{n-1} O(\log n) = O(n \log n) \end{aligned}$$

NOTE: $\sum_{k=1}^n \frac{1}{k} \leq \log(n)$

Hence, the expected number of comparisons is $O(n \log n)$, which is the expected running time of Randomized-Quicksort

Outline:

- Partition
- Randomized partition and randomized quicksort
- Analysis of the randomized quicksort
- The selection problem
- First solution: Selection by sorting
- A divide-and-conquer algorithm
- Analysis of the divide-and-conquer algorithm

Linear Time Selection

Definition (Selection Problem)

Given a sequence of numbers $\langle a_1, \dots, a_n \rangle$, and an integer i , $1 \leq i \leq n$, find the i th smallest element. When $i = \lceil n/2 \rceil$, it is called the median problem.

Example

Given $\langle 1, 8, 23, 10, 19, 33, 100 \rangle$, the 4th smallest element is 19.

Question

How do you solve this problem?

Outline:

- Partition
- Randomized partition and randomized quicksort
- Analysis of the randomized quicksort
- The selection problem
- First solution: Selection by sorting
- A divide-and-conquer algorithm
- Analysis of the divide-and-conquer algorithm

First Solution: Selection by Sorting

- 1 Sort the elements in ascending order with any algorithm of complexity $O(n \log n)$.
- 2 Return the i th element of the sorted array.

The complexity of this solution is $O(n \log n)$

Question

Can we do better?

Answer: YES, but we need to recall $\text{Partition}(A, p, r)$ used in Quicksort!

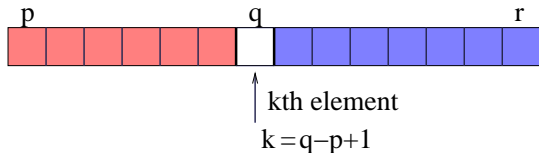
Outline:

- Partition
- Randomized partition and randomized quicksort
- Analysis of the randomized quicksort
- The selection problem
- First solution: Selection by sorting
- A divide-and-conquer algorithm
- Analysis of the divide-and-conquer algorithm

Randomized-Select(A, p, r, i), $1 \leq i \leq r - p + 1$

Problem: Select the i th smallest element in $A[p..r]$, where $1 \leq i \leq r - p + 1$

Solution: Apply Randomized-Partition(A, p, r), getting



- 1 $i = k$
 - pivot is the solution
- 2 $i < k$
 - the i th smallest element in $A[p..r]$ must be the i th smallest element in $A[p..q - 1]$
- 3 $i > k$
 - the i th smallest element in $A[p..r]$ must be the $(i - k)$ th smallest element in $A[q + 1..r]$

If necessary, **recursively** call the same procedure to the subarray

Randomized-Select(A, p, r, i), $1 \leq i \leq r - p + 1$

```
if  $p=r$  then
  | return  $A[p]$ 
end
 $q = \text{Randomized-Partition}(A, p, r)$  ;
 $k = q - p + 1$  ;
if  $i = k$  then return  $A[q]$  // the pivot is the answer
else if  $i < k$  then
  | return Randomized-Select( $A, p, q - 1, i$ )
else
  | return Randomized-Select( $A, q + 1, r, i - k$ )
end
```

To find the i th smallest element in $A[1..n]$, call
Randomized-Select($A, 1, n, i$)

Outline:

- Partition
- Randomized partition and randomized quicksort
- Analysis of the randomized quicksort
- The selection problem
- First solution: Selection by sorting
- A divide-and-conquer algorithm
- Analysis of the divide-and-conquer algorithm

Running Time of Randomized-Select($A, 1, n, i$)

$T(n)$: upper bound on the **expected** number of comparisons made by Randomized-Select($A, 1, n, i$) for any i

$$T(1) = 0$$

For $n > 1$, we get

$$T(n) \leq n + \sum_{k=1}^n \left(\frac{1}{n} \cdot T(\max\{k-1, n-k\}) \right)$$

initial partition
recursion, assume the bad case

$$T(n) \leq n + \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} T(k)$$

It is a complicated recurrence!

Use the second method: guess & induction.

Guess:

$$T(n) \leq c n, \quad \text{for all } n$$

for some constant c to be figured out later.

Proof that $T(n) \leq cn$

Induction step: Assume that $T(m) \leq cm$ for all $m \leq n - 1$. Then try to show $T(n) \leq cn$:

$$\begin{aligned} T(n) &\leq n + \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} T(k) \\ &\leq n + \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} ck \\ &\quad \dots \text{(on board; and see textbook)} \\ &\leq \frac{3c}{4}n + \frac{c}{2} + n \end{aligned}$$

We want $\frac{3c}{4}n + \frac{c}{2} + n \leq cn$, or $n \geq \frac{2c}{c-4}$.

If we choose $c \geq 12$. Then the induction step works for $n \geq 3$.

Induction basis: $T(1) \leq c \cdot 1$, $T(2) \leq c \cdot 2$.

So if we choose $c = \max\{12, T(1), T(2)/2\}$, then the entire proof works.

Running Time of Randomized-Select($A, 1, n, i$)

Worst Case:

$$T(n) = n - 1 + T(n - 1), T(n) = O(n^2).$$

Expected running time much better than worst case!

Randomized Quicksort vs Randomized Selection

Question

Why does Randomized Selection take $O(n)$ time while Randomized Quicksort takes $O(n \log n)$ time?

Answer:

- Randomized Selection needs to work on only **1** of the two subproblems.
- Randomize Quicksort needs to work on **both** of the two subproblems.

Summary of Divide-and-Conquer and Randomization

- Randomization is needed when it is not easy to divide evenly
- Use expected case analysis for randomized algorithms.
- The running time is the **expected** running time for **any** given input, expectation is with respect to the random choices made by the algorithm internally

Question

Is it possible to not use randomization in quicksort and selection?
(Next time ...)

How do we generate a random number?

Dice, coin flipping, roulette wheels, ...

How does a computer generate a random number?

- By hardware: electronic noise, thermal noise, etc. Expensive but “true” random numbers in some sense
- By software: pseudorandom numbers. A long sequence of seemingly random numbers whose pattern is difficult to find
- Pseudorandom numbers are good enough for most applications