# Algorithms

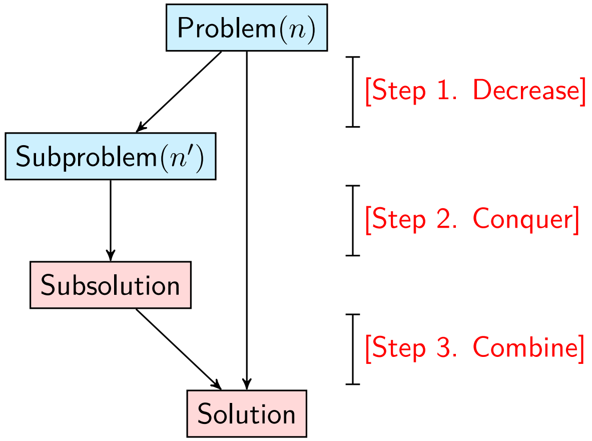
## (Decrease-and-Conquer)

**Pramod Ganapathi**
Department of Computer Science
State University of New York at Stony Brook

October 19, 2021

## Contents

## Decrease-and-conquer



## Types of decrease-and-conquer

- Decrease by constant. $n' = n - c$ for some constant $c$
- Decrease by constant factor. $n' = n/c$ for some constant $c$
- Variable size decrease. $n' = n - c$ for some variable $c$

## Decrease by constant

- Size of instance is reduced by the same constant in each iteration of the algorithm
- Decrease by 1 is common
- Examples:
  - Array sum
  - Array search
  - Find maximum/minimum element
  - Integer product
  - Exponentiation
  - Topological sorting

## Decrease by constant factor

- Size of instance is reduced by the same constant factor in each iteration of the algorithm
- Decrease by factor 2 is common
- Examples:
  - Binary search
  - Search/insert/delete in balanced search tree
  - Fake coin problem
  - Josephus problem

## Variable size decrease

- Size of instance is reduced by a variable in each iteration of the algorithm
- Examples:
  - Selection problem
  - Quicksort
  - Search/insert/delete in binary search tree
  - Interpolation search

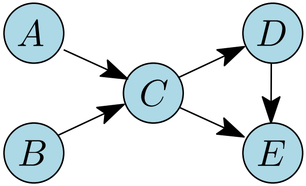
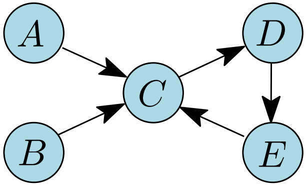
## Decrease by Constant HOME

## Topological sorting

**Problem**

- Topological sorting of vertices of a directed acyclic graph is an ordering of the vertices $v_1, v_2, \ldots, v_n$ in such a way that there is an edge directed towards vertex $v_j$ from vertex $v_i$, then $v_i$ comes before $v_j$.

## Example

| Graph | Topological sort |
|---|---|
|  | $[A, B, C, D, E]$<br>$[B, A, C, D, E]$ |
|  | Does not exist |

## DFS algorithm

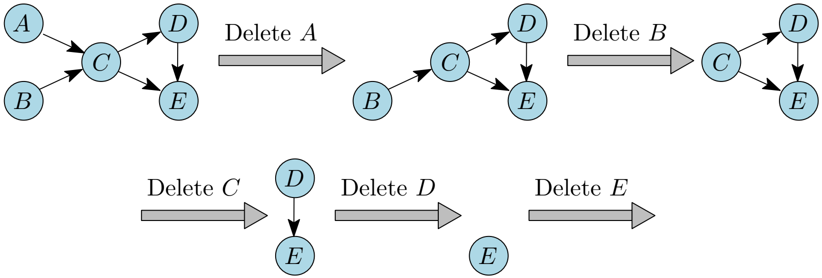
Topological sort = Reversal of the order in which the vertices become dead ends in the DFS algorithm.

```
TopologicalSort($G$)
1. Topological sort $T \leftarrow \emptyset$
2. Mark each vertex in $V$ as unvisited
3. for each vertex $v$ in $V$ do
4.   if $v$ is unvisited then
5.     DFS($v$)
6. return $T$
```

```
DFS($v$)
1. Mark $v$ as visited
2. for each vertex $w$ in $V$ adjacent to $v$ do
3.   if $w$ is unvisited then
4.     DFS($w$)
5. $T$.AddFirst($v$)
```

## Source removal algorithm

Topological sort = Order in which those vertices are removed that have 0 indegrees.



## Source-removal algorithm

```
TopologicalSort($G$)
1. Topological sort $T \leftarrow \emptyset$
2. Mark each vertex in $V$ as unvisited
3. Find $indegree[v]$ for each vertex $v$ in $V$
4. for each vertex $v$ in $V$ do
5.   if $indegree[v] = 0$ then
6.     $Q$.Enqueue($v$)
7.     Mark $v$ as visited
8. while $Q$ is not empty do
9.   $u \leftarrow Q$.Dequeue()
10.  $T$.AddLast($u$)
11.  for each vertex $w$ in $V$ adjacent to $u$ do
12.    if $w$ is unvisited then
13.      $indegree[w] \leftarrow indegree[w] - 1$
14.      if $indegree[w] = 0$ then
15.        $Q$.Enqueue($w$)
16.        Mark $w$ as visited
17. return $T$
```

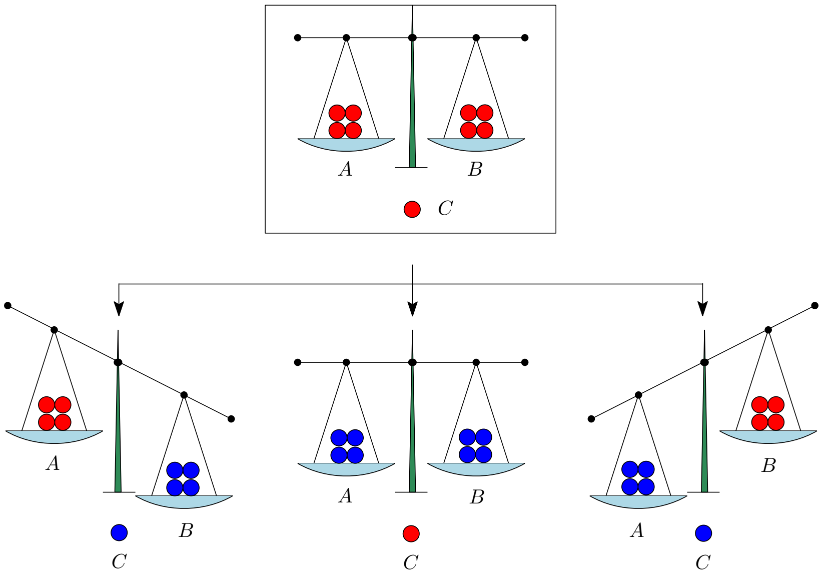
## Decrease by Constant Factor HOME

## Lighter ball

Problem

- There are $n \geq 1$ identical-looking balls, but, one of the balls is lighter than the other balls. Design an efficient algorithm to detect the lighter ball using a weighing scale/balance.

## Lighter ball among 9 balls: Divide by 2



## Decrease-by-half algorithm

```
LightérBall([$\ell, \ell+1, \ell+2, \ldots, h$])
Input: Set of $(h - \ell + 1)$ balls: $\ell, \ell+1, \ell+2, \ldots, h$
Output: Index number of the lighter ball
Require: Invocation is LighterBall([$0..n-1$]) such that $n \geq 2$
 1. if $\ell = h$ then
 2.   return $\ell$
 3. $half \leftarrow \lfloor (h - \ell + 1)/2 \rfloor$
 4. $A \leftarrow$ first $half$ number of balls i.e., $[\ell, \ell+1, \ldots, \ell + half - 1]$
 5. $B \leftarrow$ second $half$ number of balls i.e., $[\ell + half, \ldots, \ell + 2 \cdot half]$
 6. $C \leftarrow$ remaining ball $[h]$ if total balls is odd
 7. weigh sets $A$ and $B$
 8. if weight($A$) $<$ weight($B$) then
 9.   return LighterBall($A$)
10. else if weight($A$) $>$ weight($B$) then
11.   return LighterBall($B$)
12. else if weight($A$) $=$ weight($B$) then
13.   return LighterBall($C$)
```

## Complexity

- Weighings.

$$W(n) = \left\{ \begin{array}{ll} 0 & \text{if } n = 1, \\ W(\lfloor n/2 \rfloor) + 1 & \text{if } n \geq 2. \end{array} \right\}$$

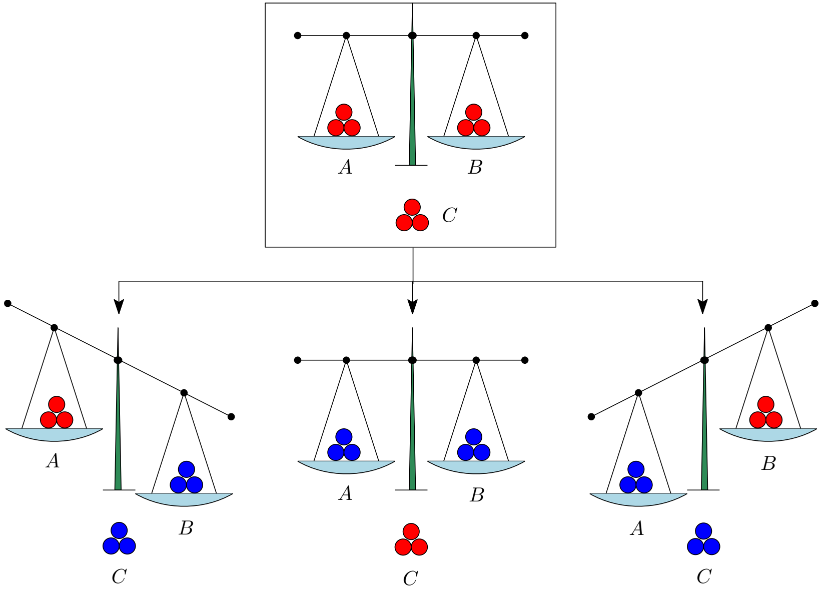
Solving, $W(n) = \lfloor \log_2 n \rfloor$

- Time complexity.

$$T(n) = \left\{ \begin{array}{ll} \Theta(1) & \text{if } n = 1, \\ T(n/2) + \Theta(1) & \text{if } n \geq 2. \end{array} \right\}$$

Solving, $T(n) \in \Theta(\log n)$

## Lighter ball among 9 balls: Divide by 3



## Decrease-by-third algorithm

```
LighterBall($[\ell, \ell+1, \ell+2, \ldots, h]$)

Input: Set of $(h-\ell+1)$ balls: $\ell, \ell+1, \ell+2, \ldots, h$
Output: Index number of the lighter ball
Require: Invocation is LighterBall($[0..n-1]$) such that $n \geq 3$
 1. if $\ell = h$ then                                                      ▷ 1 ball
 2.   return $\ell$
 3. else if $\ell = h-1$ then                                               ▷ 2 balls
 4.   return lighter ball among $\ell$ and $h$
 5. $third \leftarrow \lfloor (h-\ell+1)/3 \rfloor$
 6. $A \leftarrow$ first $third$ number of balls i.e., $[\ell, \ell+1, \ldots, \ell+third-1]$
 7. $B \leftarrow$ second $third$ number of balls i.e., $[\ell+third, \ldots, \ell+2\cdot third-1]$
 8. $C \leftarrow$ remaining balls, i.e., $[\ell+2\cdot third, \ldots, h]$
 9. weigh sets $A$ and $B$
10. if weight($A$) $<$ weight($B$) then
11.   return LighterBall($A$)
12. else if weight($A$) $>$ weight($B$) then
13.   return LighterBall($B$)
14. else if weight($A$) $=$ weight($B$) then
15.   return LighterBall($C$)
```

## Complexity

- Weighings.

$$W(n) = \left\{ \begin{array}{ll} 0 & \text{if } n = 1, \\ 1 & \text{if } n = 2, \\ W(\lceil n/3 \rceil) + 1 & \text{if } n \geq 3. \end{array} \right\}$$

Solving, $W(n) = \lceil \log_3 n \rceil$

- Time complexity.

$$T(n) = \left\{ \begin{array}{ll} \Theta(1) & \text{if } n = 1 \text{ or } 2, \\ T(n/3) + \Theta(1) & \text{if } n \geq 3. \end{array} \right\}$$

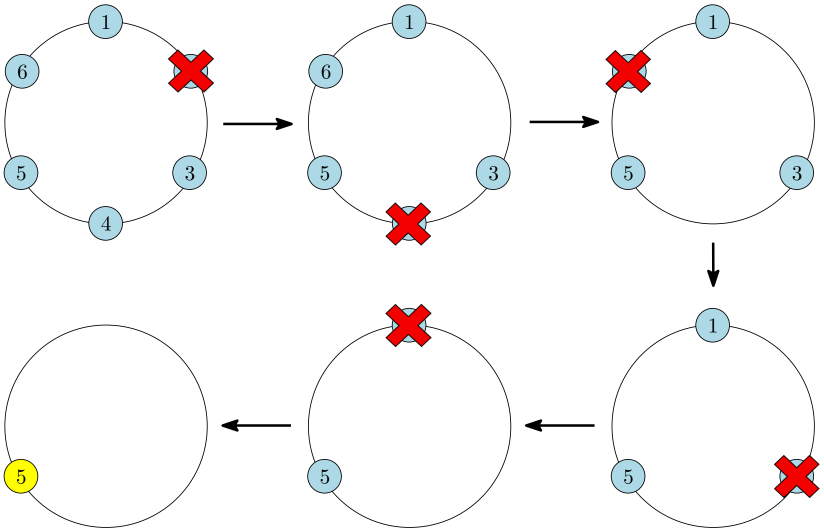
Solving, $T(n) \in \Theta(\log n)$

## Josephus problem

Problem

- There are $n$ people numbered from 1 to $n$ in a circle. Starting from person 1, we eliminate every second person until only survivor is left. Design an efficient algorithm to find the survivor's number $J(n)$.

## Example: $J(6) = 5$



## CLL algorithm

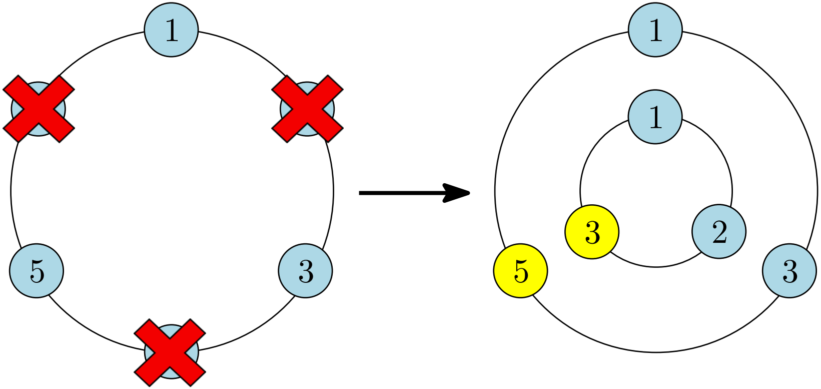
- Create a circular linked list (CLL) of size $n$.
- Node at location $i$ stores item $i$.
- Delete alternate nodes until only one node is left.
- Time is $\Theta\left(n\right)$, space is $\Theta\left(n\right)$
- Is there a more efficient algorithm?

## Decrease-by-half algorithm

$$J(n) = \left\{ \begin{array}{ll} 1 & \text{if } n = 1, \\ 2J(n/2) - 1 & \text{if } n \geq 2 \text{ and } n \text{ is even,} \\ 2J(n/2) + 1 & \text{if } n \geq 2 \text{ and } n \text{ is odd.} \end{array} \right\}$$

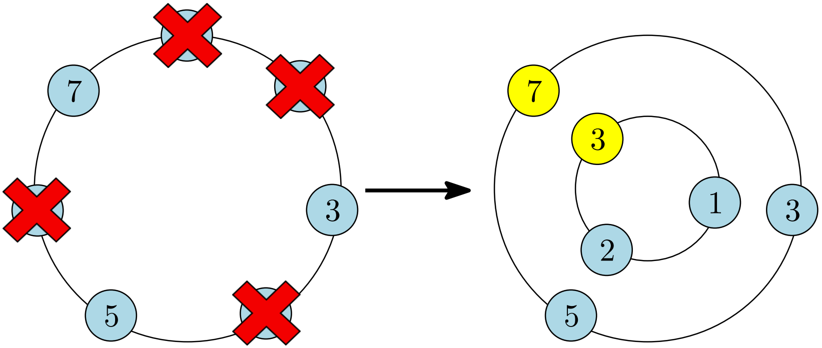
```
JOSEPHUS($n$)
Input: Whole number $n$
Output: Josephus number $J(n)$
1. if $n = 1$ then
2.   return $1$
3. else if $n$ is even then
4.   return $2 \times$ JOSEPHUS$(n/2) - 1$
5. else if $n$ is odd then
6.   return $2 \times$ JOSEPHUS$(n/2) + 1$
```

## Case: $n$ is even $(n = 2k)$



$$J(6) = 2J(3) - 1$$
$$\boxed{J(2k) = 2J(k) - 1}$$

## Case: $n$ is odd $(n = 2k + 1)$



$$J(7) = 2J(3) + 1$$

$$\boxed{J(2k+1) = 2J(k) + 1}$$

## Complexity

- Time complexity.

$$T(n) = \left\{ \begin{array}{ll} \Theta(1) & \text{if } n = 1, \\ T(n/2) + \Theta(1) & \text{if } n > 1. \end{array} \right\}$$

Solving, $T(n) \in \Theta(\log n)$

- Space complexity.

$$S(n) = \left\{ \begin{array}{ll} \Theta(1) & \text{if } n = 1, \\ S(n/2) + \Theta(1) & \text{if } n > 1. \end{array} \right\}$$

Solving, $S(n) \in \Theta(\log n)$ stack space

## Variable Size Decrease HOME

## Selection problem

**Problem**

- Find the $k$th smallest element (or $k$th order statistic) in a given array $A[0..n-1]$.

- Easiest cases. Minimum $(k = 1)$, maximum $(k = n)$
- Hardest case. Median $(k = \lfloor n/2 \rfloor)$

## Selection algorithms

| Algorithm | Time | $k$-smallest items? Sorted? |
|---|---|---|
| Sorting | $\Theta(n \log n)$ | ✓, ✓ |
| Partial selection sort | $\Theta(kn)$ | ✓, ✓ |
| Partial heapsort | $\Theta(n + k \log n)$ | ✓, ✗ |
| Online selection | $\Theta(n \log k)$ | ✓, ✗ |
| Rand. quickselect | $\Theta(n^2)$ ($\Theta(n)$ avg.) | ✓, ✗ |
| Linear-time algorithm | $\Theta(n)$ | ✗, ✗ |

## Partial selection sort

```
PartialSelectionSort($A[0..(n-1)]$)
1. Run SelectionSort on $A[0..(n-1)]$ for $k$ iterations
   to find the $k$ smallest elements in sorted order
2. return $k$th smallest element
```

Time is $\Theta(kn)$

## Partial heapsort

```
PartialHeapsort($A[0..(n-1)]$)
1. $H \leftarrow$ Construct a min-heap from $A[0..(n-1)]$ in-place    $\triangleright\ \Theta(n)$
2. $H$.DeleteMin() $k$ times                                         $\triangleright\ \Theta(k \log n)$
3. return $k$th smallest element
```

Time is $\Theta(n + k \log n)$

## Online selection

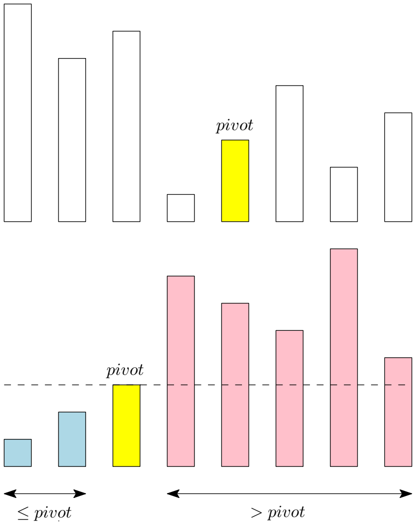
```
ONLINESELECTION($A[0..(n-1)]$)
1. $H \leftarrow$ Construct a $k$-sized max-heap from $A[0..(k-1)]$    $\triangleright\ \Theta(k)$
2. for $i \leftarrow k$ to $(n-1)$ times do    $\triangleright\ \Theta(n \log k)$
3.   if $A[i]$ is not more than the heap's maximum then
4.     $H$.INSERT($A[i]$)
5.     $H$.DELETEMAX()
6. $k$th smallest element $\leftarrow H$.DELETEMAX()    $\triangleright\ \Theta(\log k)$
7. return $k$th smallest element
```

Time is $\Theta(n \log k)$

# Randomized quickselect

```
RandomizedQuickSelect($A[\ell..h], k$)
1.  if $\ell = h$ then
2.     return $A[\ell]$
3.  $s \leftarrow$ RandomizedPartition($A[\ell..h]$)
4.  $size \leftarrow s - \ell + 1$
5.  if $k = size$ then
6.     return $A[s]$
7.  else if $k < size$ then
8.     return RandomizedQuickSelect($A[\ell..s-1], k$)
9.  else if $k > size$ then
10.    return RandomizedQuickSelect($A[s+1..h], k - size$)
```

## Randomized partition

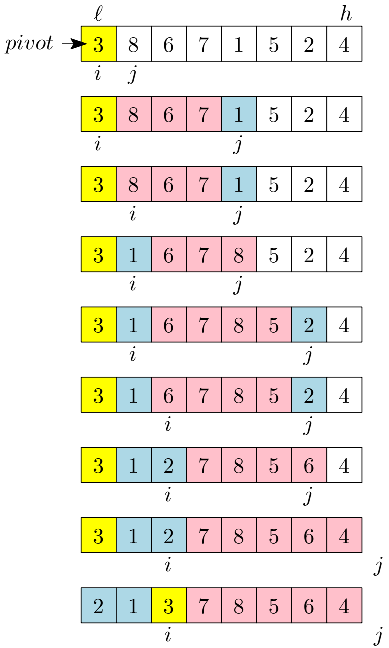


## Randomized partition (using Lomuto partition)

```
RandomizedPartition($A[\ell..h]$)
1. $i \leftarrow$ Random($\{\ell, \ell+1, \ldots, h\}$)
2. Swap($A[\ell]$, $A[i]$)
3. LomutoPartition($A[\ell..h]$)
```

```
LomutoPartition($A[\ell..h]$)
1. $pivot \leftarrow A[\ell]$                    ▷ first element is the pivot
2. $i \leftarrow \ell$
3. for $j \leftarrow \ell + 1$ to $h$ do
4.   if $A[j] \leq pivot$ then
5.     $i \leftarrow i + 1$
6.     Swap($A[i]$, $A[j]$)
7. Swap($pivot$, $A[i]$)
8. return $i$
```

# Lomuto partition



$i \leftarrow \ell$ and $j \leftarrow i + 1$. $A[j] > pivot$.

Increment $j$ until $A[j] \leq pivot$.

Increment $i$.

Swap $A[i]$ and $A[j]$.

Increment $j$ until $A[j] \leq pivot$.

Increment $i$.

Swap $A[i]$ and $A[j]$.

Increment $j$ until $j = h + 1$.

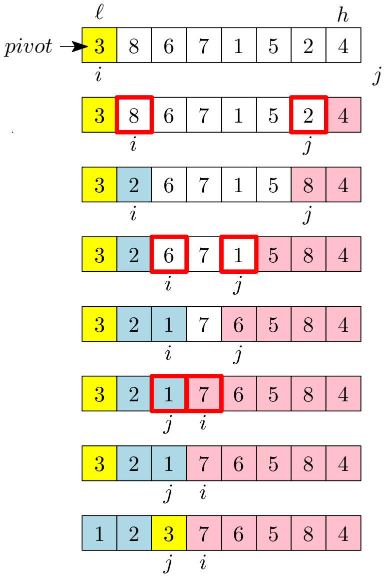
Swap *pivot* and $A[i]$. Return $i$.

## Randomized partition (using Hoare partition)

```
RandomizedPartition($A[\ell..h]$)
1.  $i \leftarrow$ Random($\{\ell, \ell+1, \ldots, h\}$)
2.  Swap($A[\ell], A[i]$)
3.  HoarePartition($A[\ell..h]$)
```

```
HoarePartition($A[\ell..h]$)
1.  $pivot \leftarrow A[\ell]$                    ▷ first element is the pivot
2.  $i \leftarrow \ell$; $j \leftarrow h+1$
3.  while $true$ do
4.  {
5.    while $A[++i] < pivot$ do
6.      if $i = h$ then break
7.    while $pivot < A[--j]$ do
8.      if $j = \ell$ then break
9.    if $i \geq j$ then break
10.   else Swap($A[i], A[j]$)
11. }
12. Swap($pivot, A[j]$)
13. return $j$
```

## Hoare partition



Initially, $i \leftarrow \ell$ and $j \leftarrow h + 1$.

Incr. $i$ and decr. $j$ until $A[i] \geq pivot \geq A[j]$.

Swap $A[i]$ and $A[j]$.

Incr. $i$ and decr. $j$ until $A[i] \geq pivot \geq A[j]$.

Swap $A[i]$ and $A[j]$.

Incr. $i$ and decr. $j$ until $A[i] \geq pivot \geq A[j]$.

Break loop because $j \leq i$.

Swap $pivot$ and $A[j]$. Return $j$.