Algorithmic Problem Solving

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

April 11, 2023

Contents

Contributors

- Usha Vudatha
- Tejas Bhatia
- Taha Kothawala
- Sai Sujith Bezawada

Algorithmic-problem-solving template

Step 1. Problem

Step 2. Solutions

Step 3. Complexity

Step 4. Performance

Step 5. Extensions

Step 6. References

Majority Element

HOME

Problem

- An election was held in a democratic nation to elect their next leader. The citizens of the nation voted for their favorite candidates. It is now time to find whether someone won the election. Winning the election means getting a majority of votes. Given a set of elements, an element is a majority in that set if that element occurs greater than 50% of the number of elements in that set. If there is no majority in an election, there will be a re-election and the process repeats until there is a majority.
  So, how do you find whether someone won an election?

Problem

- Assumption: Equality comparison $(A[i] = A[j])$ between elements are allowed. Inequality comparisons $(A[i] \leq A[j]$ or $A[i] < A[j])$ between elements are not allowed.

- Input: Array of natural numbers
  Output: Majority if it exists, -1 if there is no majority
- Input: $[3, 3, 4, 2, 4, 4, 2, 4, 4]$
  Output: 4
- Input: $[3, 3, 4, 2, 4, 4, 2, 4]$
  Output: -1

Solutions → Brute force

1. Count occurrences of each element
2. Find the majority

```
MAJORITY-BRUTEFORCE($A[1 \dots n]$)

Input: Array $A[1 \dots n]$ of natural numbers.
Output: Majority element if it exists and $-1$ otherwise.
for $i \leftarrow 1$ to $\lfloor n/2 \rfloor$ do
    $count \leftarrow$ COUNTOCCURRENCES($A[i \dots n], A[i]$)
    if $count > \lfloor n/2 \rfloor$ then
        return $A[i]$
return $-1$
```

```
COUNTOCCURRENCES($A[\ell \dots h], k$)

$count \leftarrow 0$
for $i \leftarrow \ell$ to $h$ do
    if $A[i] = k$ then
        $count \leftarrow count + 1$
return $count$
```

$$\langle \text{Time, Space} \rangle = \left\langle \Theta\left(n^2\right), \Theta\left(1\right) \right\rangle$$

Solutions → Sorting

1. Sort the array
2. Count occurrences of each element
3. Find the majority

```
MAJORITY-SORT($A[1 \ldots n]$)

$A[1 \ldots n] \leftarrow$ SORT($A[1 \ldots n]$)

$i \leftarrow 1$
for $j \leftarrow 2$ to $n$ do
    if $A[j] \neq A[i]$ then
        if $(j - i) > \lfloor n/2 \rfloor$ then
            return $A[i]$
        $i \leftarrow j$

if $(n - i + 1) > \lfloor n/2 \rfloor$ then
    return $A[i]$
return $-1$
```

$$\langle \text{Time, Space} \rangle = \langle \Theta\left(n \log n\right), \Theta\left(n\right) \rangle$$

Solutions → Divide-and-conquer

1. Split the array into two halves
2. $\ell majority \leftarrow$ majority in the left half
3. $rmajority \leftarrow$ majority in the right half
4. Check if $\ell majority$ or $rmajority$ is the array majority

```
MAJORITY-D&C($A[1 \ldots n]$)

return D&C($A[1 \ldots n]$)
```

```
D&C($A[low \ldots high]$)

if $low = high$ then return $A[low]$
$size \leftarrow (high - low + 1)$; $mid \leftarrow \lfloor (low + high)/2 \rfloor$
$\ell majority \leftarrow$ D&C($A[low \ldots mid]$)
$rmajority \leftarrow$ D&C($A[(mid + 1) \ldots high]$)
$\ell count \leftarrow$ COUNTOCCURRENCES($A[low \ldots high], \ell majority$)
$rcount \leftarrow$ COUNTOCCURRENCES($A[low \ldots high], rmajority$)
if $\ell count > \lfloor size/2 \rfloor$ then return $\ell majority$
if $rcount > \lfloor size/2 \rfloor$ then return $rmajority$
return $-1$
```

$$\langle \text{Time, Space} \rangle = \langle \Theta(n \log n), \Theta(\log n) \rangle$$

Solutions → Hashing

1. Store $\langle uniqueelement, frequency\rangle$ pairs in hash map
2. Find majority

```
MAJORITY-HASHING($A[1 \ldots n]$)

Create hash map $H$ to insert (element, frequency) pairs
for $i \leftarrow 1$ to $n$ do
    if $H$.ContainsKey($A[i]$) then
        $H$.Add($\langle A[i], H$.GetValue($A[i]) + 1\rangle$)
    else
        $H$.Add($\langle A[i], 1\rangle$)
    if $H$.GetValue($A[i]) > \lfloor n/2 \rfloor$ then
        return $A[i]$
return $-1$
```

$$\langle \text{Time, Space}\rangle = \langle \Theta\left(n\right), \Theta\left(n\right)\rangle$$

Solutions → Median

1. Find the median element
2. Check if the median is the majority

```
MAJORITY-MEDIAN($A[1 \dots n]$)
$median \leftarrow$ SELECTION($A[1 \dots n], \lfloor n/2 \rfloor$)
$count \leftarrow$ COUNTOCCURRENCES($A[1 \dots n], median$)
if $count > \lfloor n/2 \rfloor$ then
    return $median$
return $-1$
```

$$\langle \text{Time, Space} \rangle = \langle \Theta(n), \Theta(1) \rangle$$

Solutions → Probabilistic

1. Select a random element and check if it is majority
2. Repeat Step 1 for at most $\left\lfloor \log_2 \frac{1}{\epsilon} \right\rfloor$ number of times
3. Return majority

```
MAJORITY-PROBABILISTIC($A[1 \ldots n]$)

for $i \leftarrow 1$ to $\left\lfloor \log_2 \frac{1}{\epsilon} \right\rfloor$ do
    $random \leftarrow$ RANDOM($A[1 \ldots n]$)
    $count \leftarrow$ COUNTOCCURRENCES($A[1 \ldots n], random$)
    if $count > \lfloor n/2 \rfloor$ then
        return $random$
return $-1$
```

$$\langle \text{Time, Space} \rangle = \left\langle \Theta\left(n \log \frac{1}{\epsilon}\right), \Theta(1) \right\rangle$$

Solutions → BoyerMoore-Multipass

1. Consider a pair. If they are same, keep one copy, else, discard. Repeat for the entire array.
2. Repeat step 1 until there is only one element
3. Check if the element is the majority

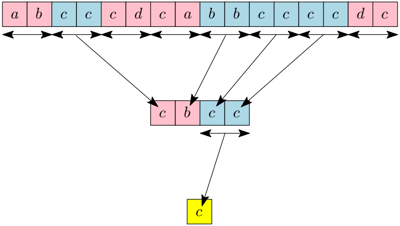
```
MAJORITY-BOYERMOORE-MULTIPASS($A[1 \ldots n]$)

Create a dynamic array $B \leftarrow [\,]$
for $i \leftarrow 1$ to $n - 1$ increment $2$ do
    if $A[i] = A[i+1]$ then
        $B$.Add($A[i]$)
if $n \bmod 2 = 1$ then  $tiebreaker \leftarrow A[n]$
if $B$ is empty then  return $tiebreaker$
$C \leftarrow$ MAJORITY-MULTIPASS($B, tiebreaker$)
if $C = -1$ then  return $-1$
$count \leftarrow$ COUNTOCCURRENCES($A[1 \ldots n], C$)
if $count > \lfloor n/2 \rfloor$ or ($count = \lfloor n/2 \rfloor$ and $C = tiebreaker$) then
    return $C$
return $-1$
```

$$\langle \text{Time, Space} \rangle = \langle \Theta(n), \Theta(n) \rangle$$

Solutions → BoyerMoore-Multipass

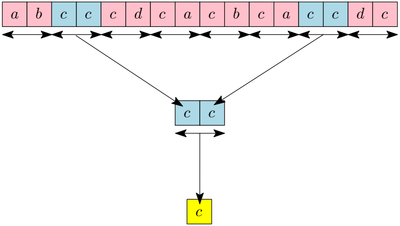
1. If a pair is different, then discard
   If a pair is same, then keep one copy
2. Repeat step 1 until only one element if left
3. If array has majority, then final element is majority
   If array has no majority, then final element has no meaning

Array has a majority



Final element is the majority

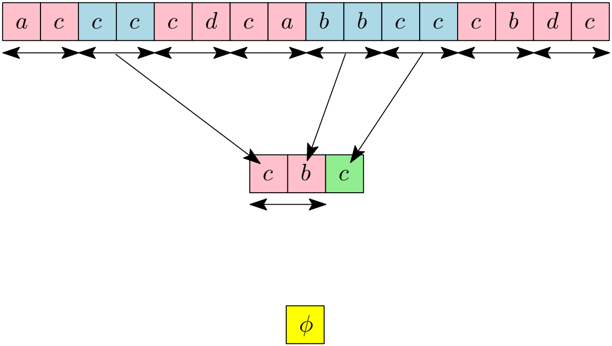
Array has no majority



Final element has no meaning

Solutions → BoyerMoore-Multipass

Array has a majority



$tiebreaker = -1$
candidate $= c$

$tiebreaker = c$
candidate $= c$

$tiebreaker = c$
candidate $= c$

Solutions → BoyerMoore-Twopass

1. Create a stack. Scan the elements one at a time.
2. If stack is empty or if element considered is the same as stack top, then push element. Else, pop an element from stack.
3. If stack is non-empty, check if stack top is the majority element. Else, return -1.

Solutions → BoyerMoore-Twopass

```
MAJORITY-BOYERMOORE-TWOPASS($A[1\dots n]$)
// Stage 1. Eliminate all except one candidate $C$
Create a stack $S$
for $i \leftarrow 1$ to $n$ do
    if $S$ is empty then  $S$.Push($A[i]$)
    else
        $top \leftarrow S$.Top()
        if $A[i] = top$ then  $S$.Push($A[i]$)
        else  $S$.Pop()
// Stage 2. Check whether $C$ is the majority
if $S$ is empty then  return $-1$
$C \leftarrow S$.Top()
$count \leftarrow$ COUNTOCCURRENCES($A[1\dots n], C$)
if $count > \lfloor n/2 \rfloor$ then  return $C$
return $-1$
```

$$\langle \text{Time, Space} \rangle = \langle \Theta\left(n\right), \mathcal{O}\left(n\right) \rangle$$

Solutions → BoyerMoore-Twopass

i	$A[i]$	S
1	a	$[a]$
2	a	$[a, a]$
3	a	$[a, a, a]$
4	b	$[a, a]$
5	b	$[a]$
6	b	ϕ
7	b	$[b]$

i	$A[i]$	S
1	a	$[a]$
2	b	ϕ
3	a	$[a]$
4	b	ϕ
5	a	$[a]$
6	b	ϕ
7	c	$[c]$

i	$A[i]$	S
1	a	$[a]$
2	b	ϕ
3	a	$[a]$
4	b	ϕ
5	a	$[a]$
6	b	ϕ

Solutions → BoyerMoore-Twopass-Inplace

1. Let C be majority candidate; m be $\#$unpaired occurrences of C
2. In iteration 1, we set $C \leftarrow$ 1st element and $m \leftarrow 1$
3. In iteration $i \in [2, n]$, if m is zero, then set $C \leftarrow i$th element and $m \leftarrow 1$. Otherwise, compare if ith element is same as C. If same, then increment m, else, decrement m.
4. If m is positive, then check if C is majority

Solutions → BoyerMoore-Twopass-Inplace

```
MAJORITY-BOYERMOORE-TWOPASS-INPLACE($A[1 \dots n]$)
$C \leftarrow A[1]$; $m \leftarrow 1$
// Stage 1. Eliminate all except one candidate $c$
for $i \leftarrow 2$ to $n$ do
    if $m = 0$ then
        { $C \leftarrow A[i]$; $m \leftarrow 1$ }
    else
        if $C = A[i]$ then $m \leftarrow m + 1$
        else $m \leftarrow m - 1$
// Stage 2. Check whether $c$ is the majority
if $m \neq 0$ then
    $count \leftarrow$ COUNTOCCURRENCES($A[1 \dots n], C$)
    if $count > \lfloor n/2 \rfloor$ then return $C$
return $-1$
```

$$\langle \text{Time, Space} \rangle = \langle \Theta(n), \Theta(1) \rangle$$

Solutions → BoyerMoore-Twopass-Inplace

1. Let C be majority candidate; m be #unpaired occurrences of C
2. In iteration 1, we set $C \leftarrow$ 1st element and $m \leftarrow 1$
3. In iteration $i \in [2, n]$, if m is zero, then set $C \leftarrow i$th element and $m \leftarrow 1$. Otherwise, compare if ith element is same as C. If same, then increment m, else, decrement m.
4. If m is positive, then check if C is majority

i	$A[i]$	C	m
1	a	a	1
2	a	a	2
3	a	a	3
4	b	b	2
5	b	b	1
6	b	b	0
7	b	b	1

i	$A[i]$	C	m
1	a	a	1
2	b	a	0
3	a	a	1
4	b	a	0
5	a	a	1
6	b	a	0
7	c	c	1

i	$A[i]$	C	m
1	a	a	1
2	b	a	0
3	a	a	1
4	b	a	0
5	a	a	1
6	b	a	0

Solutions → FischerSalzberg

```
Majority-FischerSalzberg($A[1 \ldots n]$)
// Stage 1. Find the majority candidate $C$
Create two stacks $S_1$ and $S_2$
for $i \leftarrow 1$ to $n$ do
    if $S_1$ is empty or $S_1$.Top() $\neq A[i]$ then
        $S_1$.Push($A[i]$)  if $S_2$ is not empty then  $S_1$.Push($S_2$.Pop())
    else  $S_2$.Push($A[i]$)
$C \leftarrow S_1$.Top()
// Stage 2. Confirm if the candidate is the majority
while $S_1$ is not empty do
    $item \leftarrow S_1$.Pop()
    if $item = C$ then
        if $S_1$ is empty then  $S_2$.Push($C$)
        else  $S_1$.Pop()
    else
        if $S_2$ is empty then  return $-1$
        else  $S_2$.Pop()
if $S_2$ is not empty then  return $C$
return $-1$
```

$$\langle \text{Time, Space} \rangle = \langle \Theta(n), \Theta(n) \rangle$$

Solutions → FischerSalzberg

i	$A[i]$	S_1	S_2
1	a	$[a]$	ϕ
2	a	$[a]$	$[a]$
3	a	$[a]$	$[a, a]$
4	b	$[a, b, a]$	$[a]$
5	b	$[a, b, a, b, a]$	ϕ
6	b	$[a, b, a, b, a, b]$	ϕ
7	b	$[a, b, a, b, a, b]$	$[b]$
		$[a, b, a, b]$	$[b]$
		$[a, b]$	$[b]$
		ϕ	$[b]$

i	$A[i]$	S_1	S_2
1	a	$[a]$	ϕ
2	b	$[a, b]$	ϕ
3	a	$[a, b, a]$	ϕ
4	b	$[a, b, a, b]$	ϕ
5	a	$[a, b, a, b, a]$	ϕ
6	b	$[a, b, a, b, a, b]$	ϕ
7	c	$[a, b, a, b, a, b, c]$	ϕ
		$[a, b, a, b, a]$	ϕ
		$[a, b, a, b]$	ϕ

i	$A[i]$	S_1	S_2
1	a	$[a]$	ϕ
2	b	$[a, b]$	ϕ
3	a	$[a, b, a]$	ϕ
4	b	$[a, b, a, b]$	ϕ
5	a	$[a, b, a, b, a]$	ϕ
6	b	$[a, b, a, b, a, b]$	ϕ
		$[a, b, a, b]$	ϕ
		$[a, b]$	ϕ
		ϕ	ϕ

Complexity

Algorithm	Time	Extra Space	Comments
Brute force	$\Theta\left(n^2\right)$	$\Theta(1)$	–
Sorting-based	$\Theta(n \log n)$	$\Theta(n)$	Can't solve.
Divide-and-conquer	$\Theta(n \log n)$	$\Theta(\log n)$	–
Probabilistic	$\Theta\left(n \log \frac{1}{\epsilon}\right)$	$\Theta(1)$	Can't solve. Success $> 1 - \epsilon$.
Hashing-based	$\Theta(n)$	$\Theta(n)$	Can't solve.
Median-based	$\Theta(n)$	$\Theta(n)$	Can't solve.
BoyerMoore multipass	$\Theta(n)$	$\Theta(n)$	–
BoyerMoore twopass	$\Theta(n)$	$\mathcal{O}(n)$	–
BoyerMoore twopass inplace	$\Theta(n)$	$\Theta(1)$	–
FischerSalzberg	$\Theta(n)$	$\Theta(n)$	–

References

- Puzzle book

Selection Two Sorted Arrays

HOME

Problem

- Find the kth smallest element among two sorted arrays $A[1 \ldots m]$ and $B[1 \ldots n]$, where $k \in [1, (m+n)]$.
- Input: $[10, 30, 40, 60, 70, 80, 100]$, $[20, 50, 90, 110]$, and $k = 9$
  Output: 90
- Input: $[10, 30, 40, 60, 70, 80, 100]$, $[20, 50, 90, 110]$, and $k = 4$
  Output: 40

Solutions → Concatenate&Sort

1. Concatenate the two arrays and sort it
2. Return the kth smallest element

```
SELECTION-CONCATENATE&SORT($A[1 \dots m], B[1 \dots n], k$)
    Create an array $M[1 \dots (m+n)]$
    $M[1 \dots m] \leftarrow A[1 \dots m]$                    // Copy the first array to $M$
    $M[(m+1) \dots (m+n)] \leftarrow B[1 \dots n]$            // Copy the second array to $M$
    SORT($M[1 \dots (m+n)]$)                                  // Sort $M$
    return $M[k]$                                             // Return the $k$th smallest element of $M$
```

$$\langle \text{Time, Space} \rangle = \langle \Theta\left((m+n)\log(m+n)\right), \Theta\left(m+n\right) \rangle$$

Solutions → Concatenate&Heapify

1. Concatenate the two arrays and build a heap
2. Return the kth smallest element

```
SELECTION-CONCATENATE&HEAPIFY($A[1 \ldots m], B[1 \ldots n], k$)
Create a heap $H[1 \ldots (m+n)]$
$H[1 \ldots m] \leftarrow A[1 \ldots m]$                         // Copy the first array to $H$
$H[(m+1) \ldots (m+n)] \leftarrow B[1 \ldots n]$            // Copy the second array to $H$
HEAPIFY($H[1 \ldots (m+n)]$)      // Construct heap from $H$ in linear time
// RemoveMin from $H$ for a total of $k$ times
for $i \leftarrow 1$ to $k$ do
|   $result \leftarrow H$.RemoveMin()
return $result$
```

$$\langle \text{Time, Space} \rangle = \langle \Theta\left((m+n) + k \log(m+n)\right), \Theta\left(m+n\right) \rangle$$

Solutions → Merge

1. Merge the two sorted arrays
2. Return the kth smallest element

```
SELECTION-MERGE($A[1 \ldots m], B[1 \ldots n], k$)
$i \leftarrow 1$; $j \leftarrow 1$; $\ell \leftarrow 1$
Create an array $M[1 \ldots (m+n)]$
 // Merge the two sorted arrays to $M$ until an array becomes empty
while $i \leq m$ and $j \leq n$ do
    if $A[i] \leq B[j]$ then { $M[\ell] \leftarrow A[i]$; $i \leftarrow i+1$ }
    else { $M[\ell] \leftarrow B[j]$; $j \leftarrow j+1$ }
    $\ell \leftarrow \ell + 1$
 // Copy the remaining elements to $M$
if $i > m$ then $M[\ell \ldots (m+n)] \leftarrow B[j \ldots n]$
else if $j > n$ then $M[\ell \ldots (m+n)] \leftarrow A[i \ldots m]$
 // Return the $k$th smallest element of $M$
return $M[k]$
```

$$\langle \text{Time, Space} \rangle = \langle \Theta\left(m+n\right), \Theta\left(m+n\right) \rangle$$

Solutions → MergeOptimized

1. Merge the two sorted arrays without using extra space to find the kth smallest element

```
SELECTION-MERGEOPTIMIZED($A[1 \dots m], B[1 \dots n], k$)
$i \leftarrow 1$; $j \leftarrow 1$
// Iterate for $k$ elements
while $k > 0$ do
    // If one of the arrays is reached
    if $i > m$ then return $B[j + k - 1]$
    if $j > n$ then return $A[i + k - 1]$
    // Merge-like operations
    if $A[i] < B[j]$ then $i \leftarrow i + 1$
    else if $A[i] \geq B[j]$ then $j \leftarrow j + 1$
    $k \leftarrow k - 1$
return min($A[i], B[j]$)
```

$$\langle \text{Time, Space} \rangle = \langle \Theta\left(k\right), \Theta\left(1\right) \rangle$$

Solutions → Decrease-and-Conquer (recursive)

1. Find the middle indices $mid1$ and $mid2$ of the two arrays
2. Compare $mid1 + mid2$ and k and compare $A[mid1]$ and $B[mid2]$. Recursively call the algorithm on a smaller subproblem depending on the four cases.

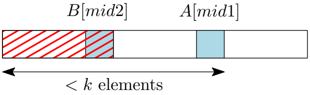
Solutions → Decrease-and-Conquer (recursive)

```
SELECTION-DE&C($A[1 \ldots m], B[1 \ldots n], k$)
    // If an array is empty, return $k$th element of other array
    if $m = 0$ then return $B[k]$
    if $n = 0$ then return $A[k]$
    // Recursive case: Find the midpoint of each array
    $mid1 \leftarrow \lfloor m/2 \rfloor$; $mid2 \leftarrow \lfloor n/2 \rfloor$
    if $mid1 + mid2 < k$ and $A[mid1] > B[mid2]$ then
        // $k$th smallest can't be in the first half of the second array $B$
        return SELECTION-DE&C($A, B[(mid2 + 1) \ldots n], k - mid2$)
    else if $mid1 + mid2 < k$ and $A[mid1] \leq B[mid2]$ then
        // $k$th smallest can't be in the first half of the first array $A$
        return SELECTION-DE&C($A[(mid1 + 1) \ldots m], B, k - mid1$)
    else if $mid1 + mid2 \geq k$ and $A[mid1] > B[mid2]$ then
        // $k$th smallest can't be in the second half of the first array $A$
        return SELECTION-DE&C($A[1 \ldots mid1], B, k$)
    else if $mid1 + mid2 \geq k$ and $A[mid1] \leq B[mid2]$ then
        // $k$th smallest can't be in the second half of the second array $B$
        return SELECTION-DE&C($A, B[1 \ldots mid2], k$)
```

$$\langle \text{Time, Space} \rangle = \langle \mathcal{O}\left(\log(m+n)\right), \mathcal{O}\left(\log(m+n)\right) \rangle$$

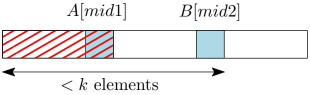
Solutions → Decrease-and-Conquer (recursive)

Case 1: $mid1 + mid2 < k$ and $A[mid1] > B[mid2]$



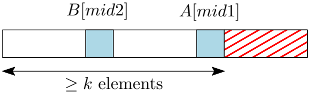
$\textsc{Selection}(A, B[(mid2+1)\ldots n], k - mid2)$
as kth smallest item isn't present in the first half of B

Case 2: $mid1 + mid2 < k$ and $A[mid1] \leq B[mid2]$



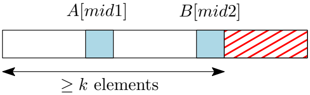
$\textsc{Selection}(A[(mid1+1)\ldots m], B, k - mid1)$
as kth smallest item isn't present in the first half of A

Case 3: $mid1 + mid2 \geq k$ and $A[mid1] > B[mid2]$



$\textsc{Selection}(A[1\ldots mid1], B, k)$
as kth smallest item isn't present in the second half of A

Case 4: $mid1 + mid2 \geq k$ and $A[mid1] \leq B[mid2]$



$\textsc{Selection}(A, B[1\ldots mid2], k)$
as kth smallest item isn't present in the second half of B

Solutions → BinarySearch (recursive)

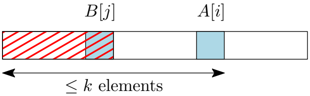
```
SELECTION-BINARYSEARCH($A[1 \dots m], B[1 \dots n], k$)
// First array should be the shorter of the two arrays
if $m > n$ then
    return SELECTION-BINARYSEARCH($B[1 \dots n], A[1 \dots m], k$)
// If first array is empty, return the $k$th element of the second array
if $m = 0$ then return $B[k]$
// If $k = 1$, return the minimum of the first elements of the two arrays
if $k = 1$ then return min($A[1], B[1]$)
// Pick the number of elements that will be discarded in the two arrays
$i \leftarrow$ min($m, \lfloor k/2 \rfloor$); $j \leftarrow k - i$
// If $A[i] > B[j]$, then discard the first $j$ elements of $B$
if $A[i] > B[j]$ then
    return SELECTION-BINARYSEARCH($A, B[j + 1 \dots n], k - j$)
// If $A[i] \leq B[j]$, then discard the first $i$ elements of $A$
else if $A[i] \leq B[j]$ then
    return SELECTION-BINARYSEARCH($A[i + 1 \dots n], B, k - i$)
```

$$\langle \text{Time, Space} \rangle = \langle \Theta\left(\log k\right), \Theta\left(\log k\right) \rangle$$

Solutions → BinarySearch (recursive)

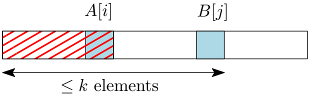
1. Select index i in the first array A, where $i = \min(m, \lfloor k/2 \rfloor)$
2. Select index j in the second array B, where $j = k - i$
3. If $A[i] > B[j]$, discard the first j elements of B and find the $(k-j)$th smallest element recursively
4. If $A[i] \leq B[j]$, discard the first i elements of A and find the $(k-i)$th smallest element recursively

Case 1: $i + j \leq k$ and $A[i] > B[j]$



$\textsc{Selection}(A, B[(j+1)\ldots n], k-j)$
as kth smallest item isn't present in the first part of B

Case 2: $i + j \leq k$ and $A[i] \leq B[j]$



$\textsc{Selection}(A[(i+1)\ldots m], B, k-i)$
as kth smallest item isn't present in the first part of A

Complexity

Algorithm	Time	Extra Space
Concatenate-sort	$\Theta\left((m+n)\log(m+n)\right)$	$\Theta\left(m+n\right)$
Concatenate-heapify	$\Theta\left((m+n)+k\log(m+n)\right)$	$\Theta\left(m+n\right)$
Merge	$\Theta\left(m+n\right)$	$\Theta\left(m+n\right)$
Merge optimized	$\Theta\left(k\right)$	$\Theta\left(1\right)$
De&C (recursive)	$\Theta\left(\log(m+n)\right)$	$\Theta\left(\log(m+n)\right)$
Binary search (recursive)	$\Theta\left(\log k\right)$	$\Theta\left(\log k\right)$

Y-shaped Linked List

HOME

Problem

- There are two singly linked lists of sizes m and n, respectively. Due to some error, the two linked lists are connected in Y-shape. Design an efficient algorithm to determine the point of intersection of the two lists given their head nodes i.e., $head1$ and $head2$.

Solutions → Brute force

1. Run two nested loops. One loop for list 1 and another for list 2.
2. If the two pointers match then that is the intersection node. Else return null.

```
YShapedLinkedList-BruteForce($head1, head2$)
pointer1 ← head1
// Outer loop for nodes in list 1
while $pointer1 \neq null$ do
    $pointer2 \leftarrow head2$
    // Inner loop for nodes in list 2
    while $pointer2 \neq null$ do
        // First time the two pointers are the same is the intersection
        if $pointer1 = pointer2$ then
            return $pointer1$
        $pointer2 \leftarrow pointer2$.Next()
    $pointer1 \leftarrow pointer1$.Next()
return $null$
```

$$\langle \text{Time, Space} \rangle = \langle \mathcal{O}\left(mn\right), \Theta\left(1\right) \rangle$$

Solutions → Hashset

1. Store every node reference of list 1 in a hashset
2. Scan each node reference of list 2 and check if it exists in the hashset

```
YShapedLinkedList-Hashset($head1, head2$)
pointer1 ← head1; pointer2 ← head2
Create a hashset $H$ to store node references
// Store every node reference of list 1 in a hashset
while $pointer1 \neq null$ do
    $H$.Add($pointer1$)
    $pointer1 \leftarrow pointer1$.Next()
// Scan each node pointer of list 2 and check if it exists in the hashset
while $pointer2 \neq null$ do
    if $H$.ContainsKey($pointer2$) then
        return $pointer2$
    $pointer2 \leftarrow pointer2$.Next()
return $null$
```

$$\langle \text{Time, Space} \rangle = \langle \mathcal{O}\left(m+n\right), \Theta\left(m\right) \rangle$$

Solutions → DifferenceCount

1. Find the difference $diff$ in the lengths of the lists. This is the length of the bottom portion of Y.
2. Advance the pointer of the longer list by $diff$
3. Now move pointers of both longer and shorter one node at a time until there the intersection node is found. Else return $null$

```
YShapedLinkedList-DifferenceCount($head1, head2$)
  // Find the difference in the lengths of the lists. Determine which list is
     longer and set the longer and shorter lists accordingly
  if $m > n$ then { $diff \leftarrow m - n$; $longer \leftarrow head1$; $shorter \leftarrow head2$ }
  else { $diff \leftarrow n - m$; $longer \leftarrow head2$; $shorter \leftarrow head1$ }

  // Advance the pointer of the longer list by the difference in lengths
  for $i \leftarrow 1$ to $diff$ do $longer \leftarrow longer$.Next()

  // Iterate through both lists until the pointers meet at the merge point
  while $longer \neq shorter$ do
      $longer \leftarrow longer$.Next()
      $shorter \leftarrow shorter$.Next()

  return $longer$
```

$$\langle \text{Time, Space} \rangle = \langle \mathcal{O}\left(m+n\right), \mathcal{O}\left(1\right) \rangle$$

Solutions → TwoStacks

1. Store all nodes of list 1 in stack 1 and list 2 in stack 2
2. Pop the same node pointers from both stacks until the pointers are different
3. The last same node reference is the intersecting node

```
YShapedLinkedList-TwoStacks($head1, head2$)
    Create two stacks $S1$ and $S2$
    $node1 \leftarrow head1$; $node2 \leftarrow head2$; $result \leftarrow null$
    // Store all nodes of list 1 in stack 1 and list 2 in stack 2
    while $node1 \neq null$ do { $S1$.Push($node1$); $node1 \leftarrow node1$.Next() }
    while $node2 \neq null$ do { $S2$.Push($node2$); $node2 \leftarrow node2$.Next() }
    // If the two stack tops are different then there is no intersection
    if $S1$.Top() $\neq$ $S2$.Top() then return $null$
    // Keep popping the same node references from the two stacks, the last
       node reference that is same is the intersection node
    while $S1$ is not empty and $S2$ is not empty and $S1$.Top() $=$ $S2$.Top() do
    |   { $result \leftarrow S1$.Pop(); $S1$.Pop() }
    // Return the intersecting node
    return $result$
```

$$\langle \text{Time, Space} \rangle = \langle \Theta\left(m+n\right), \Theta\left(m+n\right) \rangle$$

Solutions → TwoPointers

1. Scan list 1 using $pointer1$. Scan list 2 using $pointer2$.
2. If $pointer1$ reaches list 1 end, then start from list 2. If $pointer2$ reaches list 2 end, then start from list 1.
3. At any moment, when the two node references are same, it is the intersection node. Else, return $null$.

```
YShapedLinkedList-TwoPointers($head1, head2$)
    $pointer1 \leftarrow head1$; $pointer2 \leftarrow head2$
    // If one of the lists is empty, then there is no intersection node
    if $pointer1 = null$ or $pointer = null$ then return $null$
    // Traverse the lists until the intersection node is found
    while $pointer1 \neq pointer2$ do
        $pointer1 \leftarrow pointer1$.Next(); $pointer2 \leftarrow pointer2$.Next()
        if $pointer1 = pointer2$ then return $pointer1$
        // If a pointer reaches its list end, then start from other list
        if $pointer1 = null$ then $pointer1 \leftarrow head2$
        if $pointer2 = null$ then $pointer2 \leftarrow head1$
    return $pointer1$
```

$$\langle \text{Time, Space} \rangle = \langle \mathcal{O}\left(m+n\right), \Theta\left(1\right) \rangle$$

Complexity

Algorithm	Time	Extra Space
Brute force	$\mathcal{O}\left(mn\right)$	$\Theta\left(1\right)$
Hashset	$\mathcal{O}\left(m+n\right)$	$\Theta\left(m\right)$
Difference count	$\mathcal{O}\left(m+n\right)$	$\Theta\left(1\right)$
Two stacks	$\Theta\left(m+n\right)$	$\Theta\left(m+n\right)$
Two pointers	$\mathcal{O}\left(m+n\right)$	$\Theta\left(1\right)$

Random Permutation

HOME

Random Permutation

- Which of the three algorithms are correct?
- Only the third algorithm. Why?

```
RandomPermute($A[1 \ldots n]$)
for $i \leftarrow 1$ to $n - 1$ do
    $A[i] \leftarrow$ Swap($A[i], A[$Random($1 \ldots n$)$]$)
```

```
RandomPermute($A[1 \ldots n]$)
for $i \leftarrow 1$ to $n - 1$ do
    $A[i] \leftarrow$ Swap($A[i], A[$Random($i + 1 \ldots n$)$]$)
```

```
RandomPermute($A[1 \ldots n]$)
for $i \leftarrow 1$ to $n - 1$ do
    $A[i] \leftarrow$ Swap($A[i], A[$Random($i \ldots n$)$]$)
```

$$\langle \text{Time, Space} \rangle = \langle \mathcal{O}\left(\infty\right), \Theta\left(1\right) \rangle$$

Sorting Algorithms

HOME

Problem

- Design an efficient algorithm to sort a given array $A[1 \dots n]$.
- Input: $[80, 30, 90, 50, 40, 20, 100]$
  Output: $[20, 30, 40, 50, 80, 90, 100]$
- Input: $[23, 15, 40, 15, 10]$
  Output: $[10, 15, 15, 23, 40]$

Solutions → Permutation Sort

```
PermutationSort($A[1 \dots n]$)

while $true$ do
    RandomPermute($A[1 \dots n]$)
    if IsSorted($A[1 \dots n]$) then
        break
```

```
RandomPermute($A[1 \dots n]$)

for $i \leftarrow 1$ to $n - 1$ do
    $A[i] \leftarrow$ Swap($A[i], A[$Random($i \dots n$)$]$)
```

$$\langle \text{Time, Space} \rangle = \langle \mathcal{O}\left(\infty\right), \Theta\left(1\right) \rangle$$

Solutions → Slow Sort

1. Divide the subarray into two halves.
2. Sort the first half recursively.
3. Sort the second half recursively.
4. Swap the last elements of the two halves if they are out of order.
5. Sort the subarray except the last element recursively.

```
SlowSort($A[low \ldots high]$)
    if $i \geq j$ then return
    // Sort the two halves recursively
    $mid \leftarrow (low + high)/2$
    SlowSort($A[low \ldots mid]$)
    SlowSort($A[mid + 1 \ldots high]$)
    // The largest element of the subarray should go to its correct position
    if $A[high] < A[mid]$ then
        Swap($A[high], A[mid]$)
    // Sort the remaining subarray
    SlowSort($A[low \ldots high - 1]$)
```

$$\langle \text{Time, Space} \rangle = \left\langle \mathcal{O}\left(n^{\frac{\log_2 n}{2}}\right), \Theta\left(n\right) \right\rangle$$

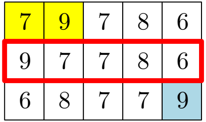
Solutions → Pancake Sort

1. Suppose the index of the maximum element in $A[1 \ldots n]$ is $maxindex$.
2. Reverse $A[1 \ldots maxindex]$ to move the largest element in the array to index 1.
3. Reverse $A[1 \ldots n]$ to move the largest element to $A[n]$.
4. Recursively sort $A[1 \ldots n-1]$.

```
PancakeSort($A[1 \ldots n]$)
    // The $i$th iteration finds the $i$th largest element
    for $i \leftarrow n$ downto $2$ do
        // Step 1. Find the index of the $\max(A[1 \ldots i])$
        $maxindex \leftarrow 1$
        for $j \leftarrow 2$ to $i$ do
            if $A[j] > A[maxindex]$ then
                $maxindex \leftarrow j$
        // Step 2. Move $\max(A[1 \ldots maxindex])$ to index 1
        Reverse($A[1 \ldots maxindex]$)
        // Step 3. Move $A[1]$ to its correct position
        Reverse($A[1 \ldots i]$)
```

$$\langle \text{Time, Space} \rangle = \left\langle \mathcal{O}\left(n^2\right), \Theta\left(1\right) \right\rangle$$

Solutions → Pancake Sort

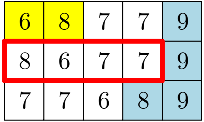


7	9	7	8	6
9	7	7	8	6
6	8	7	7	9

$i = 5$; $maxindex = 2$

Reverse $A[1 \ldots maxindex]$

Reverse $A[1 \ldots i]$

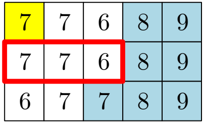


6	8	7	7	9
8	6	7	7	9
7	7	6	8	9

$i = 4$; $maxindex = 2$

Reverse $A[1 \ldots maxindex]$

Reverse $A[1 \ldots i]$

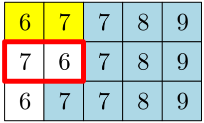


7	7	6	8	9
7	7	6	8	9
6	7	7	8	9

$i = 3$; $maxindex = 1$

Reverse $A[1 \ldots maxindex]$

Reverse $A[1 \ldots i]$



6	7	7	8	9
7	6	7	8	9
6	7	7	8	9

$i = 2$; $maxindex = 2$

Reverse $A[1 \ldots maxindex]$

Reverse $A[1 \ldots i]$

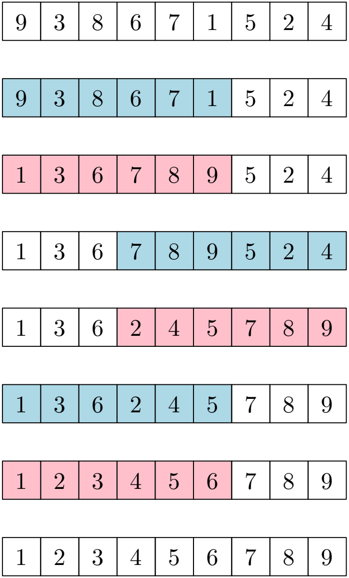
Solutions → Stooge Sort

1. If the start element is greater than the end element, swap them.
2. If there are three or more elements in the array:
    1. Recursively sort the first 2/3rd of the array
    2. Recursively sort the last 2/3rd of the array
    3. Recursively sort the first 2/3rd of the array

```
StoogeSort($A[\ell \ldots h]$)
    $size \leftarrow h - \ell + 1$
    if ($A[\ell] > A[h]$) then
        Swap($A[\ell], A[h]$)
    if ($size > 2$) then
        $third \leftarrow size/3$
        StoogeSort($A[\ell \ldots h - third]$)
        StoogeSort($A[\ell + third \ldots h]$)
        StoogeSort($A[\ell \ldots h - third]$)
```

$$\langle \text{Time, Space} \rangle = \left\langle \Theta\left(n^{\log_{1.5} 3}\right), \Theta\left(\log n\right) \right\rangle$$

Solutions → Stooge Sort



9	3	8	6	7	1	5	2	4	The original array
9	3	8	6	7	1	5	2	4	First (2/3)rd of the array
1	3	6	7	8	9	5	2	4	Sort the first (2/3)rd of the array
1	3	6	7	8	9	5	2	4	Last (2/3)rd of the array
1	3	6	2	4	5	7	8	9	Sort the last (2/3)rd of the array
1	3	6	2	4	5	7	8	9	First (2/3)rd of the array
1	2	3	4	5	6	7	8	9	Sort the first (2/3)rd of the array
1	2	3	4	5	6	7	8	9	The original array is sorted

Solutions → Counting Sort

Assumption

- Items are natural numbers with maximum value k.

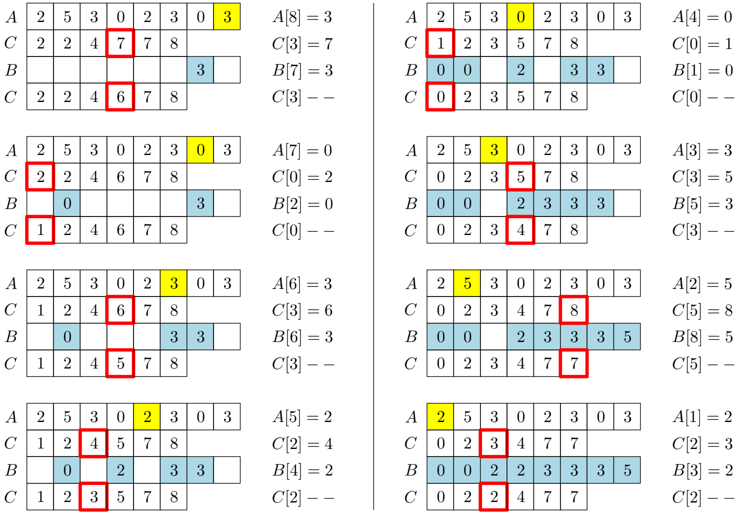
1. Create an array for indices in the range $[0, k]$
2. Distribute items to these indices to compute item frequences
3. Compute the cumulative frequencies of items for indices in the range $[0, k]$
4. Find the sorted array

A	2	5	3	0	2	3	0	3	Unsorted array $A[1..n]$
C	2	0	2	3	0	1			Frequencies array $C[0..k]$
C	2	2	4	7	7	8			Cumulative frequencies array $C[0..k]$
B	0	0	2	2	3	3	3	5	Sorted array $B[1..n]$

Solutions → Counting Sort

```
CountingSort($A[1 \ldots n]$)
$k \leftarrow \max(A[1 \ldots n])$
Create new array $B[1 \ldots n]$
Create new array $C[0 \ldots k]$ and initialize it to $0$

// Find the frequencies of items
// After this step, $C[i]$ will contain #elements equal to $i$
for $j \leftarrow 1$ to $n$ do
    $C[A[j]] \leftarrow C[A[j]] + 1$

// Find the cumulative frequencies of items
// After this step, $C[i]$ will contain #elements less than or equal to $i$
for $i \leftarrow 1$ to $k$ do
    $C[i] \leftarrow C[i] + C[i - 1]$

// Get the sorted array in $B$
for $j \leftarrow n$ downto $1$ do
    $B[C[A[j]] \leftarrow A[j]$
    $C[A[j]] \leftarrow C[A[j]] - 1$

// Copy the sorted array to $A$
for $j \leftarrow 1$ to $n$ do
    $A[j] \leftarrow B[j]$
```

Solutions → Counting Sort



A	2	5	3	0	2	3	0	3
C	2	2	4	7	7	8		
B							3	
C	2	2	4	6	7	8		

$A[8] = 3$
$C[3] = 7$
$B[7] = 3$
$C[3] - -$

A	2	5	3	0	2	3	0	3
C	2	2	4	6	7	8		
B		0					3	
C	1	2	4	6	7	8		

$A[7] = 0$
$C[0] = 2$
$B[2] = 0$
$C[0] - -$

A	2	5	3	0	2	3	0	3
C	1	2	4	6	7	8		
B		0				3	3	
C	1	2	4	5	7	8		

$A[6] = 3$
$C[3] = 6$
$B[6] = 3$
$C[3] - -$

A	2	5	3	0	2	3	0	3
C	1	2	4	5	7	8		
B		0		2		3	3	
C	1	2	3	5	7	8		

$A[5] = 2$
$C[2] = 4$
$B[4] = 2$
$C[2] - -$

A	2	5	3	0	2	3	0	3
C	1	2	3	5	7	8		
B	0	0		2		3	3	
C	0	2	3	5	7	8		

$A[4] = 0$
$C[0] = 1$
$B[1] = 0$
$C[0] - -$

A	2	5	3	0	2	3	0	3
C	0	2	3	5	7	8		
B	0	0		2	3	3	3	
C	0	2	3	4	7	8		

$A[3] = 3$
$C[3] = 5$
$B[5] = 3$
$C[3] - -$

A	2	5	3	0	2	3	0	3
C	0	2	3	4	7	8		
B	0	0		2	3	3	3	5
C	0	2	3	4	7	7		

$A[2] = 5$
$C[5] = 8$
$B[8] = 5$
$C[5] - -$

A	2	5	3	0	2	3	0	3
C	0	2	3	4	7	7		
B	0	0	2	2	3	3	3	5
C	0	2	2	4	7	7		

$A[1] = 2$
$C[2] = 3$
$B[3] = 2$
$C[2] - -$

Solutions → Counting Sort Variant

- This algorithm counts the number of occurrences of each element in the input sequence and then uses that information to construct the sorted output. It is often used when the range of input elements is known in advance.
- The algorithm works by distributing the input elements into a number of bins/buckets based on their values and then collecting from the bins in order, resulting in a sorted output
- This algorithm is typically used for sorting a large number of elements with a small range of possible values

Solutions → Counting Sort Variant

```
CountingSortVariant($A[1 \dots n]$)
$(max, min) \leftarrow$ MaxMin($A[1 \dots n]$)
$size \leftarrow max - min + 1$                          // size of range $[min, max]$
Create an array $B[1 \dots size] \leftarrow [0 \dots 0]$

 // Distribute array $A$ elements to buckets in $B$
for $j \leftarrow 1$ to $n$ do
    $i \leftarrow A[j] - min + 1$; $B[i] \leftarrow B[i] + 1$

 // Construct the sorted array $A$ based on the bucket array
$index \leftarrow 1$
for $i \leftarrow 1$ to $size$ do
    while $B[i] > 0$ do
        $A[index] \leftarrow i + min - 1$
        $index \leftarrow index + 1$
        $B[i] \leftarrow B[i] - 1$
```

Let $\#buckets = \max(A[1 \dots n]) - \min(A[1 \dots n])$

$$\langle \text{Time, Space} \rangle = \langle \Theta\left(n + \#buckets\right), \Theta\left(\#buckets\right) \rangle$$

Solutions → Counting Sort Variant

A	5	7	9	3	5	3	4	5
B	2	1	3	0	1	0	1	

B	2	1	3	0	1	0	1	
A	3							
B	1	1	3	0	1	0	1	

$index = 1$, $i = 1$
$A[index] = i + min - 1$
$B[i] - -$

B	1	1	3	0	1	0	1	
A	3	3						
B	0	1	3	0	1	0	1	

$index = 2$, $i = 1$
$A[index] = i + min - 1$
$B[i] - -$

B	0	1	3	0	1	0	1	
A	3	3	4					
B	0	0	3	0	1	0	1	

$index = 3$, $i = 2$
$A[index] = i + min - 1$
$B[i] - -$

B	0	0	3	0	1	0	1	
A	3	3	4	5				
B	0	0	2	0	1	0	1	

$index = 4$, $i = 3$
$A[index] = i + min - 1$
$B[i] - -$

B	0	0	2	0	1	0	1	
A	3	3	4	5	5			
B	0	0	1	0	1	0	1	

$index = 5$, $i = 3$
$A[index] = i + min - 1$
$B[i] - -$

B	0	0	1	0	1	0	1	
A	3	3	4	5	5	5		
B	0	0	0	0	1	0	1	

$index = 6$, $i = 3$
$A[index] = i + min - 1$
$B[i] - -$

B	0	0	0	0	1	0	1	
A	3	3	4	5	5	5		
B	0	0	0	0	1	0	1	

$index = 4$, $i = 4$

B	0	0	0	0	1	0	1	
A	3	3	4	5	5	5	7	
B	0	0	0	0	0	0	1	

$index = 7$, $i = 5$
$A[index] = i + min - 1$
$B[i] - -$

B	0	0	0	0	0	0	1	
A	3	3	4	5	5	5	7	
B	0	0	0	0	0	0	1	

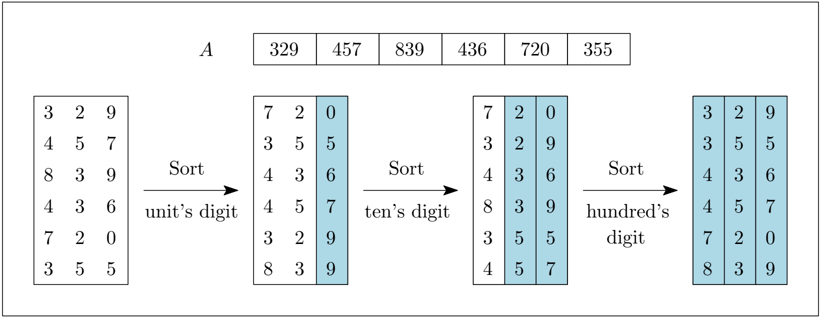
$index = 8$, $i = 6$

B	0	0	0	0	0	0	1	
A	3	3	4	5	5	5	7	9
B	0	0	0	0	0	0	0	

$index = 8$, $i = 7$
$A[index] = i + min - 1$
$B[i] - -$

Solutions → Radix Sort

1. Sort the numbers based on digits at unit's place
2. Sort the numbers based on digits at ten's place
3. Sort the numbers based on digits at hundred's place
4. Continue the process until you cover all decimal digits
5. By the end, the entire array will be sorted



A					
329	457	839	436	720	355

3	2	9
4	5	7
8	3	9
4	3	6
7	2	0
3	5	5

Sort unit's digit

7	2	0
3	5	5
4	3	6
4	5	7
3	2	9
8	3	9

Sort ten's digit

7	2	0
3	2	9
4	3	6
8	3	9
3	5	5
4	5	7

Sort hundred's digit

3	2	9
3	5	5
4	3	6
4	5	7
7	2	0
8	3	9

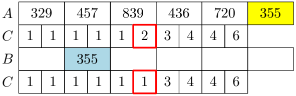
Solutions → Radix Sort

```
$\textsc{RadixSort}(A[1 \ldots n])$
$max \leftarrow \mathsf{Max}(A[1 \ldots n])$; $exp \leftarrow 1$
// Sort the array for each digit
while $exp \leq max$ do
    $C[0 \ldots 9] \leftarrow [0 \ldots 0]$; $B[1 \ldots n] \leftarrow [0 \ldots 0]$
    // Find the cumulative frequencies of items
    for $i \leftarrow 1$ to $n$ do
        { $index \leftarrow \left\lfloor \frac{A[i]}{exp} \right\rfloor \bmod 10$; $C[index] \leftarrow C[index] + 1$ }
    for $i \leftarrow 1$ to $9$ do  $C[i] \leftarrow C[i] + C[i-1]$
    // Populate output using count array
    for $i \leftarrow n$ downto $1$ do
        $index \leftarrow \left\lfloor \frac{A[i]}{exp} \right\rfloor \bmod 10$
        $B[C[index]] \leftarrow A[i]$
        $C[index] \leftarrow C[index] - 1$
    $A[1 \ldots n] \leftarrow B[1 \ldots n]$; $exp \leftarrow exp \times 10$
```

$$\langle \text{Time, Space} \rangle = \langle \Theta\left(n \log n\right), \Theta\left(n\right) \rangle$$

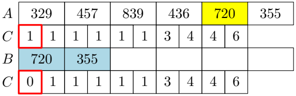
Solutions → Radix Sort

- Sort numbers based on the digits at unit's place



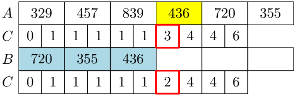
A	329	457	839	436	720	355				
C	1	1	1	1	1	2	3	4	4	6
B		355								
C	1	1	1	1	1	1	3	4	4	6

$A[6] = 355$
$C[5] = 2$
$B[2] = 355$
$C[5] - -$



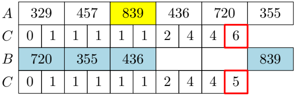
A	329	457	839	436	720	355				
C	1	1	1	1	1	1	3	4	4	6
B	720	355								
C	0	1	1	1	1	1	3	4	4	6

$A[5] = 720$
$C[0] = 1$
$B[1] = 720$
$C[0] - -$



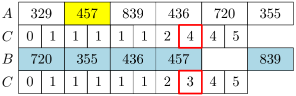
A	329	457	839	436	720	355				
C	0	1	1	1	1	1	3	4	4	6
B	720	355	436							
C	0	1	1	1	1	1	2	4	4	6

$A[4] = 436$
$C[6] = 3$
$B[3] = 436$
$C[6] - -$



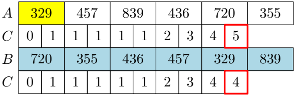
A	329	457	839	436	720	355				
C	0	1	1	1	1	1	2	4	4	6
B	720	355	436			839				
C	0	1	1	1	1	1	2	4	4	5

$A[3] = 839$
$C[9] = 6$
$B[6] = 839$
$C[9] - -$



A	329	457	839	436	720	355				
C	0	1	1	1	1	1	2	4	4	5
B	720	355	436	457		839				
C	0	1	1	1	1	1	2	3	4	5

$A[2] = 457$
$C[7] = 4$
$B[4] = 457$
$C[7] - -$

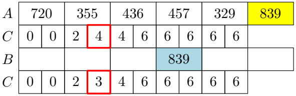


A	329	457	839	436	720	355				
C	0	1	1	1	1	1	2	3	4	5
B	720	355	436	457	329	839				
C	0	1	1	1	1	1	2	3	4	4

$A[1] = 329$
$C[9] = 5$
$B[5] = 329$
$C[9] - -$

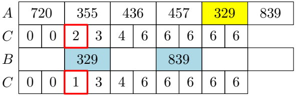
Solutions → Radix Sort

- Sort numbers based on the digits at ten's place
- B from the previous iteration will be the A for this iteration.



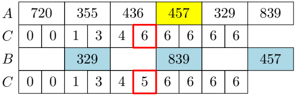
A	720		355		436		457		329	839
C	0	0	2	4	4	6	6	6	6	6
B							839			
C	0	0	2	3	4	6	6	6	6	6

$A[6] = 839$
$C[3] = 4$
$B[4] = 839$
$C[3] - -$



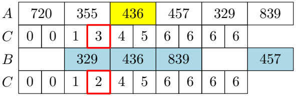
A	720		355		436		457		329	839
C	0	0	2	3	4	6	6	6	6	6
B			329				839			
C	0	0	1	3	4	6	6	6	6	6

$A[5] = 329$
$C[2] = 1$
$B[1] = 329$
$C[2] - -$



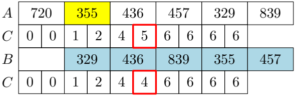
A	720		355		436		457		329	839
C	0	0	1	3	4	6	6	6	6	6
B			329				839			457
C	0	0	1	3	4	5	6	6	6	6

$A[4] = 457$
$C[5] = 6$
$B[6] = 457$
$C[5] - -$



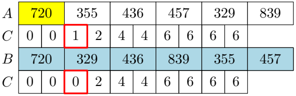
A	720		355		436		457		329	839
C	0	0	1	3	4	5	6	6	6	6
B			329		436		839			457
C	0	0	1	2	4	5	6	6	6	6

$A[3] = 436$
$C[3] = 3$
$B[3] = 436$
$C[3] - -$



A	720		355		436		457		329	839
C	0	0	1	2	4	5	6	6	6	6
B			329		436		839		355	457
C	0	0	1	2	4	4	6	6	6	6

$A[2] = 355$
$C[5] = 5$
$B[5] = 355$
$C[5] - -$

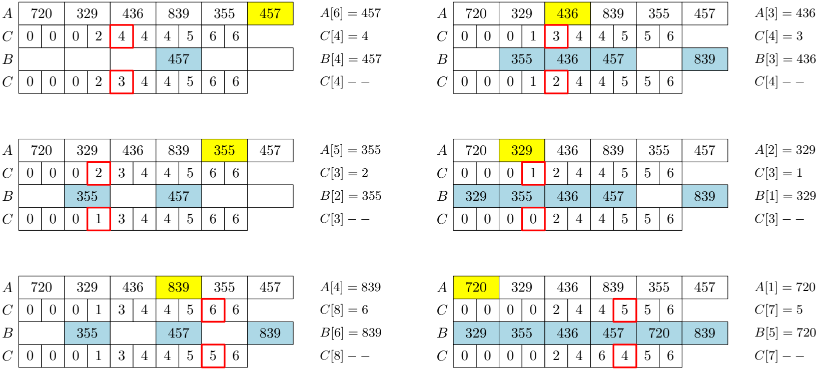


A	720		355		436		457		329	839
C	0	0	1	2	4	4	6	6	6	6
B	720		329		436		839		355	457
C	0	0	0	2	4	4	6	6	6	6

$A[1] = 720$
$C[2] = 1$
$B[1] = 720$
$C[2] - -$

Solutions → Radix Sort

- Sort numbers based on the digits at hundred's place
- B from the previous iteration will be the A for this iteration.



<table>
<tr><td>A</td><td colspan="2">720</td><td colspan="2">329</td><td colspan="2">436</td><td colspan="2">839</td><td colspan="2">355</td><td colspan="2">457</td></tr>
<tr><td>C</td><td>0</td><td>0</td><td>0</td><td>2</td><td>4</td><td>4</td><td>4</td><td>5</td><td>6</td><td>6</td><td colspan="2"></td></tr>
<tr><td>B</td><td colspan="2"></td><td colspan="2"></td><td colspan="2"></td><td colspan="2">457</td><td colspan="2"></td><td colspan="2"></td></tr>
<tr><td>C</td><td>0</td><td>0</td><td>0</td><td>2</td><td>3</td><td>4</td><td>4</td><td>5</td><td>6</td><td>6</td><td colspan="2"></td></tr>
</table>

$A[6] = 457$
$C[4] = 4$
$B[4] = 457$
$C[4] --$

<table>
<tr><td>A</td><td colspan="2">720</td><td colspan="2">329</td><td colspan="2">436</td><td colspan="2">839</td><td colspan="2">355</td><td colspan="2">457</td></tr>
<tr><td>C</td><td>0</td><td>0</td><td>0</td><td>2</td><td>3</td><td>4</td><td>4</td><td>5</td><td>6</td><td>6</td><td colspan="2"></td></tr>
<tr><td>B</td><td colspan="2"></td><td colspan="2">355</td><td colspan="2"></td><td colspan="2">457</td><td colspan="2"></td><td colspan="2"></td></tr>
<tr><td>C</td><td>0</td><td>0</td><td>0</td><td>1</td><td>3</td><td>4</td><td>4</td><td>5</td><td>6</td><td>6</td><td colspan="2"></td></tr>
</table>

$A[5] = 355$
$C[3] = 2$
$B[2] = 355$
$C[3] --$

<table>
<tr><td>A</td><td colspan="2">720</td><td colspan="2">329</td><td colspan="2">436</td><td colspan="2">839</td><td colspan="2">355</td><td colspan="2">457</td></tr>
<tr><td>C</td><td>0</td><td>0</td><td>0</td><td>1</td><td>3</td><td>4</td><td>4</td><td>5</td><td>6</td><td>6</td><td colspan="2"></td></tr>
<tr><td>B</td><td colspan="2"></td><td colspan="2">355</td><td colspan="2"></td><td colspan="2">457</td><td colspan="2"></td><td colspan="2">839</td></tr>
<tr><td>C</td><td>0</td><td>0</td><td>0</td><td>1</td><td>3</td><td>4</td><td>4</td><td>5</td><td>5</td><td>6</td><td colspan="2"></td></tr>
</table>

$A[4] = 839$
$C[8] = 6$
$B[6] = 839$
$C[8] --$

<table>
<tr><td>A</td><td colspan="2">720</td><td colspan="2">329</td><td colspan="2">436</td><td colspan="2">839</td><td colspan="2">355</td><td colspan="2">457</td></tr>
<tr><td>C</td><td>0</td><td>0</td><td>0</td><td>1</td><td>3</td><td>4</td><td>4</td><td>5</td><td>5</td><td>6</td><td colspan="2"></td></tr>
<tr><td>B</td><td colspan="2"></td><td colspan="2">355</td><td colspan="2">436</td><td colspan="2">457</td><td colspan="2"></td><td colspan="2">839</td></tr>
<tr><td>C</td><td>0</td><td>0</td><td>0</td><td>1</td><td>2</td><td>4</td><td>4</td><td>5</td><td>5</td><td>6</td><td colspan="2"></td></tr>
</table>

$A[3] = 436$
$C[4] = 3$
$B[3] = 436$
$C[4] --$

<table>
<tr><td>A</td><td colspan="2">720</td><td colspan="2">329</td><td colspan="2">436</td><td colspan="2">839</td><td colspan="2">355</td><td colspan="2">457</td></tr>
<tr><td>C</td><td>0</td><td>0</td><td>0</td><td>1</td><td>2</td><td>4</td><td>4</td><td>5</td><td>5</td><td>6</td><td colspan="2"></td></tr>
<tr><td>B</td><td colspan="2">329</td><td colspan="2">355</td><td colspan="2">436</td><td colspan="2">457</td><td colspan="2"></td><td colspan="2">839</td></tr>
<tr><td>C</td><td>0</td><td>0</td><td>0</td><td>0</td><td>2</td><td>4</td><td>4</td><td>5</td><td>5</td><td>6</td><td colspan="2"></td></tr>
</table>

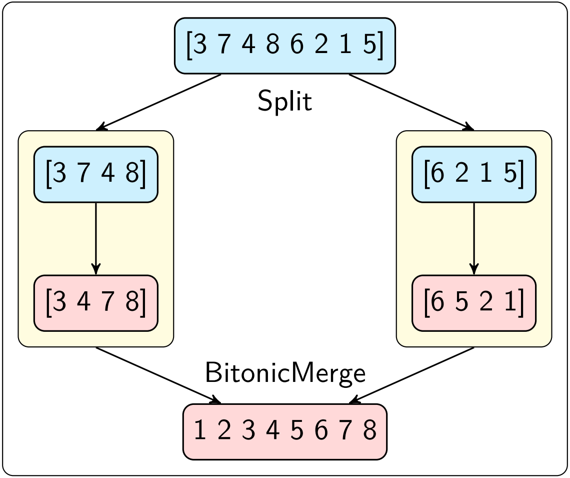
$A[2] = 329$
$C[3] = 1$
$B[1] = 329$
$C[3] --$

<table>
<tr><td>A</td><td colspan="2">720</td><td colspan="2">329</td><td colspan="2">436</td><td colspan="2">839</td><td colspan="2">355</td><td colspan="2">457</td></tr>
<tr><td>C</td><td>0</td><td>0</td><td>0</td><td>0</td><td>2</td><td>4</td><td>4</td><td>5</td><td>5</td><td>6</td><td colspan="2"></td></tr>
<tr><td>B</td><td colspan="2">329</td><td colspan="2">355</td><td colspan="2">436</td><td colspan="2">457</td><td colspan="2">720</td><td colspan="2">839</td></tr>
<tr><td>C</td><td>0</td><td>0</td><td>0</td><td>0</td><td>2</td><td>4</td><td>6</td><td>4</td><td>5</td><td>6</td><td colspan="2"></td></tr>
</table>

$A[1] = 720$
$C[7] = 5$
$B[5] = 720$
$C[7] --$

Solutions → Bitonic Sort

BitonicSort(n)



[3 7 4 8 6 2 1 5]

Split

[3 7 4 8]

[6 2 1 5]

[3 4 7 8]

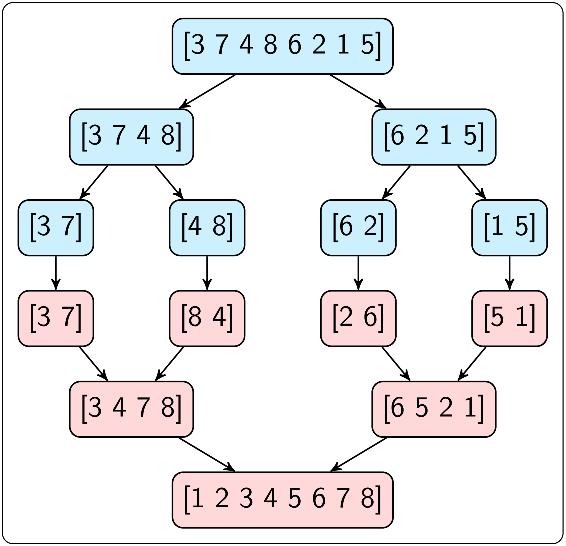
[6 5 2 1]

BitonicMerge

1 2 3 4 5 6 7 8

Solutions → Bitonic Sort

BitonicSort(n)



[3 7 4 8 6 2 1 5]

[3 7 4 8]

[6 2 1 5]

[3 7]

[4 8]

[6 2]

[1 5]

[3 7]

[8 4]

[2 6]

[5 1]

[3 4 7 8]

[6 5 2 1]

[1 2 3 4 5 6 7 8]

Solutions → Bitonic Sort

- Invoke $\textsc{BitonicSort}(A[1 \ldots n], ascending)$

```
$\textsc{BitonicSort}(A[\ell \ldots h], order)$
$size \leftarrow h - \ell + 1$
if $size > 1$ then
    $m \leftarrow (\ell + h)/2$
    $\textsc{BitonicSort}(A[\ell \ldots m], ascending)$
    $\textsc{BitonicSort}(A[m + 1 \ldots h], descending)$
    $\textsc{BitonicMerge}(A[\ell \ldots h], order)$
```

Solutions → Bitonic Sort

```
BitonicMerge($A[\ell \ldots h], order$)
Input: Array $A[\ell \ldots h]$, ascending/descending order
Output: Bitonic merge the array
$size \leftarrow h - \ell + 1$
if $size > 1$ then
    $m \leftarrow (\ell + h)/2$
    Compare&Swap($A[\ell \ldots h], order$)
    BitonicMerge($A[\ell \ldots m], order$)
    BitonicMerge($A[m + 1 \ldots h], order$)
```

```
Compare&Swap($A[\ell \ldots h], order$)
Input: Array $A[\ell \ldots h]$, ascending/descending order
Output: Compare items in left & right halves of $A[\ell \ldots h]$ and order them
$size \leftarrow h - \ell + 1$
for $i \leftarrow \ell$ to $\ell + size/2 - 1$ do
    $j \leftarrow i + size/2$
    if ($order$ is $ascending$ and $A[i] > A[j]$) or ($order$ is $descending$
     and $A[i] < A[j]$) then
        Swap($A[i], A[j]$)
```

Solutions → Bitonic Sort

$$\langle \text{Time, Space} \rangle = \left\langle \Theta\left(n \log^2 n\right), \Theta\left(n\right) \right\rangle$$

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

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

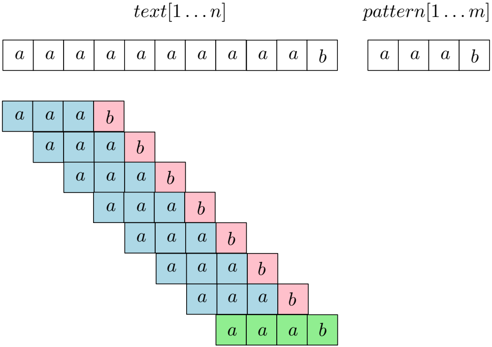
String Matching

HOME

Problem

- Given a text $text[1 \ldots n]$ and a pattern $pattern[1 \ldots m]$, design an algorithm to find the location of the first occurrence of the pattern in the text.

Solutions → Brute Force



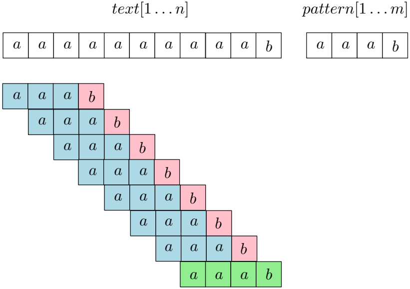
Solutions → Brute Force

1. Check if the pattern matches the text starting from the 1st index of text.
2. If not, check if the pattern matches with the text starting from the 2nd index of the text.
3. Repeat this process until either the pattern is found or the end of the text is reached (without finding any pattern).

```
StringMatching-BruteForce($text[1 \ldots n], pattern[1 \ldots m]$)
for $i \leftarrow 1$ to $n - m + 1$ do
    // If text window at position $i$ matches with pattern, return position
    if $text[i \ldots (i + m - 1)] = pattern$ then
        return $i$
return $-1$
```

$$\langle \text{PreprocessTime, MatchTime, Space} \rangle = \langle 0, \mathcal{O}(mn), \Theta(1) \rangle$$

Solutions → Hashing



```
HASH($string[1 \ldots m], b, p$)
    // Polynomial hash: $s_1 b^{m-1} + s_2 b^{m-2} + \cdots + s_{m-1} b^1 + s_m b^0$
    // Use Horner's rule to compute polynomial hash
    $hash \leftarrow 0$
    for $i \leftarrow 1$ to $m$ do
        $hash \leftarrow (hash \times b + string[i]) \bmod p$
    return $hash$
```

Time $= \Theta(m)$

Solutions → Hashing

1. Check if patternhash matches the texthash at index 1.
2. If not, check if patternhash matches the texthash at index 2.
3. Repeat this process until either the pattern is found or the end of the text is reached (without finding any pattern).

```
StringMatching-Hashing($text[1 \dots n], pattern[1 \dots m]$)
$p \leftarrow$ a good prime                                   // e.g.: $101$
$b \leftarrow$ size of ASCII set                              // i.e., $256$
$patternhash \leftarrow$ Hash($pattern, b, p$)
$texthash \leftarrow$ Hash($text[1 \dots m], b, p$)

for $i \leftarrow 1$ to $n - m + 1$ do
    // If hash value of text window matches the hash value of pattern and
       if the text window matches the pattern then there is a match
    if $texthash = patternhash$ and $text[i \dots (i + m - 1)] = pattern$ then
        return $i$

    // Compute hash value of the next text window in $\Theta(m)$ time
    if $i \neq n - m + 1$ then
        $texthash \leftarrow$ Hash($text[i + 1 \dots i + m]$)
return $-1$
```

$$\langle \text{PreprocessTime, MatchTime, Space} \rangle = \langle \Theta(m), \mathcal{O}(mn), \Theta(1) \rangle$$

Solutions → RabinKarp (rolling hash)

```
StringMatching-RabinKarp($text[1 \ldots n], pattern[1 \ldots m]$)
$p \leftarrow$ a good prime                                   // e.g.: $101$
$b \leftarrow$ size of ASCII set                              // i.e., $256$
$h \leftarrow b^{m-1} \bmod p$          // highest term in the polynomial hash
$patternhash \leftarrow$ Hash($pattern, b, p$)
$texthash \leftarrow$ Hash($text[1 \ldots m], b, p$)

for $i \leftarrow 1$ to $n - m + 1$ do
    if $texthash = patternhash$ and $text[i \ldots (i + m - 1)] = pattern$ then
        return $i$

     // Rolling hash: Compute hash value of the next text window using
        the current text window in $\Theta\left(1\right)$ time
    if $i \neq n - m + 1$ then
        $texthash \leftarrow$ RollingHash($texthash, text[i \ldots i + m]$)
return $-1$
```

Solutions → RabinKarp (rolling hash)

```
RollingHash($texthash, string[1 \ldots m']$)                    $(m' = m + 1)$
$texthash \leftarrow ((texthash - string[1] \times h) \times b + string[m']) \bmod p$
return $texthash$
```

Time $= \Theta(1)$

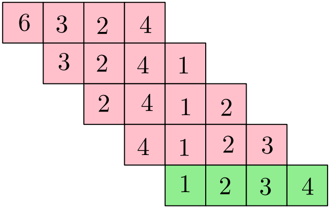
Solutions → RabinKarp (rolling hash)

$text[1 \ldots n]$

6	3	2	4	1	2	3	4	2	6	8	9	7

$pattern[1 \ldots m]$

1	2	3	4



$$1 \cdot 10^3 + 2 \cdot 10^2 + 3 \cdot 10^1 + 4 \cdot 10^0 = 1234$$

$$6 \cdot 10^3 + 3 \cdot 10^2 + 2 \cdot 10^1 + 4 \cdot 10^0 = 6324$$

$$(6324 - 6 \cdot 10^3) \cdot 10 + 1 = 3241$$

$$(3241 - 3 \cdot 10^3) \cdot 10 + 2 = 2412$$

$$(2412 - 2 \cdot 10^3) \cdot 10 + 3 = 4123$$

$$(4123 - 4 \cdot 10^3) \cdot 10 + 4 = 1234$$

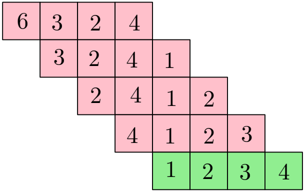
Solutions → RabinKarp (rolling hash)

$text[1 \ldots n]$

6	3	2	4	1	2	3	4	2	6	8	9	7

$pattern[1 \ldots m]$

1	2	3	4



$$(1 \cdot 10^3 + 2 \cdot 10^2 + 3 \cdot 10^1 + 4 \cdot 10^0) \bmod 31 = 2$$

$$(6 \cdot 10^3 + 3 \cdot 10^2 + 2 \cdot 10^1 + 4 \cdot 10^0) \bmod 31 = 8$$

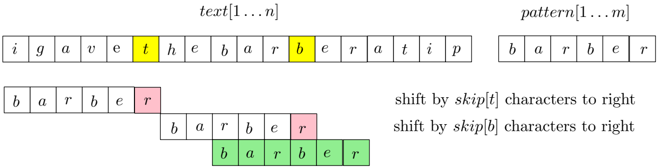
$$((8 - 6 \cdot 10^3 \bmod 31) \cdot 10 + 1) \bmod 31 = 25$$

$$((25 - 3 \cdot 10^3 \bmod 31) \cdot 10 + 2) \bmod 31 = 2$$

$$((2 - 2 \cdot 10^3 \bmod 31) \cdot 10 + 3) \bmod 31 = 8$$

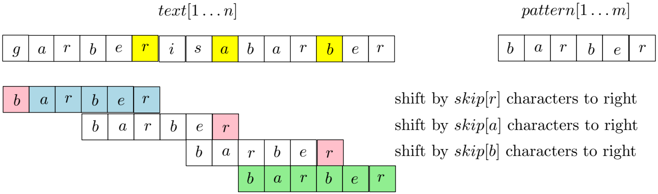
$$((8 - 4 \cdot 10^3 \bmod 31) \cdot 10 + 4) \bmod 31 = 2$$

Solutions → Boyer-Moore-Horspool



shift by $skip[t]$ characters to right

shift by $skip[b]$ characters to right



shift by $skip[r]$ characters to right

shift by $skip[a]$ characters to right

shift by $skip[b]$ characters to right

character	b	a	r	e	other
$skip$[character]	2	4	3	1	6

$$skip[\alpha] = \left\{ \begin{array}{ll} \text{distance from the end of the pattern of } \alpha\text{'s last occurrence} & \text{if } \alpha \neq pattern[m] \\ \text{distance from the end of the pattern of } \alpha\text{'s last but one occurrence} & \text{if } \alpha = pattern[m] \end{array} \right\}$$

Solutions → Boyer-Moore-Horspool

```
STRINGMATCHING-BMH($text[1 \dots n], pattern[1 \dots m]$)
skip[0...255] ← CONSTRUCTSKIPTABLE($pattern$)
$i \leftarrow m$
while $i \leq n$ do
    if $text[(i - m + 1) \dots i] = pattern$ comparing from right to left then
        return $i - m + 1$
    else
        $i \leftarrow i + skip[text[i]]$
return $-1$
```

```
CONSTRUCTSKIPTABLE($pattern[1 \dots m]$)
// Initialize the skip table of ASCII characters to $m$
$skip[0 \dots 255] \leftarrow [m \dots m]$
for $i \leftarrow 1$ to $m - 1$ do
    $skip[pattern[i]] \leftarrow m - i$
return $skip[0 \dots 255]$
```

$\langle$PreprocessTime, MatchTime, Space$\rangle = \langle \Theta(m + |\Sigma|), \mathcal{O}(mn), \Theta(|\Sigma|) \rangle$

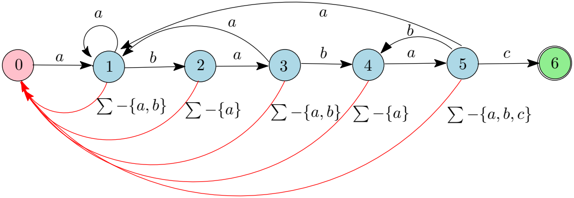
Solutions → Aho-Corasick

$text[1 \dots n]$

a	a	b	a	c	a	a	b	a	b	a	c	a	a

$pattern[1 \dots m]$

a	b	a	b	a	c



State	a	b	c	$\sum - \{a, b, c\}$
0	1	0	0	0
1	1	2	0	0
2	3	0	0	0
3	1	4	0	0
4	5	0	0	0
5	1	4	6	0
6	–	–	–	–

Solutions → Aho-Corasick

```
StringMatching-AhoCorasick($text[1 \dots n], pattern[1 \dots m]$)
$transitiontable[0 \dots m, 0 \dots 255] \leftarrow$ BuildTransitionTable($pattern$)
$state \leftarrow 0$
for $i \leftarrow 1$ to $n$ do
    $state \leftarrow transitiontable[state, text[i]]$
    if $state = m$ then
        return $i - m + 1$
return $-1$
```

```
BuildTransitionTable($pattern[1 \dots m]$)
Create matrix $transitiontable[0 \dots m, 0 \dots 255]$; initialize it to $-1$
$state \leftarrow 0$
for $x \leftarrow 0$ to $m$ do
    for $c \leftarrow 0$ to $255$ do
        if $state < m$ and $pattern[state + 1] = c$ then
            $x \leftarrow state + 1$
        $transitiontable[state, c] = x$
        $x \leftarrow transitiontable[x, c]$
return $transitiontable$
```

$$\langle \text{PreprocessTime, MatchTime, Space} \rangle = \langle \Theta(m), \mathcal{O}(n), \Theta(m) \rangle$$

Complexity

Algorithm	Preprocess time	Matching time	Space
Brute force	–	$\mathcal{O}(mn)$	$\Theta(1)$
Rabin Karp	$\Theta(m)$	$\mathcal{O}(mn)$	$\Theta(1)$
Horspool	$\Theta(\lvert\Sigma\rvert)$	$\mathcal{O}(mn)$	$\Theta(\lvert\Sigma\rvert)$
Aho-Corasick	$\Theta(m)$	$\mathcal{O}(n)$	$\Theta(m\lvert\Sigma\rvert)$

First Missing Positive

HOME

Problem

- Given an array $A[1 \dots n]$ of unique integers, design an efficient approach to find the smallest missing natural number.
- Input: $[2, -9, 5, 11, 1, -10, 7]$
  Output: 3
- Extension: What if we allow duplicates or repetitions?

Solutions → Brute Force 1

1. Check if i is missing in the array $A[1 \ldots n]$ for $i \in [1 \ldots n]$
2. Stop and return the smallest such i, otherwise return $n + 1$

```
FirstMissingPositive-BruteForce1($A[1 \ldots n]$)

// Check if the any natural number from 1 to $n$ is missing
for $i \leftarrow 1$ to $n$ do
    $imissing \leftarrow true$
    // Iterate over the array to check if the natural number exists
    for $j \leftarrow 1$ to $n$ do
        // If $i$ is found then break
        if $i = A[j]$ then
            $imissing \leftarrow false$
            break
    // Missing value found
    if $imissing = true$ then
        return $i$
return $n + 1$
```

$$\langle \text{Time, Space} \rangle = \left\langle \mathcal{O}\left(n^2\right), \Theta\left(1\right) \right\rangle$$

Solutions → Brute Force 2

1. Create an empty sorted set S to add all natural numbers from array
2. Check if i is missing in the array $A[1 \ldots n]$ for $i \in [1 \ldots n]$
3. Stop and return the smallest such i, otherwise return $n + 1$

```
FirstMissingPositive-BruteForce2($A[1 \ldots n]$)
// Create a sorted set to store the natural numbers
Create an empty sorted set $S$ using a balanced search tree
for $i \leftarrow 1$ to $n$ do
    if $A[i] > 0$ then
        $S$.Add($A[i]$)
// Find the first missing natural number from $A[1 \ldots n]$ using $S$
for $i \leftarrow 1$ to $n$ do
    if $S$ does not contain $i$ then
        return $i$
return $n + 1$
```

$$\langle \text{Time, Space} \rangle = \langle \mathcal{O}\left(n \log n\right), \mathcal{O}\left(n\right) \rangle$$

Solutions → Scan

1. Sort the input array in-place to skip non-natural numbers
2. Check if i is missing in the array $A[1 \ldots n]$ for $i \in [1 \ldots n]$
3. Stop and return the smallest such i, otherwise return $n + 1$

```
FirstMissingPositive-Scan($A[1 \ldots n]$)
// Sort the array in-place
Sort($A[1 \ldots n]$)
// Skip negative numbers and zero from the array
$index \leftarrow 1$
while $A[index] < 1$ do  $index \leftarrow index + 1$
$i \leftarrow 1$
// Find the missing natural number from the sorted input array
for $j \leftarrow index$ to $n$ do
    if $A[j] = i$ then  $i \leftarrow i + 1$
    else if $A[j] > i$ then
    return $i$
return $i$
```

$$\langle \text{Time, Space} \rangle = \langle \mathcal{O}(n \log n), \Theta(1) \rangle$$

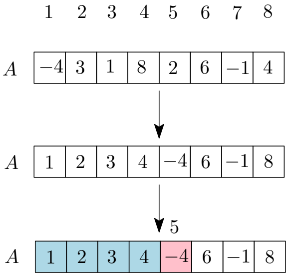
Solutions → In-Place Hashing

1. Use $i \in [1 \dots n]$ of the same array to mark the presence of the numbers
2. If $A[i]$ is a natural number and $i \leq n$, swap & place it in $A[A[i]]$
3. Stop and return smallest i where $A[i] \neq i$, otherwise return $n+1$

```
FirstMissingPositive-InPlaceHashing($A[1 \dots n]$)
// Swap natural number $A[i]$ to $A[i]$th index if $A[i] \in [1 \dots n]$
for $i \leftarrow 1$ to $n$ do
    while $A[i] \geq 1$ and $A[i] \leq n$ and $A[i] \neq A[A[i]]$ do
        Swap($A[i]$,$A[A[i]]$)
// Find the first natural number that is not $A[i] \neq i$ in $A[1 \dots n]$
for $i \leftarrow 1$ to $n$ do
    if $A[i] \neq i$ then
        return $i$
return $n+1$
```

$$\langle \text{Time, Space} \rangle = \langle \Theta(n), \Theta(1) \rangle$$

Solutions → In-Place Hashing



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

$A[1 \ldots n]$

A	1	2	3	4	-4	6	-1	8

For each $i \in [1, n]$, do this in a loop.

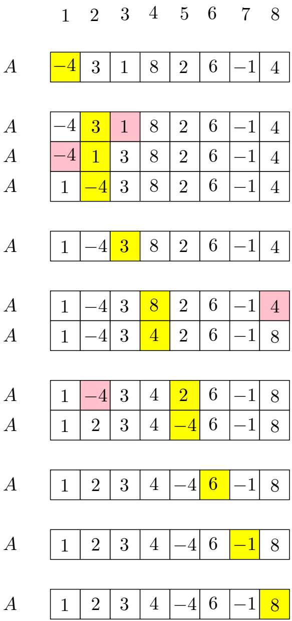
If $A[i] \in [1, n]$ and $A[i] \neq A[A[i]]$, then Swap($A[i]$,$A[A[i]]$)

A	1	2	3	4	-4	6	-1	8

Return the first i which is not equal to $A[i]$

That is, return 5

Solutions → In-Place Hashing



	1	2	3	4	5	6	7	8	
A	-4	3	1	8	2	6	-1	4	$i = 1$; $A[i] \notin [1, n]$, so do nothing
A	-4	3	1	8	2	6	-1	4	$i = 2$; $A[i] \in [1, n]$ and $A[i] \neq A[A[i]]$, so $\textsc{Swap}(A[i], A[A[i]])$
A	-4	1	3	8	2	6	-1	4	$i = 2$; $A[i] \in [1, n]$ and $A[i] \neq A[A[i]]$, so $\textsc{Swap}(A[i], A[A[i]])$
A	1	-4	3	8	2	6	-1	4	$i = 2$; $A[i] \notin [1, n]$, so do nothing
A	1	-4	3	8	2	6	-1	4	$i = 3$; $A[i] \in [1, n]$ but $A[i] = A[A[i]]$, so do nothing
A	1	-4	3	8	2	6	-1	4	$i = 4$; $A[i] \in [1, n]$ and $A[i] \neq A[A[i]]$, so $\textsc{Swap}(A[i], A[A[i]])$
A	1	-4	3	4	2	6	-1	8	$i = 4$; $A[i] \in [1, n]$ but $A[i] = A[A[i]]$, so do nothing
A	1	-4	3	4	2	6	-1	8	$i = 5$; $A[i] \in [1, n]$ and $A[i] \neq A[A[i]]$, so $\textsc{Swap}(A[i], A[A[i]])$
A	1	2	3	4	-4	6	-1	8	$i = 5$; $A[i] \notin [1, n]$, so do nothing
A	1	2	3	4	-4	6	-1	8	$i = 6$; $A[i] \in [1, n]$ but $A[i] = A[A[i]]$, so do nothing
A	1	2	3	4	-4	6	-1	8	$i = 7$; $A[i] \notin [1, n]$, so do nothing
A	1	2	3	4	-4	6	-1	8	$i = 8$; $A[i] \in [1, n]$ but $A[i] = A[A[i]]$, so do nothing

Solutions → HashTable

1. Insert all the array numbers in a hashtable H
2. Find the first natural number i that is not present in H

```
FirstMissingPositive-HashTable($A[1 \ldots n]$)
// Create a HashTable to store the natural numbers
Create a HashTable $H$
for $i \leftarrow 1$ to $n$ do  $H[A[i]] \leftarrow true$
// Find the first missing natural number from $A[1 \ldots n]$ using $H$
for $i \leftarrow 1$ to $n + 1$ do
    if $H$ does not contain $i$ then
        return $i$
```

$$\langle \text{Time, Space} \rangle = \langle \mathcal{O}(n), \mathcal{O}(n) \rangle$$

THIS SOLUTION MIGHT NOT ALWAYS WORK. WHY?

Complexity

Algorithm	Time	Space
Brute Force 1	$\mathcal{O}\left(n^2\right)$	$\Theta\left(1\right)$
Brute Force 2	$\mathcal{O}\left(n \log n\right)$	$\mathcal{O}\left(n\right)$
Scan	$\mathcal{O}\left(n \log n\right)$	$\Theta\left(1\right)$
In-Place Hashing	$\Theta\left(n\right)$	$\Theta\left(1\right)$
In-Place Hashing & Partition	$\Theta\left(n\right)$	$\Theta\left(1\right)$

Primality Test

HOME

Problem

- Given a positive integer greater than 1, check if the number is prime or not.
- A prime is a natural number greater than 1 that has no positive divisors other than 1 and itself.
- Input: $n = 11$
  Output: prime
- Input: $n = 15$
  Output: composite

Solutions → Naive Algorithm

- If n is divisible by any number in the range $[2, n-1]$, then n is composite, else, n is prime

```
PRIMALITY-NAIVEALGORITHM($n$)
for $i \leftarrow 2$ to $n - 1$ do
    if $n \bmod i = 0$ then
        return composite
return prime
```

$$\langle \text{Time, Space} \rangle = \langle \mathcal{O}(n), \Theta(1) \rangle$$

Solutions → School Algorithm

- If n is divisible by any number in the range $[2, n-1]$, then n is composite, else, n is prime
- This is because a larger factor of n must be a multiple of a smaller factor that has been already checked

```
PRIMALITY-SCHOOLALGORITHM($n$)
for $i \leftarrow 2$ to $\lfloor \sqrt{n} \rfloor$ do
    if $n \bmod i = 0$ then
        return composite
return prime
```

$$\langle \text{Time, Space} \rangle = \langle \mathcal{O}\left(\sqrt{n}\right), \Theta\left(1\right) \rangle$$

Solutions → Optimized School Algorithm

- All integers can be expressed as $(6k + i)$, where $i \in \{-1, 0, 1, 2, 3, 4\}$.
- Test whether n is divisible by 2 or 3. But 2 divides $(6k + 0), (6k + 2)$, $(6k + 4)$ and 3 divides $(6k + 3)$. So, simply check if n is divisible by any number in the form $(6k \pm 1)$ not greater than $\sqrt{n}$.

```
PRIMALITY-OPTIMIZEDSCHOOLALGORITHM($n$)

if $n = 2$ or $n = 3$ then  return prime
if $n \bmod 2 = 0$ or $n \bmod 3 = 0$ then  return composite

 // Check if $n$ is divisible by a number of the form $6k \pm 1$
for $i \leftarrow 5$ to $\left(\lfloor\sqrt{n}\rfloor - 2\right)$ increment $6$ do
    if $n \bmod i = 0$ then
        return composite                                // $i = 6k - 1$
    if $n \bmod (i + 2) = 0$ then
        return composite                                // $i = 6k + 1$
return prime
```

$$\langle \text{Time, Space} \rangle = \left\langle \mathcal{O}\left(\sqrt{n}\right), \Theta\left(1\right) \right\rangle$$

Solutions → Sieve of Eratosthenes

```
PRIMALITY-SIEVEOFERATOSTHENES($n$)

$last \leftarrow \lfloor \sqrt{n} \rfloor$
Create a Boolean array $P[2 \ldots last]$ to indicate prime numbers
for $i \leftarrow 2$ to $last$ do
    $P[i] \leftarrow$ true
for $j \leftarrow 2$ to $last$ do
    if $P[j] =$ true then
        for $k \leftarrow 2$ to $\lfloor last/j \rfloor$ do
            $i \leftarrow j \times k$
            $P[i] \leftarrow$ false
        if $n \bmod j = 0$ then
            return composite
return prime
```

$$\langle \text{Time, Space} \rangle = \langle \mathcal{O}\left(\sqrt{n}\log\log n\right), \Theta\left(\sqrt{n}\right) \rangle$$

Solutions → Wilson's Theorem

- Wilson's theorem: A positive integer $n > 1$ is prime iff $((n-1)! + 1) \bmod n = 0$

n	$(n-1)!$	$((n-1)!+1) \bmod n$	Is Prime?
2	1	0	✓
3	2	0	✓
4	6	2	✗
5	24	0	✓
6	120	1	✗
7	720	0	✓
8	5040	1	✗
9	40320	1	✗
10	362880	1	✗
11	3628800	0	✓
12	39916800	1	✗
13	479001600	0	✓

Solutions → Wilson's Theorem

- Wilson's theorem: A positive integer $n > 1$ is prime iff $((n-1)! + 1) \bmod n = 0$

```
PRIMALITY-WILSONTHEOREM($n$)
$factorial \leftarrow 1$
for $i \leftarrow 2$ to $n - 1$ do
    $factorial \leftarrow (factorial \times i) \bmod n$
if $(factorial + 1) = n$ then return prime
return composite
```

$$\langle \text{Time, Space} \rangle = \langle \Theta(n), \Theta(1) \rangle$$

Solutions → Fermat's Theorem

- $n \geq 4$ is prime iff for all $a \in [2, n-2]$, we have $(a^{n-1} - 1) \bmod n = 0$.

$n : a$	2	3	4	5	6	7	8	9	10	11	12	13
4	3											
5	0	0										
6	1	2	3									
7	0	0	0	0								
8	7	2	7	4	7							
9	3	8	6	6	8	3						
10	1	2	3	4	5	6	7					
11	0	0	0	0	0	0	0	0				
12	7	2	3	4	11	6	7	8	3			
13	0	0	0	0	0	0	0	0	0	0		
14	1	2	3	4	5	6	7	8	9	10	11	
15	3	8	0	9	5	3	3	5	9	0	8	3

Solutions → Fermat's Theorem

```
PRIMALITY-FERMATTHEOREM($n$)

if $n = 2$ or $n = 3$ then  return prime

for $a \leftarrow 2$ to $n - 2$ do
    // If $(a^{n-1} - 1) \bmod n \neq 0$, then $n$ is definitely composite
    if POWER($a, n - 1, n$) $\neq 1$ then
        return composite

return prime
```

$$\langle \text{Time, Space} \rangle = \langle \mathcal{O}\left(n \log n\right), \Theta\left(1\right) \rangle$$

Power function using repeated squaring

```
POWER($a, b, c$)
Output: Computes $(a^b) \bmod c$ in $\Theta(\log b)$ time
$result \leftarrow 1$
while $b > 0$ do
    if $b \bmod 2 = 1$ then $result \leftarrow (result \times a) \bmod c$
    $b = b/2$; $a = (a \times a) \bmod c$
return $result$
```

$$\langle \text{Time, Space} \rangle = \langle \Theta(\log b), \Theta(1) \rangle$$

Solutions → Fermat's Test

If a cell in the nth row of the table is nonzero, then n is definitely composite.

Bad news.

- If a cell in the nth row of the table is 0, then n may or may not be prime.
- Formally, for all $n \geq 4$, for some $a \in [2, n-2]$, if $(a^{n-1} - 1) \bmod n = 0$, then n may or may not be prime.
- Example: Cell in $n = 13$, $a = 8$ is zero and n is prime
  Example: Cell in $n = 15$, $a = 11$ is zero but n is composite

Good news.

- There are very few cases when n is composite and it has some cells as zeros in its row
- So, we run this check multiple times to increase our success probability of guessing whether n is prime or composite

Solutions → Fermat's Test

```
PRIMALITY-FERMATTEST($n$)
if $n = 2$ or $n = 3$ then  return prime
// More trials increases the probability of success
for $trials \leftarrow 1$ to $100$ do
    $a \leftarrow$ RandomNumber($\{2, 3, 4, \ldots, n-2\}$)
    // If $(a^{n-1} - 1) \bmod n \neq 0$, then $n$ is definitely composite
    if POWER($a, n-1, n$) $\neq 1$ then
        return composite
return prime                                   // may or may not be prime
```

$$\langle \text{Time, Space} \rangle = \langle \mathcal{O}\left(\#trials \cdot \log n\right), \Theta\left(1\right) \rangle$$

Solutions → Naive AKS's Test

- $n \geq 2$ is prime iff all coefficients, except first and last, of the nth row in the Pascal's triangle are multiples of n

	0	1	2	3	4	5	6	7	8	9
0	1									
1	1	1								
2	1	2	1							
3	1	3	3	1						
4	1	4	6	4	1					
5	1	5	10	10	5	1				
6	1	6	15	20	15	6	1			
7	1	7	21	35	35	21	7	1		
8	1	8	28	56	70	56	28	8	1	
9	1	9	36	84	126	126	84	36	9	1

Solutions → Naive AKS's Test

- $n \geq 2$ is prime iff for all $i \in [1, n-1]$, ${}^nC_i$ is a multiple of n.

```
PRIMALITY-NAIVEAKSTEST($n$)
// Step 1. Compute all required binomial coefficients
$r \leftarrow \lfloor n/2 \rfloor$                    // binomial coefficients are symmetric
Create an array $C[0 \ldots r]$
for $i \leftarrow 0$ to $n$ do
    for $j \leftarrow 0$ to $\min(i, r)$ do
        if $j = 0$ or $j = i$ then  $C[j] \leftarrow 1$
        else  $C[j] \leftarrow C[j-1] + C[j]$
// Step 2. Check if the binomial coefficients are multiples of $n$
for $j \leftarrow 1$ to $r$ do
    if $C[j] \bmod n \neq 0$ then
        return composite
return prime
```

$$\langle \text{Time, Space} \rangle = \left\langle \Theta\left(n^2\right), \Theta\left(n\right) \right\rangle$$

Complexity

Algorithm	Time	Space	Probabilistic?
Naive algorithm	$\mathcal{O}\left(n\right)$	$\Theta\left(1\right)$	✗
School algorithm	$\mathcal{O}\left(\sqrt{n}\right)$	$\Theta\left(1\right)$	✗
Opt. school algo.	$\mathcal{O}\left(\sqrt{n}\right)$	$\Theta\left(1\right)$	✗
SieveOfEratosthenes	$\mathcal{O}\left(\sqrt{n}\log\log n\right)$	$\Theta\left(\sqrt{n}\right)$	✗
Wilson's theorem	$\Theta\left(n\right)$	$\Theta\left(1\right)$	✗
Fermat's theorem	$\mathcal{O}\left(n\log n\right)$	$\Theta\left(1\right)$	✗
Fermat's test	$\mathcal{O}\left(\#trials \cdot \log n\right)$	$\Theta\left(1\right)$	✓
Miller-Rabin's test	$\mathcal{O}\left(\#trials \cdot \log n\right)$	$\Theta\left(1\right)$	✓
Naive AKS test	$\Theta\left(n^2\right)$	$\Theta\left(n\right)$	✗

Largest Subarray Sum

HOME

Problem

- Given an array of reals, find the subarray with the largest sum, and return its sum.
- Input: $[-2, 1, -3, \underbrace{4, -1, 2, 1}, -5, 4]$

  Output: 6
- Input: $[\underbrace{5, 4, -1, 7, 8}]$

  Output: 23

Solutions → Brute Force

```
BruteForce($A[1 \ldots n]$)

$max \leftarrow -\infty$
for $i \leftarrow 1$ to $n$ do
    for $j \leftarrow i$ to $n$ do
        $sum \leftarrow \text{Sum}(A[i \ldots j])$
        if $sum > max$ then $max \leftarrow sum$
return $max$
```

$$\langle \text{Time, Space} \rangle = \left\langle \Theta\left(n^3\right), \Theta\left(1\right) \right\rangle$$

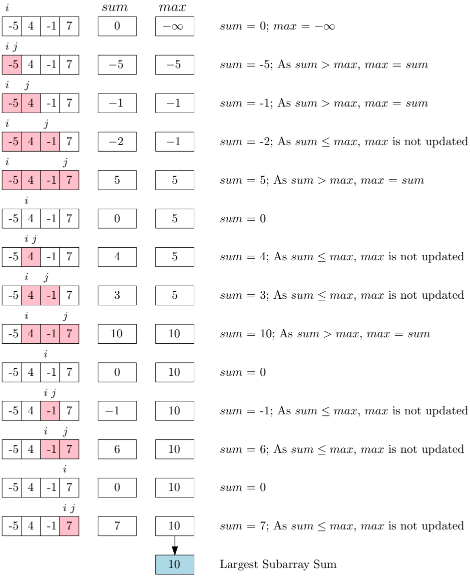
Solutions → Optimized Brute Force

- For all subarrays, calculate the sum of the subarray as you travel the array.

```
OptimizedBruteForce($A[1 \ldots n]$)
$max \leftarrow -\infty$
for $i \leftarrow 1$ to $n$ do
    $sum \leftarrow 0$
    for $j \leftarrow i$ to $n$ do
        $sum \leftarrow sum + A[j]$
        if $sum > max$ then $max \leftarrow sum$
return $max$
```

$$\langle \text{Time, Space} \rangle = \left\langle \Theta\left(n^2\right), \Theta\left(1\right) \right\rangle$$

Solutions → Optimized Brute Force



-5	4	-1	7	sum	max	
i				0	$-\infty$	sum = 0; $max = -\infty$
i j				-5	-5	sum = -5; As $sum > max$, $max = sum$
i	j			-1	-1	sum = -1; As $sum > max$, $max = sum$
i		j		-2	-1	sum = -2; As $sum \leq max$, max is not updated
i			j	5	5	sum = 5; As $sum > max$, $max = sum$
	i			0	5	$sum = 0$
	i j			4	5	sum = 4; As $sum \leq max$, max is not updated
	i	j		3	5	sum = 3; As $sum \leq max$, max is not updated
	i		j	10	10	sum = 10; As $sum > max$, $max = sum$
		i		0	10	$sum = 0$
		i j		-1	10	sum = -1; As $sum \leq max$, max is not updated
		i	j	6	10	sum = 6; As $sum \leq max$, max is not updated
			i	0	10	$sum = 0$
			i j	7	10	sum = 7; As $sum \leq max$, max is not updated
					10	Largest Subarray Sum

Solutions → Divide-and-Conquer

1. Divide the given array in two halves
2. Return the maximum of the following three:
   - (a) Max subarray sum in left half (recurse)
   - (b) Max subarray sum in right half (recurse)
   - (c) Max subarray sum so that the subarray crosses the midpoint

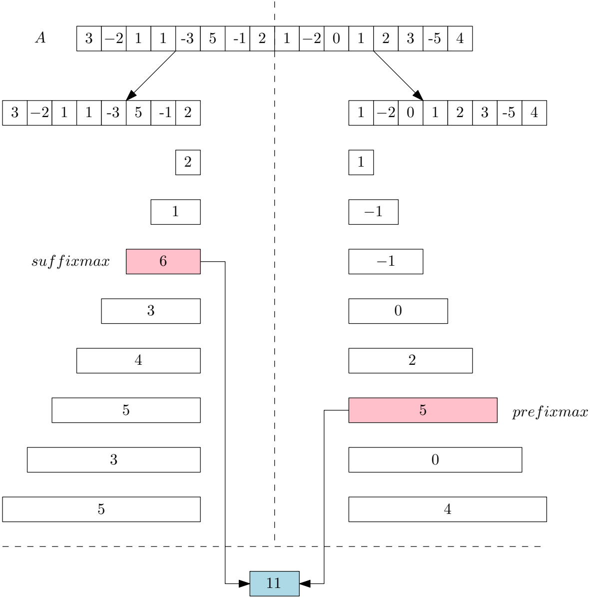
```
$\textsc{DivideAndConquer}(A[1 \ldots n])$

if $n = 1$ then  return $A[1]$
$mid \leftarrow \left\lfloor \frac{n}{2} \right\rfloor$
$S_{\text{left}} \leftarrow \textsc{DivideAndConquer}(A[1 \ldots mid])$
$S_{\text{right}} \leftarrow \textsc{DivideAndConquer}(A[mid + 1 \ldots n])$
$S_{\text{merge}} \leftarrow \textsc{Merge}(A[1 \ldots n], mid)$
return Max$(S_{\text{left}}, S_{\text{right}}, S_{\text{merge}})$
```

$$\langle \text{Time, Space} \rangle = \langle \Theta\left(n \log n\right), \Theta\left(\log n\right) \rangle$$

Solutions → Divide-and-Conquer

```
$\textsc{Merge}(A[1 \ldots n], mid)$
// Find the maximum suffix in the first half
$suffixmax \leftarrow -\infty$; $sum \leftarrow 0$
for $i \leftarrow mid$ to $1$ do
    $sum \leftarrow sum + A[i]$
    if $sum > suffixmax$ then $suffixmax \leftarrow sum$
// Find the maximum prefix in the second half
$prefixmax \leftarrow -\infty$; $sum \leftarrow 0$
for $i \leftarrow mid + 1$ to $n$ do
    $sum \leftarrow sum + A[i]$
    if $sum > prefixmax$ then $prefixmax \leftarrow sum$
// Max subarray sum so that subarray crosses the midpoint
return $(suffixmax + prefixmax)$
```

Solutions → Divide-and-Conquer



$suffixmax + prefixmax$

Solutions → Improved Divide-and-Conquer

1. Create a class called Node for any subproblem (subarray)
2. For any subproblem/subarray, we define a node with four values:
   $sum =$ largest subarray sum, $total =$ total subarray sum
   $prefixmax =$ max prefix sum, $suffixmax =$ max suffix sum
3. Compute and return the sum corresponding to the node of $A[1 \ldots n]$ using D&C

```
ImprovedDivideAndConquer($A[1 \ldots n]$)

Node $answer \leftarrow$ GetMaxSumSubarray($A[1 \ldots n]$)
return $answer.sum$
```

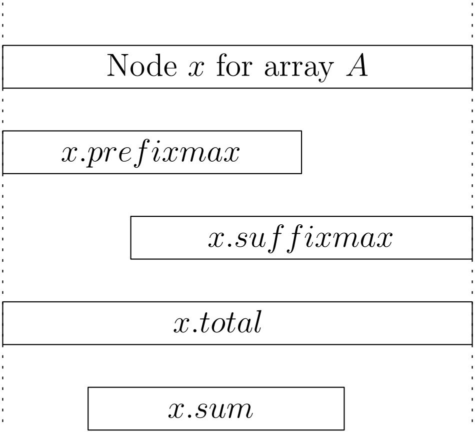
```
GetMaxSumSubarray($A[low \ldots high]$)

if $low = high$ then  return Node($A[low]$)
$mid \leftarrow \lfloor (low + high)/2 \rfloor$
$node_\ell \leftarrow$ GetMaxSumSubarray($A[low \ldots mid]$)
$node_r \leftarrow$ GetMaxSumSubarray($A[mid + 1 \ldots high]$)
return Merge($node_\ell, node_r$)
```

Solutions → Improved Divide-and-Conquer

```
MERGE($\ell$, $r$)
$x \leftarrow$ NODE(0)
// Max prefix subarray sum
$x.prefixmax \leftarrow$ MAX($\ell.prefixmax$, $\ell.total + r.prefixmax$, $\ell.total + r.total$)
// Max suffix subarray sum
$x.suffixmax \leftarrow$ MAX($r.suffixmax$, $r.total + \ell.suffixmax$, $\ell.total + r.total$)
// Total subarray sum
$x.total \leftarrow \ell.total + r.total$
// Max subarray sum
$x.sum \leftarrow$ MAX($x.prefixmax$, $x.suffixmax$, $x.total$, $\ell.sum$, $r.sum$
                    $\ell.suffixmax + r.prefixmax$)
return $x$
```

$$\langle \text{Time, Space} \rangle = \langle \Theta\left(n\right), \Theta\left(\log n\right) \rangle$$

Solutions → Improved Divide-and-Conquer



Node x for array A

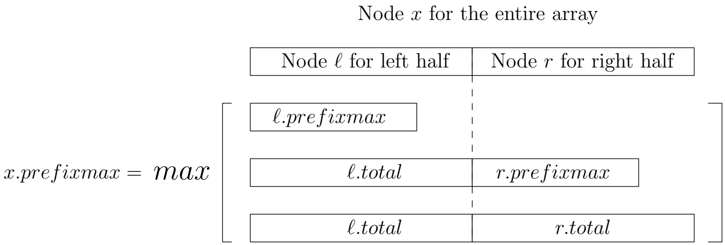
$x.prefixmax$

$x.suffixmax$

$x.total$

$x.sum$

Solutions → Improved Divide-and-Conquer



Node x for the entire array

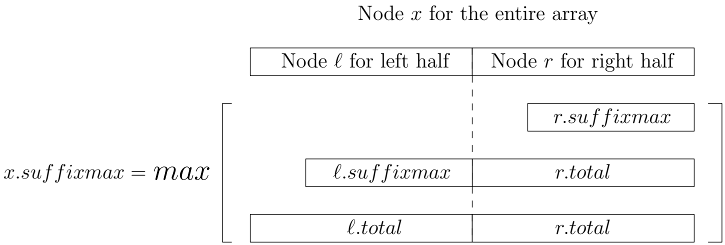
Node ℓ for left half | Node r for right half

$x.prefixmax = max$

$\ell.prefixmax$

$\ell.total$ | $r.prefixmax$

$\ell.total$ | $r.total$



Node x for the entire array

Node ℓ for left half | Node r for right half

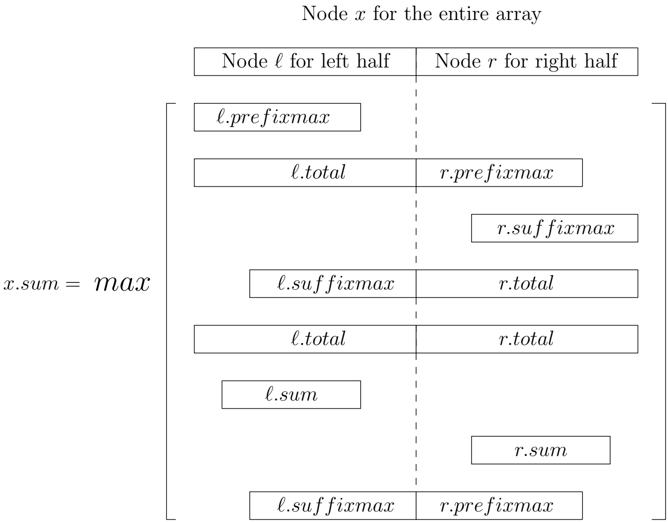
$x.suffixmax = max$

$r.suffixmax$

$\ell.suffixmax$ | $r.total$

$\ell.total$ | $r.total$

Solutions → Improved Divide-and-Conquer



Node x for the entire array

Node ℓ for left half | Node r for right half

$x.sum = max$

$\ell.prefixmax$

$\ell.total$ | $r.prefixmax$

$r.suffixmax$

$\ell.suffixmax$ | $r.total$

$\ell.total$ | $r.total$

$\ell.sum$

$r.sum$

$\ell.suffixmax$ | $r.prefixmax$

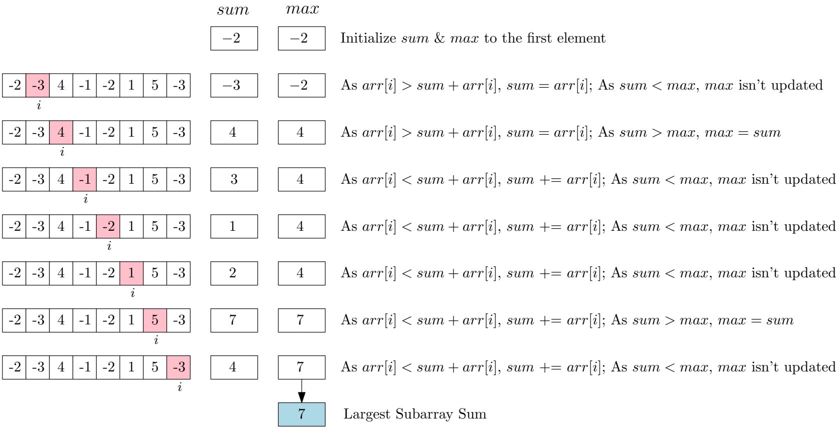
Solutions → Kadane's Algorithm

1. Iterate through the array. For each number, add it to the sum we are building.
2. If sum is smaller than the element value, we know it isn't worth keeping, so throw it away.
3. Update max (max subarray sum) every time we find a new maximum.

```
KadaneAlgorithm($A[1 \dots n]$)
$max \leftarrow A[1]$; $sum \leftarrow A[1]$
for $i \leftarrow 2$ to $n$ do
    // If sum is negative, throw it away. Otherwise, keep adding to it.
    $sum \leftarrow$ Max($A[i], sum + A[i]$)
    $max \leftarrow$ Max($max, sum$)
return $max$
```

$$\langle \text{Time, Space} \rangle = \langle \Theta\left(n\right), \Theta\left(1\right) \rangle$$

Solutions → Kadane's Algorithm



	sum	max	
	-2	-2	Initialize sum & max to the first element
-2 -3 4 -1 -2 1 5 -3 i	-3	-2	As $arr[i] > sum + arr[i]$, $sum = arr[i]$; As $sum < max$, max isn't updated
-2 -3 4 -1 -2 1 5 -3 i	4	4	As $arr[i] > sum + arr[i]$, $sum = arr[i]$; As $sum > max$, $max = sum$
-2 -3 4 -1 -2 1 5 -3 i	3	4	As $arr[i] < sum + arr[i]$, $sum \mathrel{+}= arr[i]$; As $sum < max$, max isn't updated
-2 -3 4 -1 -2 1 5 -3 i	1	4	As $arr[i] < sum + arr[i]$, $sum \mathrel{+}= arr[i]$; As $sum < max$, max isn't updated
-2 -3 4 -1 -2 1 5 -3 i	2	4	As $arr[i] < sum + arr[i]$, $sum \mathrel{+}= arr[i]$; As $sum < max$, max isn't updated
-2 -3 4 -1 -2 1 5 -3 i	7	7	As $arr[i] < sum + arr[i]$, $sum \mathrel{+}= arr[i]$; As $sum > max$, $max = sum$
-2 -3 4 -1 -2 1 5 -3 i	4	7	As $arr[i] < sum + arr[i]$, $sum \mathrel{+}= arr[i]$; As $sum < max$, max isn't updated
		7	Largest Subarray Sum

Complexity

Algorithm	Time	Space
Brute force	$\Theta\left(n^3\right)$	$\Theta\left(1\right)$
Optimized brute force	$\Theta\left(n^2\right)$	$\Theta\left(1\right)$
Divide-and-conquer	$\Theta\left(n \log n\right)$	$\Theta\left(\log n\right)$
Improved divide-and-conquer	$\Theta\left(n\right)$	$\Theta\left(\log n\right)$
Kadane's algorithm	$\Theta\left(n\right)$	$\Theta\left(1\right)$