Decrease-and-Conquer

Variable-Size-Decrease Algorithms

In the variable-size-decrease variation of decrease-and-conquer, instance size reduction varies from one iteration to another

Examples:

- Euclid's algorithm for greatest common divisor
- Partition-based algorithm for selection problem
- Interpolation search
- Some algorithms on binary search trees
- Nim game

Euclid's Algorithm

- Euclid's algorithm is based on repeated application of equality

$$\gcd(m, n) = \gcd(n, m \bmod n)$$

  - Ex.: gcd(80,44) = gcd(44,36) = gcd(36, 12) = gcd(12,0) = 12

Euclid's Algorithm

- Euclid's algorithm is based on repeated application of equality

$$\gcd(m, n) = \gcd(n, m \bmod n)$$

  - Ex.: gcd(80,44) = gcd(44,36) = gcd(36, 12) = gcd(12,0) = 12

- One can prove that the size, measured by the second number, decreases at least by half after two consecutive iterations. Hence, $T(n) \in O(\log n)$

Selection Problem

- Find the $k-th$ smallest element in a list of n numbers

Selection Problem

- Find the $k-th$ smallest element in a list of n numbers

- A.k.a. find the $k-th$ order statistic
  - $k = 1 \rightarrow$ Compare all list, return min $C(n) \in \theta(n)$
  - $k = n \rightarrow$ Compare all list, return max $C(n) \in \theta(n)$

Selection Problem

Sort based algorithm:

- Sort the list
- Return $k - th$ element.
-

Selection Problem

Sort based algorithm:

- Sort the list
- Return $k - th$ element.

- Time is determined by the efficiency of the sorting algorithm used.
- Using merge sort would result in: $C(n) \in \theta(nlog\ n)$

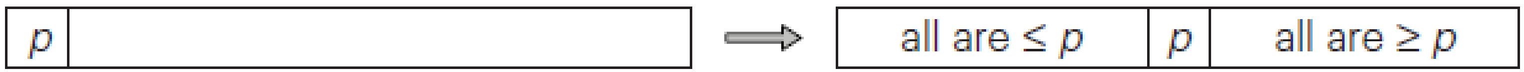
Algorithms for the Selection Problem

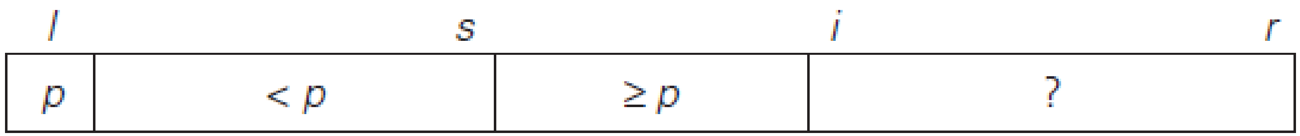
Partition based algorithm

- Use Lumoto partitioning
  - Partition list around the first element (named pivot $\boldsymbol{p}$)
  - Left side → elements smaller than $\boldsymbol{p}$
  - Right side → elements greater than $\boldsymbol{p}$
  - Place $\boldsymbol{p}$ between them in index $\boldsymbol{s}$
- Use quick select to find the $\boldsymbol{k-th}$ smallest element
  - $s == k - 1 \rightarrow \boldsymbol{p}$ is the $\boldsymbol{k-th}$ smallest
  - $s > k - 1 \rightarrow \boldsymbol{k-th}$ smallest is in the left side (repeat recursively)
  - $s < k - 1 \rightarrow \boldsymbol{k-th}$ smallest is in the right side (repeat recursively)

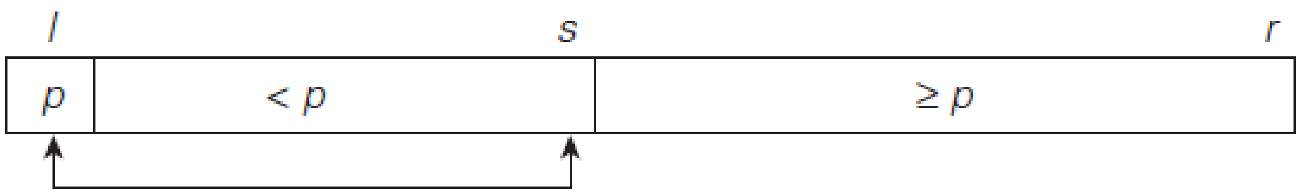
Note: The algorithm can simply continue until $\boldsymbol{s} = \boldsymbol{k}$.

Algorithms for the Selection Problem

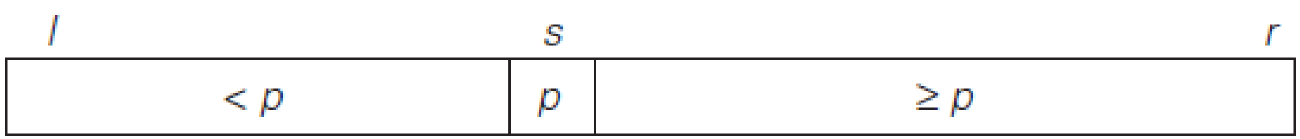




(a)



(b)



(c)

Algorithms for the Selection Problem

ALGORITHM *LomutoPartition*$(A[l..r])$

//Partitions subarray by Lomuto's algorithm using first element as pivot
//Input: A subarray $A[l..r]$ of array $A[0..n-1]$, defined by its left and right
// indices l and r $(l \le r)$
//Output: Partition of $A[l..r]$ and the new position of the pivot
$p \leftarrow A[l]$
$s \leftarrow l$
for $i \leftarrow l+1$ **to** r **do**
  if $A[i] < p$
    $s \leftarrow s+1$; swap$(A[s], A[i])$
swap$(A[l], A[s])$
return s

ALGORITHM *Quickselect*$(A[l..r], k)$

//Solves the selection problem by recursive partition-based algorithm
//Input: Subarray $A[l..r]$ of array $A[0..n-1]$ of orderable elements and
// integer k $(1 \le k \le r-l+1)$
//Output: The value of the kth smallest element in $A[l..r]$
$s \leftarrow$ *LomutoPartition*$(A[l..r])$ //or another partition algorithm
if $s = k-1$ **return** $A[s]$
else if $s > l+k-1$ *Quickselect*$(A[l..s-1], k)$
else *Quickselect*$(A[s+1..r], k-1-s)$

Algorithms for the Selection Problem

Find the median:

$$k = \left\lceil \frac{9}{2} \right\rceil = 5$$

0	1	2	3	4	5	6	7	8
s	i							
4	1	10	8	7	12	9	2	15

Algorithms for the Selection Problem

Find the median:

$$k = \left\lceil \frac{9}{2} \right\rceil = 5$$

0	1	2	3	4	5	6	7	8
s	*i*							
4	1	10	8	7	12	9	2	15
	s	*i*						
4	1	10	8	7	12	9	2	15

Algorithms for the Selection Problem

Find the median:

$$k = \left\lceil \frac{9}{2} \right\rceil = 5$$

0	1	2	3	4	5	6	7	8
s	*i*							
4	1	10	8	7	12	9	2	15
	s	*i*						
4	1	10	8	7	12	9	2	15
	s						*i*	
4	1	10	8	7	12	9	2	15

Algorithms for the Selection Problem

Find the median:

$k = \left\lceil \frac{9}{2} \right\rceil = 5$

0	1	2	3	4	5	6	7	8
s	*i*							
4	1	10	8	7	12	9	2	15
	s	*i*						
4	1	10	8	7	12	9	2	15
	s						*i*	
4	1	10	8	7	12	9	2	15
		s					*i*	
4	1	2	8	7	12	9	10	15

Algorithms for the Selection Problem

Find the median:

$$k = \left\lceil \frac{9}{2} \right\rceil = 5$$

0	1	2	3	4	5	6	7	8
s	*i*							
4	1	10	8	7	12	9	2	15
	s	*i*						
4	1	10	8	7	12	9	2	15
	s						*i*	
4	1	10	8	7	12	9	2	15
		s					*i*	
4	1	2	8	7	12	9	10	15
		s						*i*
4	1	2	8	7	12	9	10	15

Algorithms for the Selection Problem

Find the median:

$$k = \left\lceil \frac{9}{2} \right\rceil = 5$$

0	1	2	3	4	5	6	7	8
s	*i*							
4	1	10	8	7	12	9	2	15
	s	*i*						
4	1	10	8	7	12	9	2	15
	s						*i*	
4	1	10	8	7	12	9	2	15
		s					*i*	
4	1	2	8	7	12	9	10	15
		s						*i*
4	1	2	8	7	12	9	10	15
2	1	**4**	8	7	12	9	10	15

Algorithms for the Selection Problem

Find the median:

$$k = \left\lceil \frac{9}{2} \right\rceil = 5$$

0	1	2	3	4	5	6	7	8
s	*i*							
4	1	10	8	7	12	9	2	15
	s	*i*						
4	1	10	8	7	12	9	2	15
	s						*i*	
4	1	10	8	7	12	9	2	15
		s					*i*	
4	1	2	8	7	12	9	10	15
		s						*i*
4	1	2	8	7	12	9	10	15
2	1	**4**	8	7	12	9	10	15

0	1	2	3	4	5	6	7	8
			s	*i*				
			8	7	12	9	10	15

Algorithms for the Selection Problem

Find the median:

$$k = \left\lceil \frac{9}{2} \right\rceil = 5$$

0	1	2	3	4	5	6	7	8
s	*i*							
4	1	10	8	7	12	9	2	15
	s	*i*						
4	1	10	8	7	12	9	2	15
	s						*i*	
4	1	10	8	7	12	9	2	15
		s					*i*	
4	1	2	8	7	12	9	10	15
		s						*i*
4	1	2	8	7	12	9	10	15
2	1	**4**	8	7	12	9	10	15

0	1	2	3	4	5	6	7	8
			s	*i*				
			8	7	12	9	10	15
				s	*i*			
			8	7	12	9	10	15

Algorithms for the Selection Problem

Find the median:

$$k = \left\lceil \frac{9}{2} \right\rceil = 5$$

0	1	2	3	4	5	6	7	8
s	*i*							
4	1	10	8	7	12	9	2	15
	s	*i*						
4	1	10	8	7	12	9	2	15
	s						*i*	
4	1	10	8	7	12	9	2	15
		s					*i*	
4	1	2	8	7	12	9	10	15
		s						*i*
4	1	2	8	7	12	9	10	15
2	1	**4**	8	7	12	9	10	15

0	1	2	3	4	5	6	7	8
			s	*i*				
			8	7	12	9	10	15
				s	*i*			
			8	7	12	9	10	15
				s				*i*
			8	7	12	9	10	15

Algorithms for the Selection Problem

Find the median:

$$k = \left\lceil \frac{9}{2} \right\rceil = 5$$

0	1	2	3	4	5	6	7	8
s	*i*							
4	1	10	8	7	12	9	2	15
	s	*i*						
4	1	10	8	7	12	9	2	15
	s						*i*	
4	1	10	8	7	12	9	2	15
		s					*i*	
4	1	2	8	7	12	9	10	15
		s						*i*
4	1	2	8	7	12	9	10	15
2	1	**4**	8	7	12	9	10	15

0	1	2	3	4	5	6	7	8
			s	*i*				
			8	7	12	9	10	15
				s	*i*			
			8	7	12	9	10	15
				s				*i*
			8	7	12	9	10	15
			7	**8**	12	9	10	15

Algorithms for the Selection Problem

Partition based algorithm

- Use Lumoto partitioning
- Use quick select to find the k-th smallest element

$C_{best}(n) =$

$C_{worst}(n) =$

$C_{average}(n) \in$

Algorithms for the Selection Problem

Partition based algorithm

- Use Lumoto partitioning
- Use quick select to find the k-th smallest element

$$C_{best}(n) = n - 1 \ \in \theta(n)$$

$$C_{worst}(n) = (n-1) + (n-2) + \cdots + 2 + 1 = \sum_{i=1}^{n-1} i = \frac{(n-1)n}{2} \in \theta(n^2)$$

$$C_{average}(n) \in \theta(n)$$

Efficiency of the Partition-based Algorithm

Average case (average split in the middle):

A more sophisticated choice of the pivot leads to a complicated algorithm with $\Theta(n)$ worst-case efficiency.

Efficiency of the Partition-based Algorithm

Average case (average split in the middle):

$$C(n) = C\left(\frac{n}{2}\right) + (n+1)$$
$$C(n) \in \Theta(n)$$

A more sophisticated choice of the pivot leads to a complicated algorithm with $\Theta(n)$ worst-case efficiency.

Interpolation Search

- Searching for key k in a sorted array
- Assumes the array values increase linearly
- Between left most element $A[l]$ and the right most element $A[r]$

- Compare with element in index: $x = l + \left\lfloor \frac{(k-A[l])(r-l)}{A[r]-A[l]} \right\rfloor$

- $k == A[x] \rightarrow$ stop, return $\boldsymbol{x}$
- $k < A[x] \rightarrow$ search recursively in $A[l]$ to $A[x-1]$
- $k > A[x] \rightarrow$ search recursively in $A[x+1]$ to $A[r]$

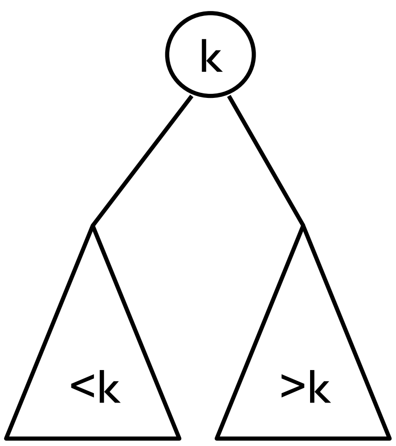
Interpolation Search

- $C_{average}(n) = (\log_2 \log_2 n) + 1 \in \theta(\log \log n)$
- $C_{worst}(n) = n \ \in \theta(n)$

- Binary search → for smaller files
- Interpolation search → for larger files

Searching in Binary Search Tree (BST)

- Nodes contain elements of orderable items k
- All elements of left subtree are smaller
- All elements of right subtree are greater
- Searching for value v



Searching in Binary Search Tree

Algorithm $BST(x, v)$

//Searches for node with key equal to v in BST rooted at node x

if $x == nil$ return -1

else if $v == K(x)$ return x

else if $v < K(x)$ return $BST(left(x), v)$

else return $BST(right(x), v)$

Searching in Binary Search Tree

Algorithm $BST(x, v)$

//Searches for node with key equal to v in BST rooted at node x

if $x == nil$ return -1

else if $v == K(x)$ return x

else if $v < K(x)$ return $BST(left(x), v)$

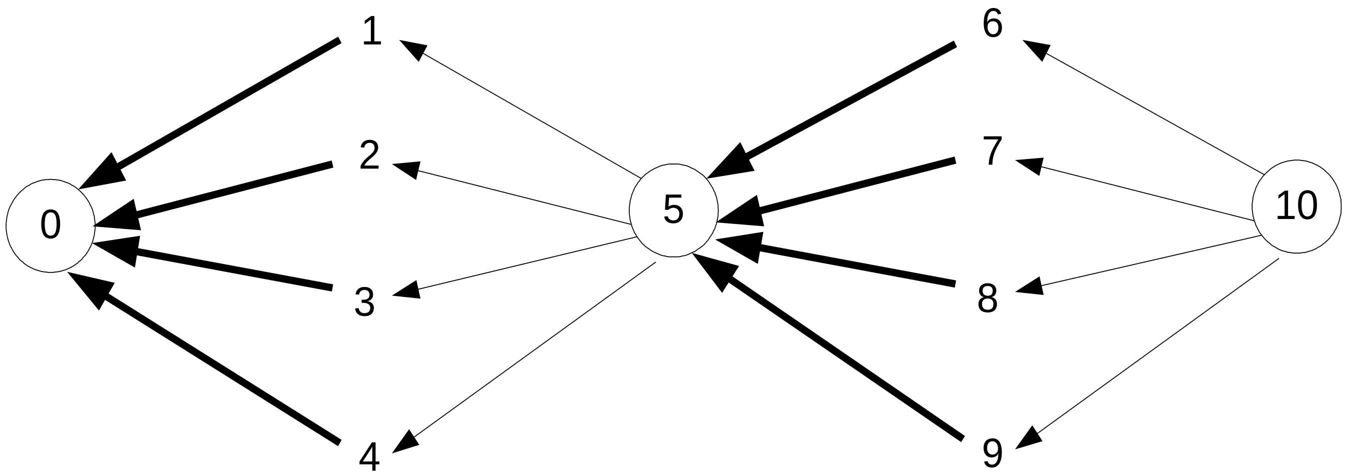
else return $BST(right(x), v)$

$C_{worst}(n) = n \in \theta(n)$
$C_{average}(n) \approx 1.39 \log_2 n \in \theta(\log n)$

One-Pile Nim

- There is a pile of n chips.
- Two players take turn.
- Removing from the pile at least 1 and at most m chips
- The winner is the player that takes the last chip.

Partial Graph of One-Pile Nim with $m = 4$



One-Pile Nim

- There is a pile of n chips.
- Two players take turn.
- Removing from the pile at least 1 and at most m chips
- The winner is the player that takes the last chip.
- Who wins the game – the player moving first or second, if both player make the best moves possible?
- Analyze this and similar games "backwards", starting with $n = 0, 1, 2, \ldots$
  - $n == 0$ → losing
  - $1 \leq n \leq m$ → winning
  - $n == m + 1$ → losing
  - $m + 1 < n \leq 2m + 1$ → winning
  - $n == 2m + 2 = 2(m + 1)$ → losing
  - .....

One-Pile Nim

- $n == 0 \rightarrow$ losing
- $1 \leq n \leq m \rightarrow$ winning
- $n == m + 1 \rightarrow$ losing
- $m + 1 < n \leq 2m + 1 \rightarrow$ winning
- $n == 2m + 2 = 2(m + 1) \rightarrow$ losing
- …..

- Instance with n chips is a winning position, iff n is not a multiple of $(m + 1)$

- Winning strategy:
- On your turn, take $n \ mod \ (m + 1)$ out