COMS21103: Order Statistics

Dima Damen
Dima.Damen@bristol.ac.uk

Bristol University, Department of Computer Science
Bristol BS8 1UB, UK
Based on slides of Raphael Clifford

November 23, 2015

Order Statistics

- In this lecture we look at order statistics.
  - Select the ith smallest of n elements (the element of rank i).
  - If $i = 1$ we have the minimum
  - If $i = n$, we have the maximum
  - If $i = \lfloor (n+1)/2 \rfloor$ or $\lceil (n+1)/2 \rceil$, then we have the median
- A simple solution is just to sort the input and choose the ith element of the sorted array.
- Worst case is $\Theta(n \log n)$ time using Merge Sort or Heap Sort for example.
- We can do better than that!

Partitioning revisited

- Recall... for Quicksort we used a random partitioning that was able to take a pivot x and rearrange the input array A so that all values $\leq x$ are to the left of x and those $\geq x$ are to right of x.

Partitioning revisited

- Recall... for Quicksort we used a random partitioning that was able to take a pivot x and rearrange the input array A so that all values $\leq x$ are to the left of x and those $\geq x$ are to right of x.

- Assume $\textsc{Rand-Partition}(A, p, q)$ partitions $A[p, \ldots, q]$ according to a randomly chosen pivot and returns r, the index of the pivot.

Partitioning revisited

- Recall... for Quicksort we used a random partitioning that was able to take a pivot x and rearrange the input array A so that all values $\leq x$ are to the left of x and those $\geq x$ are to right of x.

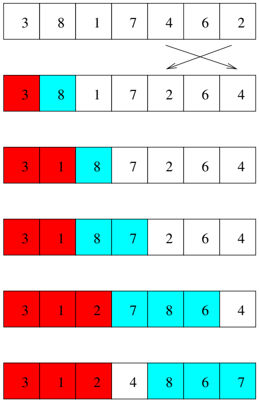
- Assume RAND-PARTITION(A, p, q) partitions $A[p, \ldots, q]$ according to a randomly chosen pivot and returns r, the index of the pivot.
- First we have to choose a random pivot and then call PARTITION with the pivot placed at end of the input array.

Input: A, p, q
$i \leftarrow$ RANDOM(p, q);
swap $A[q]$ and $A[i]$;
return PARTITION(A, p, q);

Partitioning revisited

PARTITION(A, p, q) returns the index of the pivot element after partitioning.

Input : A, p, q
Output: Index of pivot
$x \leftarrow A[q]$ $\triangleright$ x is the pivot;
$i \leftarrow p - 1$;
for $j \leftarrow p$ **to** $q - 1$ **do**
  if $A[j] \leq x$ **then**
    $i \leftarrow i + 1$;
    swap $A[i]$ and $A[j]$;
  end
end
swap $A[i + 1]$ and $A[q]$;
return $i + 1$;



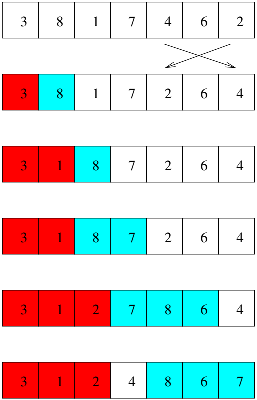
Dima Damen
Dima.Damen@bristol.ac.uk

Partitioning revisited

PARTITION(A, p, q) returns the index of the pivot element after partitioning.

Input : A, p, q
Output: Index of pivot

$x \leftarrow A[q]$ $\quad \triangleright$ x is the pivot;
$i \leftarrow p - 1$;
for $j \leftarrow p$ **to** $q - 1$ **do**
    if $A[j] \leq x$ **then**
        $i \leftarrow i + 1$;
        swap $A[i]$ and $A[j]$;
    end
end
swap $A[i + 1]$ and $A[q]$;
return $i + 1$;



- At the start of every iteration, the following *invariants* are true
  - For $p \leq k \leq i$, $A[k] \leq x$
  - For $i + 1 \leq k < j$, $A[k] > x$
  - $A[q] = x$

Randomised algorithm

We can use the partition function repeatedly to find the *i*th smallest element.

- Pick a pivot at random
- Partition according to the pivot
- Now we only need to look in one of the two "halves" after the partitioning
- Recurse until either we happen to choose the *i*th smallest element as a pivot or the half we need to look in only has one element in it

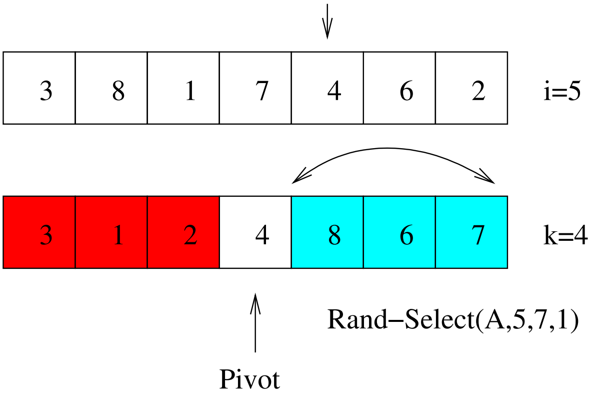
Randomised algorithm

RAND-SELECT(A, p, q, i) returns the ith smallest element of $A[p, \ldots, q]$

Input : A, p, q and i
Output: The ith smallest value in $A[p, \ldots, q]$
if $p = q$ **then**
    return $A[p]$;
end
$r \leftarrow$ RAND-PARTITION(A, p, q);
$k \leftarrow r - p + 1$ $\quad\triangleright$ pivot index in $A[p, \ldots, q]$;
if $i = k$ **then**
    return $A[r]$;
end
if $i < k$ **then**
    return RAND-SELECT$(A, p, r - 1, i)$;
else
    return RAND-SELECT$(A, r + 1, q, i - k)$;
end

Order Statistics Example

In this example we want to find the 5th smallest number in the array
3, 8, 1, 7, 4, 6, 2.



Running Time - Intuition

In the worst case

Running Time - Intuition

In the worst case we might partition the array with one element on one side and all the others on the other side. (We assume all elements are distinct to make life easy for ourselves.)

Running Time - Intuition

In the worst case we might partition the array with one element on one side and all the others on the other side. (We assume all elements are distinct to make life easy for ourselves.)

- $T(n) = \Theta(n) + T(n-1)$

Running Time - Intuition

In the worst case we might partition the array with one element on one side and all the others on the other side. (We assume all elements are distinct to make life easy for ourselves.)

- $T(n) = \Theta(n) + T(n-1)$
- Therefore $T(n) = \Theta(\quad)$

Running Time - Intuition

In the worst case we might partition the array with one element on one side and all the others on the other side. (We assume all elements are distinct to make life easy for ourselves.)

- $T(n) = \Theta(n) + T(n-1)$
- Therefore $T(n) = \Theta(n^2)$. Very unlucky and worse than sorting!

If are very lucky...

Running Time - Intuition

In the worst case we might partition the array with one element on one side and all the others on the other side. (We assume all elements are distinct to make life easy for ourselves.)

- $T(n) = \Theta(n) + T(n-1)$
- Therefore $T(n) = \Theta(n^2)$. Very unlucky and worse than sorting!

If are very lucky... we would partition the array exactly in half each time

Running Time - Intuition

In the worst case we might partition the array with one element on one side and all the others on the other side. (We assume all elements are distinct to make life easy for ourselves.)

- $T(n) = \Theta(n) + T(n-1)$
- Therefore $T(n) = \Theta(n^2)$. Very unlucky and worse than sorting!

If are very lucky... we would partition the array exactly in half each time

- $T(n) = T(n/2) + \Theta(n)$

Running Time - Intuition

In the worst case we might partition the array with one element on one side and all the others on the other side. (We assume all elements are distinct to make life easy for ourselves.)

- $T(n) = \Theta(n) + T(n-1)$
- Therefore $T(n) = \Theta(n^2)$. Very unlucky and worse than sorting!

If are very lucky... we would partition the array exactly in half each time

- $T(n) = T(n/2) + \Theta(n)$

Theorem

Master Theorem - Case 3
Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function and $T(n)$ be defined on the nonnegative integers by the recurrence

$$T(n) = aT(n/b) + f(n) \tag{1}$$

If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$ and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$

Running Time - Intuition

In the worst case we might partition the array with one element on one side and all the others on the other side. (We assume all elements are distinct to make life easy for ourselves.)

- $T(n) = \Theta(n) + T(n-1)$
- Therefore $T(n) = \Theta(n^2)$. Very unlucky and worse than sorting!

If are very lucky... we would partition the array exactly in half each time

- $T(n) = T(n/2) + \Theta(n)$
- Case 3 of the Master Theorem as $cn \in \Omega(n^{\log_2 1}) = \Omega(n^0) = \Omega(1)$.

Theorem

Master Theorem - Case 3

Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function and $T(n)$ be defined on the nonnegative integers by the recurrence

$$T(n) = aT(n/b) + f(n) \tag{1}$$

If $f(n) = \Omega(n^{log_b a+\epsilon})$ for some constant $\epsilon > 0$ and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$

Running Time - Intuition

In the worst case we might partition the array with one element on one side and all the others on the other side. (We assume all elements are distinct to make life easy for ourselves.)

- $T(n) = \Theta(n) + T(n-1)$
- Therefore $T(n) = \Theta(n^2)$. Very unlucky and worse than sorting!

If are very lucky we would partition the array exactly in half each time

- $T(n) = T(n/2) + \Theta(n)$

Running Time - Intuition

In the worst case we might partition the array with one element on one side and all the others on the other side. (We assume all elements are distinct to make life easy for ourselves.)

- $T(n) = \Theta(n) + T(n-1)$
- Therefore $T(n) = \Theta(n^2)$. Very unlucky and worse than sorting!

If are very lucky we would partition the array exactly in half each time

- $T(n) = T(n/2) + \Theta(n)$
- Therefore $T(n) = \Theta(n)$. This is very different from the worst case!

Running Time - Intuition

In the worst case we might partition the array with one element on one side and all the others on the other side. (We assume all elements are distinct to make life easy for ourselves.)

- $T(n) = \Theta(n) + T(n-1)$
- Therefore $T(n) = \Theta(n^2)$. Very unlucky and worse than sorting!

If are very lucky we would partition the array exactly in half each time

- $T(n) = T(n/2) + \Theta(n)$
- Therefore $T(n) = \Theta(n)$. This is very different from the worst case!
- We don't need to be that lucky to get linear time
  - For example, if $T(n) = T(99n/100) + \Theta(n)$

Running Time - Intuition

In the worst case we might partition the array with one element on one side and all the others on the other side. (We assume all elements are distinct to make life easy for ourselves.)

- $T(n) = \Theta(n) + T(n-1)$
- Therefore $T(n) = \Theta(n^2)$. Very unlucky and worse than sorting!

If are very lucky we would partition the array exactly in half each time

- $T(n) = T(n/2) + \Theta(n)$
- Therefore $T(n) = \Theta(n)$. This is very different from the worst case!
- We don't need to be that lucky to get linear time
  - For example, if $T(n) = T(99n/100) + \Theta(n)$ then we still have $T(n) = \Theta(n)$

Running Time - Intuition

In the worst case we might partition the array with one element on one side and all the others on the other side. (We assume all elements are distinct to make life easy for ourselves.)

- $T(n) = \Theta(n) + T(n-1)$
- Therefore $T(n) = \Theta(n^2)$. Very unlucky and worse than sorting!

If are very lucky we would partition the array exactly in half each time

- $T(n) = T(n/2) + \Theta(n)$
- Therefore $T(n) = \Theta(n)$. This is very different from the worst case!
- We don't need to be that lucky to get linear time
  - For example, if $T(n) = T(99n/100) + \Theta(n)$ then we still have $T(n) = \Theta(n)$
  - In fact, the average case is also linear time and the method is very fast in practice.

Worst case linear time order statistics

The problem with the randomised approach is that we might choose bad pivots. Blum, Floyd, Pratt, Rivest, and Tarjan [1973] found a way to guarantee a good pivot each time.

Worst case linear time order statistics

The problem with the randomised approach is that we might choose bad pivots. Blum, Floyd, Pratt, Rivest, and Tarjan [1973] found a way to guarantee a good pivot each time.

SELECT(A, p, q, i)

1. Divide the $n = q - p + 1$ elements into groups of 5. Find the median of each 5-element group.

Worst case linear time order statistics

The problem with the randomised approach is that we might choose bad pivots. Blum, Floyd, Pratt, Rivest, and Tarjan [1973] found a way to guarantee a good pivot each time.

SELECT(A, p, q, i)

1. Divide the $n = q - p + 1$ elements into groups of 5. Find the median of each 5-element group.
2. Recursively SELECT the median x of the $\lfloor n/5 \rfloor$ group medians to be the pivot.

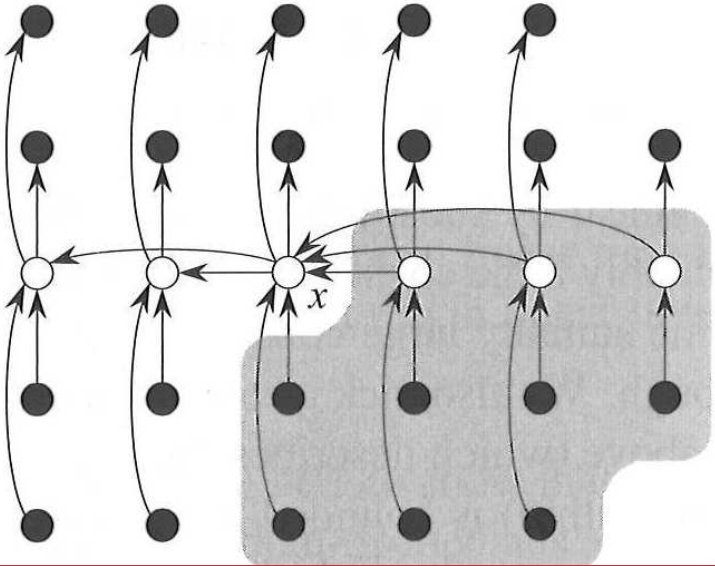
Worst case linear time order statistics

The problem with the randomised approach is that we might choose bad pivots. Blum, Floyd, Pratt, Rivest, and Tarjan [1973] found a way to guarantee a good pivot each time.

$\text{SELECT}(A, p, q, i)$

1. Divide the $n = q - p + 1$ elements into groups of 5. Find the median of each 5-element group.
2. Recursively SELECT the median x of the $\lfloor n/5 \rfloor$ group medians to be the pivot.
3. Partition $A[p, \ldots, q]$ using the pivot x. $k = \text{rank}(x)$
4. if $i = k$ then return x
5. elsif $i < k$
    - then recursively SELECT the ith smallest element in the lower part
    - else recursively SELECT the $i - k$th smallest element in the higher part

BFPRT Pivot Method



BFPRT Pivot Method - Example

A = [2, 3, 1, 7, 2, 8, 3, 9, 4, 12, 15, 3, 5, 7, 1]

BFPRT Pivot Method - Example

A = [2, 3, 1, 7, 2, 8, 3, 9, 4, 12, 15, 3, 5, 7, 1]

- Split into groups of 5 and find the median
  [2, 3, 1, 7, 2] $\rightarrow$ Median 2
  [8, 3, 9, 4, 12] $\rightarrow$ Median 8
  [15, 3, 5, 7, 1] $\rightarrow$ Median 5

BFPRT Pivot Method - Example

A = [2, 3, 1, 7, 2, 8, 3, 9, 4, 12, 15, 3, 5, 7, 1]

- Split into groups of 5 and find the median
  [2, 3, 1, 7, 2] $\rightarrow$ Median 2
  [8, 3, 9, 4, 12] $\rightarrow$ Median 8
  [15, 3, 5, 7, 1] $\rightarrow$ Median 5
- Partition using median and re-order groups
  [2, 1, 2, 7, 3]
  [1, 3, 5, 7, 15]
  [3, 4, 8, 9, 12]

Dima Damen
Dima.Damen@bristol.ac.uk

BFPRT Pivot Method - Example

A = [2, 3, 1, 7, 2, 8, 3, 9, 4, 12, 15, 3, 5, 7, 1]

- Split into groups of 5 and find the median
  [2, 3, 1, 7, 2] $\rightarrow$ Median 2
  [8, 3, 9, 4, 12] $\rightarrow$ Median 8
  [15, 3, 5, 7, 1] $\rightarrow$ Median 5
- Partition using median and re-order groups
  [2, 1, 2, 7, 3]
  [1, 3, 5, 7, 15]
  [3, 4, 8, 9, 12]
- Median chosen is 5

Worst case linear time analysis

Worst case linear time analysis

1. $\Theta(\quad)$ time to break into groups

Worst case linear time analysis

1. $\Theta(n)$ time to break into groups

Worst case linear time analysis

1. $\Theta(n)$ time to break into groups
2. $\Theta(\quad)$ time to sort one group of 5 elements - using insertion sort

Worst case linear time analysis

1. $\Theta(n)$ time to break into groups
2. $\Theta(1)$ time to sort one group of 5 elements - using insertion sort

Worst case linear time analysis

1. $\Theta(n)$ time to break into groups
2. $\Theta(1)$ time to sort one group of 5 elements - using insertion sort
3. Thus $\Theta(\quad)$ time find median of all groups

Worst case linear time analysis

1. $\Theta(n)$ time to break into groups
2. $\Theta(1)$ time to sort one group of 5 elements - using insertion sort
3. Thus $\Theta(n)$ time find median of all groups

Worst case linear time analysis

1. $\Theta(n)$ time to break into groups
2. $\Theta(1)$ time to sort one group of 5 elements - using insertion sort
3. Thus $\Theta(n)$ time find median of all groups
4. $T(\qquad)$ time to find median of group medians

Worst case linear time analysis

1. $\Theta(n)$ time to break into groups
2. $\Theta(1)$ time to sort one group of 5 elements - using insertion sort
3. Thus $\Theta(n)$ time find median of all groups
4. $T(\lfloor n/5 \rfloor)$ time to find median of group medians

Worst case linear time analysis

1. $\Theta(n)$ time to break into groups
2. $\Theta(1)$ time to sort one group of 5 elements - using insertion sort
3. Thus $\Theta(n)$ time find median of all groups
4. $T(\lfloor n/5 \rfloor)$ time to find median of group medians
5. $\Theta(\quad)$ time to partition n elements around the pivot

Worst case linear time analysis

1. $\Theta(n)$ time to break into groups
2. $\Theta(1)$ time to sort one group of 5 elements - using insertion sort
3. Thus $\Theta(n)$ time find median of all groups
4. $T(\lfloor n/5 \rfloor)$ time to find median of group medians
5. $\Theta(n)$ time to partition n elements around the pivot

Worst case linear time analysis

1. $\Theta(n)$ time to break into groups
2. $\Theta(1)$ time to sort one group of 5 elements - using insertion sort
3. Thus $\Theta(n)$ time find median of all groups
4. $T(\lfloor n/5 \rfloor)$ time to find median of group medians
5. $\Theta(n)$ time to partition n elements around the pivot
6. Time to recursively call SELECT on one "half" of the input

- There are $\lfloor n/5 \rfloor$ group medians, at least half of which are $\leq x$, the pivot by definition. This is at least $\lfloor \lfloor n/5 \rfloor / 2 \rfloor = \lfloor n/10 \rfloor$ group medians.
- Therefore at least $3\lfloor n/10 \rfloor$ elements are $\leq x$ in total
- Similary at least $3\lfloor n/10 \rfloor$ elements are $\geq x$ in total

Summary

- RAND-SELECT is linear time on average and simple to implement but harder to analyse.
- However, RAND-SELECT runs in $\Theta(n^2)$ in the worst case.
- SELECT runs in linear time in the worst case.
- SELECT is slightly harder to implement but simpler to analyse.

Further Reading

- Introduction to Algorithms
  T.H. Cormen, C.E. Leiserson, R.L. Rivest and C. Stein.
  MIT Press/McGraw-Hill, ISBN: 0-262-03293-7.
  - Chapter 9 – Order statistics
- Algorithms
  S. Dasgupta, C.H. Papadimitriou and U.V. Vazirani
  `http://www.cse.ucsd.edu/users/dasgupta/mcgrawhill/`
  - Chapter 2, Section 2.4 – Medians