Recursion (Cont.), Recurrences and Merge Sort

Subhabrata Samajder

IIIT, Delhi

Winter Semester,
11^{th} March, 2023

Recursion: A Recap (Cont.)

Fibonacci Sequence

Fibonacci sequence is defined recursively as

$$f_1 = 1, \quad f_2 = 1, \quad f_{i+1} = f_i + f_{i-1} \quad \text{for } i = 1, 2, \ldots$$

Every element ($i \geq 3$) is the sum of it's previous two elements.

The sequence begins as $1, 1, 2, 3, 5, \ldots$

Fibonacci Sequence

Consider the sequence f_{n+1}/f_n:

$$
\begin{aligned}
2/1 &= 2.0 \quad \text{(bigger)} \\
3/2 &= 1.5 \quad \text{(smaller)} \\
5/3 &= 1.67 \quad \text{(bigger)} \\
8/5 &= 1.6 \quad \text{(smaller)} \\
13/8 &= 1.625 \quad \text{(bigger)} \\
21/13 &= 1.615 \quad \text{(smaller)} \\
34/21 &= 1.619 \quad \text{(bigger)} \\
55/34 &= 1.618 \quad \text{(smaller)} \\
89/55 &= 1.618
\end{aligned}
$$

Fibonacci Sequence

Consider the sequence f_{n+1}/f_n:

$$
\begin{aligned}
2/1 &= 2.0 \quad \text{(bigger)}\\
3/2 &= 1.5 \quad \text{(smaller)}\\
5/3 &= 1.67 \quad \text{(bigger)}\\
8/5 &= 1.6 \quad \text{(smaller)}\\
13/8 &= 1.625 \quad \text{(bigger)}\\
21/13 &= 1.615 \quad \text{(smaller)}\\
34/21 &= 1.619 \quad \text{(bigger)}\\
55/34 &= 1.618 \quad \text{(smaller)}\\
89/55 &= 1.618
\end{aligned}
$$

Note:

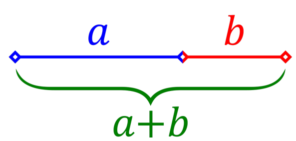
- This sequence seem to be converging!
- It converges to the *golden ratio*.

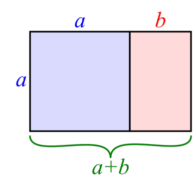
Golden Ratio

$$\varphi \;=\; \frac{1+\sqrt{5}}{2} \;\approx\; 1.6180339887498948482$$

- It is a special number.
- Couple of ways to visually understand it are with

a *line segment*   *Golden rectangles*





$$\frac{a+b}{a} = \frac{a}{b} = \varphi$$

- It is an *irrational number* that is a root of the quadratic equation

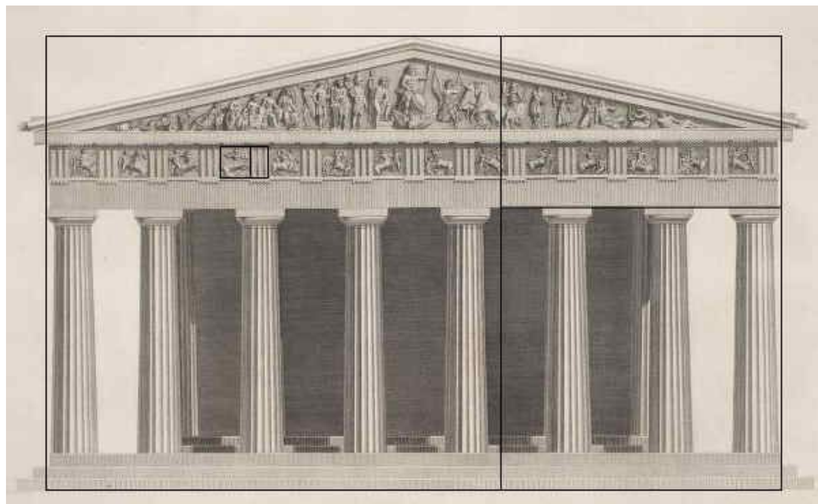
$$x^2 - x - 1 \;=\; 0$$

Golden Ratio

- Reciprocal of φ, i.e., φ^{-1}:
  - $f_n/f_{n+1} \to 0.618$ as $n \to \infty$.
  - This is the reciprocal of φ: $1/1.618 = 0.618$.
  - It is highly unusual for the decimal representation of the fractional part of a number and its reciprocal to be exactly the same.

Golden Ratio

- **Some examples:**



The ancient temple in Greece fits almost precisely into a golden rectangle.

Golden Ratio

- **Some examples:**

Butterflies.

Recursive Fibonacci Sequence: Function Calls

```
int RecFibonacci (int n) {
  if (n <= 1)
    return n;
  else
    return (RecFibonacci(n - 1) + RecFibonacci(n - 2)); }
```

Recursive Fibonacci Sequence: Function Calls

Value of n	Value of RecFibonacci(n)	Number of function calls required to recursively compute RecFibonacci(n)
0	0	1
1	1	1
2	1	3
......		
23	28657	92735
24	46368	150049
......		
42	267914296	866988873
43	433494437	1402817465

Requires a large number of function calls even for moderate values of n.

Recursion: Summary

- It is seductive to use recursion.
- But one must be careful about run-time limitations and inefficiencies.

Recursion: Summary

- It is seductive to use recursion.
- But one must be careful about run-time limitations and inefficiencies.
- It is sometimes necessary to recode to an equivalent iterative method.

Recursion: Summary

- It is seductive to use recursion.
- But one must be careful about run-time limitations and inefficiencies.
- It is sometimes necessary to recode to an equivalent iterative method.
- Some programmers feel that because the use of recursion is inefficient, it should not be used.

Recursion: Summary

- It is seductive to use recursion.
- But one must be careful about run-time limitations and inefficiencies.
- It is sometimes necessary to recode to an equivalent iterative method.
- Some programmers feel that because the use of recursion is inefficient, it should not be used.
- The inefficiencies, however, are often of little consequence - as in the case of the quicksort algorithm.

Recursion: Summary

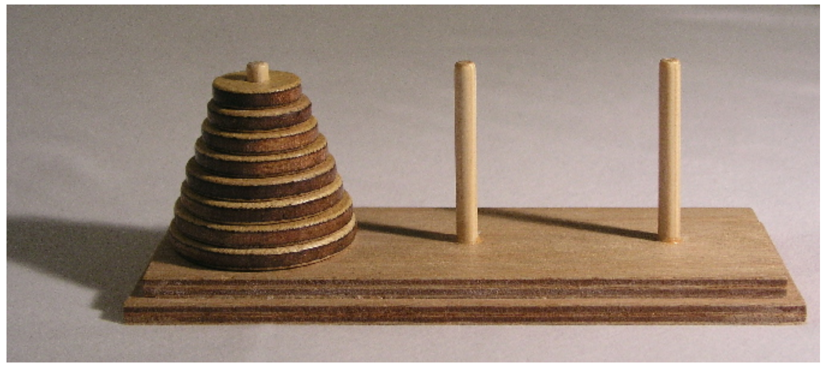
- It is seductive to use recursion.
- But one must be careful about run-time limitations and inefficiencies.
- It is sometimes necessary to recode to an equivalent iterative method.
- Some programmers feel that because the use of recursion is inefficient, it should not be used.
- The inefficiencies, however, are often of little consequence - as in the case of the quicksort algorithm.
- For many applications, recursive code is easier to write, understand, maintain.

Recursion: Summary

- It is seductive to use recursion.
- But one must be careful about run-time limitations and inefficiencies.
- It is sometimes necessary to recode to an equivalent iterative method.
- Some programmers feel that because the use of recursion is inefficient, it should not be used.
- The inefficiencies, however, are often of little consequence - as in the case of the quicksort algorithm.
- For many applications, recursive code is easier to write, understand, maintain.
- These reasons often prescribe its use.

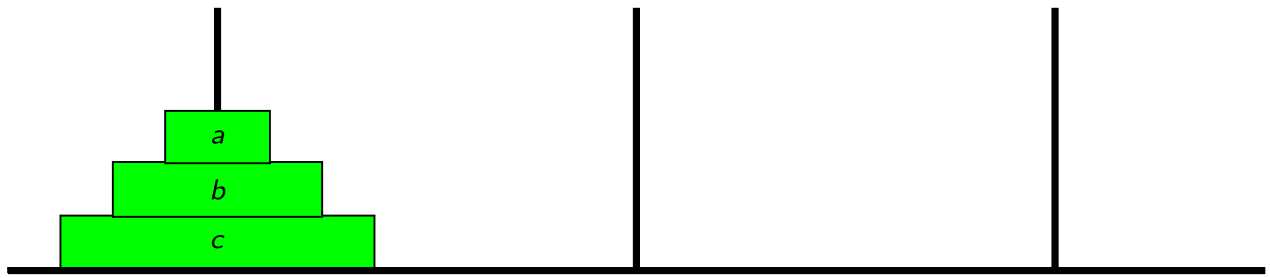
Towers of Hanoi

Towers of Hanoi: Problem Statement



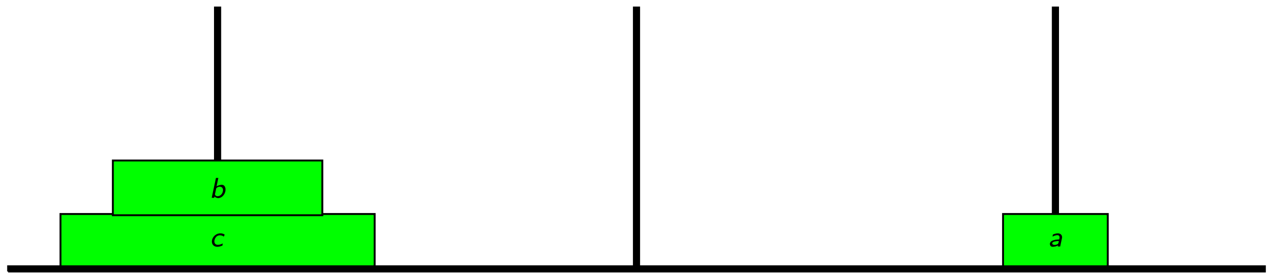
- There are three towers.
- n disks of decreasing radius are placed on the 1^{st} tower.
- Move all of the disks from the 1^{st} tower to the 3^{rd} tower.
- **Condition:** At no moment of time can a larger disk be placed on top of smaller disks.
- The remaining tower can be used to temporarily hold disks.

Towers of Hanoi: Solution for $n = 3$



Towers of Hanoi: Solution for $n = 3$

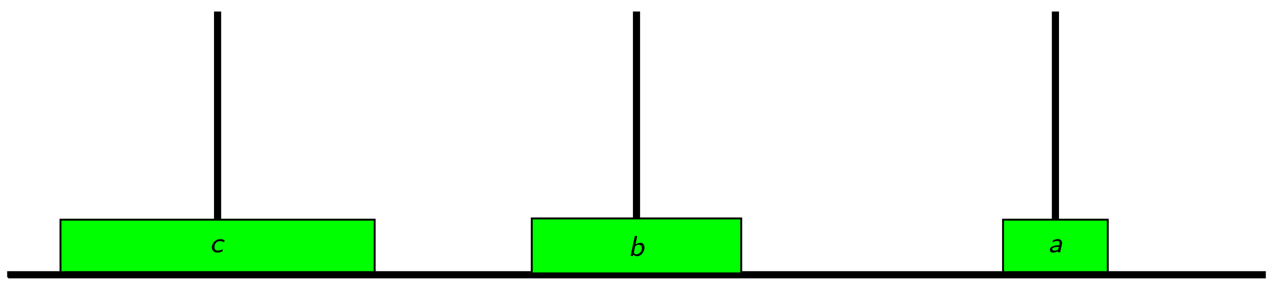
Step 1: Move disks a to tower 3.



Towers of Hanoi: Solution for $n = 3$

Step 1: Move disks a to tower 3.

Step 2: Move disks b to tower 2.

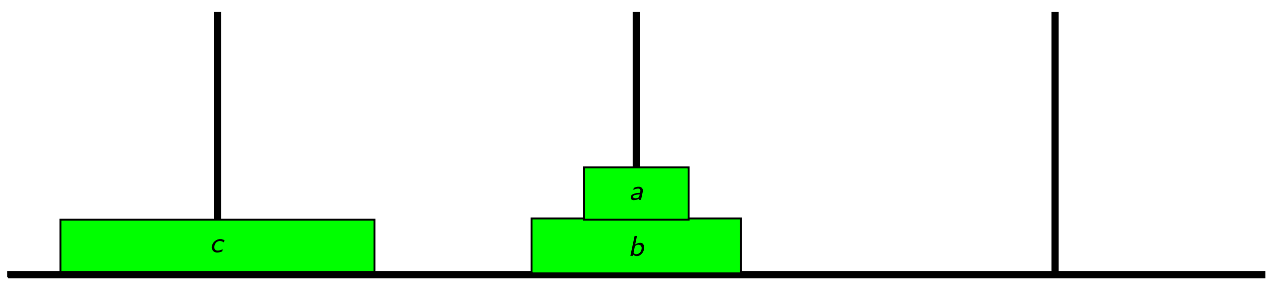


Towers of Hanoi: Solution for $n = 3$

Step 1: Move disks a to tower 3.

Step 2: Move disks b to tower 2.

Step 3: Move disks a to tower 2.



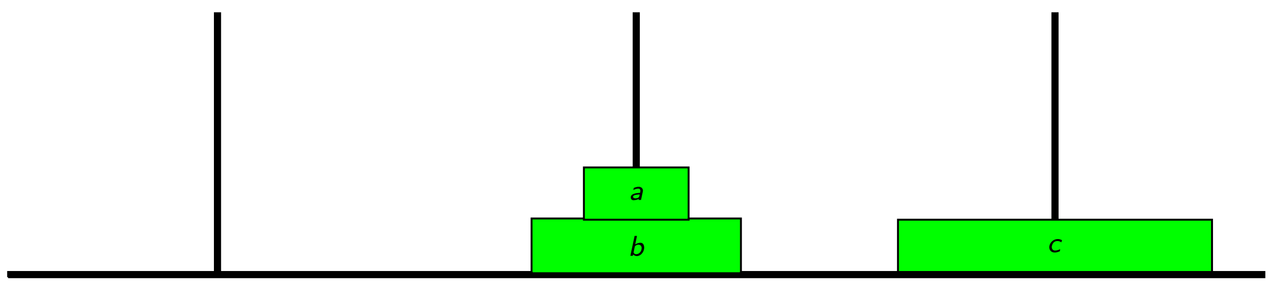
Towers of Hanoi: Solution for $n = 3$

Step 1: Move disks a to tower 3.

Step 2: Move disks b to tower 2.

Step 3: Move disks a to tower 2.

Step 4: Move disks c to tower 3.



Towers of Hanoi: Solution for $n = 3$

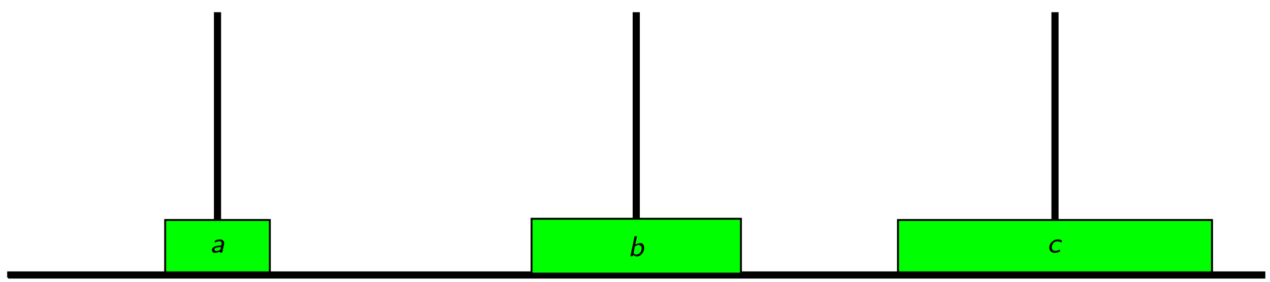
Step 1: Move disks a to tower 3.

Step 2: Move disks b to tower 2.

Step 3: Move disks a to tower 2.

Step 4: Move disks c to tower 3.

Step 5: Move disks a to tower 1.



Towers of Hanoi: Solution for $n = 3$

Step 1: Move disks a to tower 3.

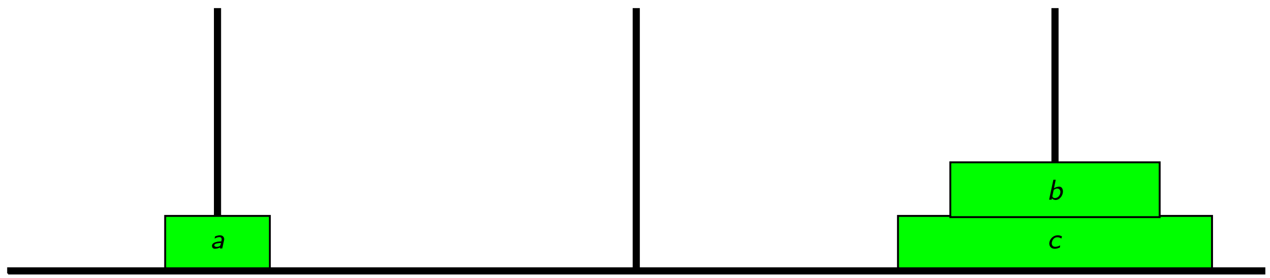
Step 2: Move disks b to tower 2.

Step 3: Move disks a to tower 2.

Step 4: Move disks c to tower 3.

Step 5: Move disks a to tower 1.

Step 6: Move disks b to tower 3.



Towers of Hanoi: Solution for $n = 3$

Step 1: Move disks a to tower 3.

Step 2: Move disks b to tower 2.

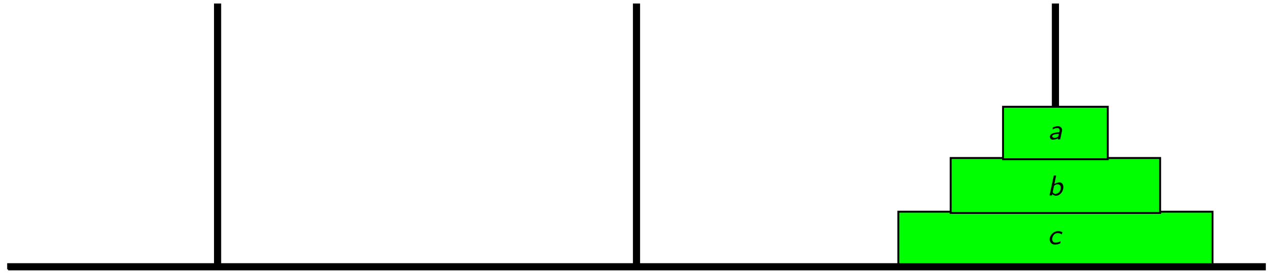
Step 3: Move disks a to tower 2.

Step 4: Move disks c to tower 3.

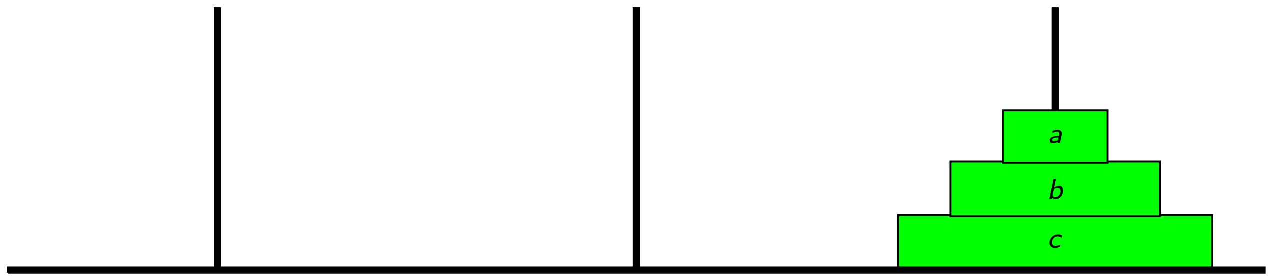
Step 5: Move disks a to tower 1.

Step 6: Move disks b to tower 3.

Step 7: Move disks a to tower 3.



Towers of Hanoi: Solution for $n = 3$



Homework:

- Write a recursive algorithm that solves the Towers of Hanoi problem for n disks.
- Implement your algorithm in C.

Recurrences

Recurrence

Definition

A **recurrence relation** is an equation that expresses each element of a sequence $\{a_n\}_{n=0}^{\infty}$ as a function of the preceding ones, i.e,

$$a_n = \psi(a_0, a_1, \ldots, a_{n-1}).$$

Why Recurrences?

Why Recurrences?

- Computing time/space complexity $\equiv$ a counting problem.

Why Recurrences?

- Computing time/space complexity $\equiv$ a counting problem.
- $\therefore$ a sequence.

Why Recurrences?

- Computing time/space complexity $\equiv$ a counting problem.
- $\therefore$ a sequence.
- **Recursive algorithms:** It is not easy to compute the number of instructions by looking at code.

Why Recurrences?

- Computing time/space complexity $\equiv$ a counting problem.
- $\therefore$ a sequence.
- **Recursive algorithms:** It is not easy to compute the number of instructions by looking at code.
  - The number of instructions in one instance of function call depends on the number of instructions executed when recursive calls are made.

Why Recurrences?

- Computing time/space complexity $\equiv$ a counting problem.
- $\therefore$ a sequence.
- **Recursive algorithms:** It is not easy to compute the number of instructions by looking at code.
  - The number of instructions in one instance of function call depends on the number of instructions executed when recursive calls are made.
  - In such cases it is easier for us to express it as some *recurrence relation* of the times/space complexity.

Why Recurrences?

- Computing time/space complexity $\equiv$ a counting problem.
- $\therefore$ a sequence.
- **Recursive algorithms:** It is not easy to compute the number of instructions by looking at code.
    - The number of instructions in one instance of function call depends on the number of instructions executed when recursive calls are made.
    - In such cases it is easier for us to express it as some *recurrence relation* of the times/space complexity.
- Appears frequently in the analysis of algorithms.

Questions?

1. How to form recurrence relation for a counting problem?

Questions?

1. How to form recurrence relation for a counting problem? Easy!!

Questions?

1. How to form recurrence relation for a counting problem? Easy!!

2. How to solve such relations?

Questions?

1. How to form recurrence relation for a counting problem? Easy!!

2. How to solve such relations?

Outline

- We briefly discuss few useful technique for solving recurrences.

- Present general solutions of two classes of recurrences that are among the most common recurrences involved in analyzing algorithms.

Intelligent Guesses

- Guessing a solution may seem like a nonscientific method!
- But, keeping our pride aside, it works very well for a wide class of recurrence relations.
- It works even better when we are *not* trying to find the exact solution, but only an upper bound.
- **Why guess?** Proving a certain bound is valid is easier than deriving that bound.

Fibonacci Sequence

$$F(n) = F(n-1) + F(n-2), \quad F(1) = 1, F(2) = 1.$$

Fibonacci Sequence

$$F(n) = F(n-1) + F(n-2), \quad F(1) = 1, F(2) = 1.$$

- Can compute the value of the function for every n.

  Example:

  $$F(3) = F(2) + F(1) = 2,\ F(4) = F(3) + F(2) = 3, \ldots .$$

Fibonacci Sequence

$$F(n) = F(n-1) + F(n-2), \quad F(1) = 1, F(2) = 1.$$

- Can compute the value of the function for every n.

  Example:

  $$F(3) = F(2) + F(1) = 2,\ F(4) = F(3) + F(2) = 3, \ldots.$$

- **Note:** By definition we need $n-2$ steps to compute $F(n)$.

Fibonacci Sequence

$$F(n) = F(n-1) + F(n-2), \quad F(1) = 1, F(2) = 1.$$

- Can compute the value of the function for every n.

  Example:

  $$F(3) = F(2) + F(1) = 2, \ F(4) = F(3) + F(2) = 3, \ldots.$$

- **Note:** By definition we need $n-2$ steps to compute $F(n)$.
- Would be more convenient to have an explicit (or closed-form) expression for $F(n)$.
  - It would enable us to compute $F(n)$ quickly.
  - We can also compare $F(n)$ with other known functions.

Fibonacci Sequence

$$F(n) = F(n-1) + F(n-2), \quad F(1) = 1, F(2) = 1.$$

- $F(n)$ is the sum of two previous values.

Fibonacci Sequence

$$F(n) = F(n-1) + F(n-2), \quad F(1) = 1, F(2) = 1.$$

- $F(n)$ is the sum of two previous values.
- **Possible guess:** $F(n)$ is doubled every time, i.e., $F(n) \approx 2^n$.

Fibonacci Sequence

$$F(n) = F(n-1) + F(n-2), \quad F(1) = 1, F(2) = 1.$$

- $F(n)$ is the sum of two previous values.
- **Possible guess:** $F(n)$ is doubled every time, i.e., $F(n) \approx 2^n$.
- Let $F(n) = ca^n$, then we get

  $$ca^n = ca^{n-1} + ca^{n-2} \Rightarrow a^2 = a + 1 \quad \text{(Characteristic equation)}.$$

Fibonacci Sequence

$$F(n) = F(n-1) + F(n-2), \quad F(1) = 1, F(2) = 1.$$

- $F(n)$ is the sum of two previous values.
- **Possible guess:** $F(n)$ is doubled every time, i.e., $F(n) \approx 2^n$.
- Let $F(n) = ca^n$, then we get

  $$ca^n = ca^{n-1} + ca^{n-2} \Rightarrow a^2 = a + 1 \quad (\text{Characteristic equation}).$$

- **Solving:** $a_1 = (1 + \sqrt{5})/2$ (> 0) and $a_2 = (1 - \sqrt{5})/2$ (< 0).

Fibonacci Sequence

$$F(n) = F(n-1) + F(n-2), \quad F(1) = 1, F(2) = 1.$$

- $F(n)$ is the sum of two previous values.
- **Possible guess:** $F(n)$ is doubled every time, i.e., $F(n) \approx 2^n$.
- Let $F(n) = ca^n$, then we get

  $$ca^n = ca^{n-1} + ca^{n-2} \Rightarrow a^2 = a + 1 \quad (\text{Characteristic equation}).$$

- **Solving:** $a_1 = (1+\sqrt{5})/2$ (> 0) and $a_2 = (1-\sqrt{5})/2$ (< 0).
- $\therefore$ $F(n) = \mathcal{O}((a_1)^n)$.
  - Find a constant c such that $c(a_1)^n \geq F(1)$ and $F(2)$.

Fibonacci Sequence: Exact Solution

- Need to consider the initial values more carefully.

Fibonacci Sequence: Exact Solution

- Need to consider the initial values more carefully.
- **General Solution:** $c_1(a_1)^n + c_2(a_2)^n$.

Fibonacci Sequence: Exact Solution

- Need to consider the initial values more carefully.
- **General Solution:** $c_1(a_1)^n + c_2(a_2)^n$.
- Find c_1 and c_2, s.t., $F(1) = 1$ and $F(2) = 1$.

Fibonacci Sequence: Exact Solution

- Need to consider the initial values more carefully.
- **General Solution:** $c_1(a_1)^n + c_2(a_2)^n$.
- Find c_1 and c_2, s.t., $F(1) = 1$ and $F(2) = 1$.
- **Solving:** $c_1 = 1/\sqrt{5}$, and $c_2 = -1/\sqrt{5}$.

Fibonacci Sequence: Exact Solution

- Need to consider the initial values more carefully.
- **General Solution:** $c_1(a_1)^n + c_2(a_2)^n$.
- Find c_1 and c_2, s.t., $F(1) = 1$ and $F(2) = 1$.
- **Solving:** $c_1 = 1/\sqrt{5}$, and $c_2 = -1/\sqrt{5}$.
- **Exact solution:**

$$F(n) = \frac{1}{\sqrt{5}} \left[\frac{1+\sqrt{5}}{2}\right]^n - \frac{1}{\sqrt{5}} \left[\frac{1-\sqrt{5}}{2}\right]^n .$$

Fibonacci Sequence: Exact Solution

- Need to consider the initial values more carefully.
- **General Solution:** $c_1(a_1)^n + c_2(a_2)^n$.
- Find c_1 and c_2, s.t., $F(1) = 1$ and $F(2) = 1$.
- **Solving:** $c_1 = 1/\sqrt{5}$, and $c_2 = -1/\sqrt{5}$.
- **Exact solution:**

$$F(n) = \frac{1}{\sqrt{5}}\left[\frac{1+\sqrt{5}}{2}\right]^n - \frac{1}{\sqrt{5}}\left[\frac{1-\sqrt{5}}{2}\right]^n.$$

Note: This idea, can be used to solve recurrences of the form

$$F(n) = b_1 F(n-1) + b_2 F(n-2) + \cdots + b_k F(n-k) \quad (k \text{ constant}).$$

Sorting

Sorting Problem

Sorting Problem

Given n numbers $x_1, x_2, \ldots, x_n$ arrange them in *increasing order*. In other words, find a sequence of distinct indices $1 \leq i_1, i_2, \ldots, i_n \leq n$, such that $x_{i_1} \leq x_{i_2} \leq \cdots \leq x_{i_n}$.

In-place Sorting: A sorting algorithm is called in-place if no additional memory is used besides the input array.

Sorting Problem

Sorting Problem

Given n numbers $x_1, x_2, \ldots, x_n$ arrange them in *increasing order*. In other words, find a sequence of distinct indices $1 \le i_1, i_2, \ldots, i_n \le n$, such that $x_{i_1} \le x_{i_2} \le \cdots \le x_{i_n}$.

In-place Sorting: A sorting algorithm is called in-place if no additional memory is used besides the input array.

Mergesort Algorithm

The Merge Algorithm

Problem: Suppose we have two lists $A = (a_1, a_2, \ldots, a_n)$ and $B = (b_1, b_2, \ldots, b_m)$ of numbers sorted in an increasing order. Merge them to get a bigger sorted list.

The Merge Algorithm

Problem: Suppose we have two lists $A = (a_1, a_2, \ldots, a_n)$ and $B = (b_1, b_2, \ldots, b_m)$ of numbers sorted in an increasing order. Merge them to get a bigger sorted list.

Basic Idea: For each numbers in the second set, find it's correct place in the first set.

The Merge Algorithm: An Example

B :	1	2	5	6	8	9	10	12

A :	3	4	7	11	13	14	15	16

The Merge Algorithm: An Example

B :								
	1	2	5	6	8	9	10	12

A :								
	3	4	7	11	13	14	15	16

M :																

The Merge Algorithm: An Example

B :

1	2	5	6	8	9	10	12

A :

3	4	7	11	13	14	15	16

M :

The Merge Algorithm: An Example

B :	~~1~~	2	5	6	8	9	10	12

A :	3	4	7	11	13	14	15	16

M :	1															

The Merge Algorithm: An Example

B :	~~1~~	~~2~~	5	6	8	9	10	12

A :	3	4	7	11	13	14	15	16

M :	1	2														

The Merge Algorithm: An Example

B :	$\cancel{1}$	$\cancel{2}$	5	6	8	9	10	12

A :	$\cancel{3}$	4	7	11	13	14	15	16

M :	1	2	3													

The Merge Algorithm: An Example

B :	~~1~~	~~2~~	5	6	8	9	10	12

A :	~~3~~	~~4~~	7	11	13	14	15	16

M :	1	2	3	4												

The Merge Algorithm: An Example

B :	~~1~~	~~2~~	~~5~~	6	8	9	10	12

A :	~~3~~	~~4~~	7	11	13	14	15	16

M :	1	2	3	4	5											

The Merge Algorithm: An Example

B :	~~1~~	~~2~~	~~5~~	~~6~~	8	9	10	12

A :	~~3~~	~~4~~	7	11	13	14	15	16

M :	1	2	3	4	5	6										

The Merge Algorithm: An Example

$B:$	~~1~~	~~2~~	~~5~~	~~6~~	8	9	10	12

$A:$	~~3~~	~~4~~	~~7~~	11	13	14	15	16

$M:$	1	2	3	4	5	6	7									

The Merge Algorithm: An Example

B :	~~1~~	~~2~~	~~5~~	~~6~~	~~8~~	9	10	12

A :	~~3~~	~~4~~	~~7~~	11	13	14	15	16

M :	1	2	3	4	5	6	7	8								

The Merge Algorithm: An Example

B :

$\cancel{1}$	$\cancel{2}$	$\cancel{5}$	$\cancel{6}$	$\cancel{8}$	$\cancel{9}$	10	12

A :

$\cancel{3}$	$\cancel{4}$	$\cancel{7}$	11	13	14	15	16

M :

1	2	3	4	5	6	7	8	9							

The Merge Algorithm: An Example

B :	~~1~~	~~2~~	~~5~~	~~6~~	~~8~~	~~9~~	~~10~~	12

A :	~~3~~	~~4~~	~~7~~	11	13	14	15	16

M :	1	2	3	4	5	6	7	8	9	10						

The Merge Algorithm: An Example

B :	~~1~~	~~2~~	~~5~~	~~6~~	~~8~~	~~9~~	~~10~~	12

A :	~~3~~	~~4~~	~~7~~	~~11~~	13	14	15	16

M :	1	2	3	4	5	6	7	8	9	10	11					

The Merge Algorithm: An Example

B :	~~1~~	~~2~~	~~5~~	~~6~~	~~8~~	~~9~~	~~10~~	~~12~~

A :	~~3~~	~~4~~	~~7~~	~~11~~	13	14	15	16

M :	1	2	3	4	5	6	7	8	9	10	11	12				

The Merge Algorithm: An Example

B :

~~1~~	~~2~~	~~5~~	~~6~~	~~8~~	~~9~~	~~10~~	~~12~~

A :

~~3~~	~~4~~	~~7~~	~~11~~	~~13~~	14	15	16

M :

1	2	3	4	5	6	7	8	9	10	11	12	13			

The Merge Algorithm: An Example

B : | ~~1~~ | ~~2~~ | ~~5~~ | ~~6~~ | ~~8~~ | ~~9~~ | ~~10~~ | ~~12~~ |

A : | ~~3~~ | ~~4~~ | ~~7~~ | ~~11~~ | ~~13~~ | ~~14~~ | 15 | 16 |

M : | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | | |

The Merge Algorithm: An Example

B :	~~1~~	~~2~~	~~5~~	~~6~~	~~8~~	~~9~~	~~10~~	~~12~~

A :	~~3~~	~~4~~	~~7~~	~~11~~	~~13~~	~~14~~	~~15~~	16

M :	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	

The Merge Algorithm: An Example

B :	~~1~~	~~2~~	~~5~~	~~6~~	~~8~~	~~9~~	~~10~~	~~12~~

A :	~~3~~	~~4~~	~~7~~	~~11~~	~~13~~	~~14~~	~~15~~	~~16~~

M :	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

The Merge Algorithm (Cont.)

- Scan the first set until the right place to insert b_1 is found.

The Merge Algorithm (Cont.)

- Scan the first set until the right place to insert b_1 is found.
- Insert b_1.

The Merge Algorithm (Cont.)

- Scan the first set until the right place to insert b_1 is found.
- Insert b_1.
- Continue the scan from that place until the right place to insert b_2 is found.

The Merge Algorithm (Cont.)

- Scan the first set until the right place to insert b_1 is found.
- Insert b_1.
- Continue the scan from that place until the right place to insert b_2 is found.
- Repeat this for all elements of B.

The Merge Algorithm (Cont.)

Note:

- Since the b's are sorted, we never have to go back.

The Merge Algorithm (Cont.)

Note:

- Since the b's are sorted, we never have to go back.
- The total number of comparisons, in the worst case, is $m + n$.

The Merge Algorithm (Cont.)

Note:

- Since the b's are sorted, we never have to go back.
- The total number of comparisons, in the worst case, is $m + n$.

Question: What about data movements?

The Merge Algorithm (Cont.)

Note:

- Since the b's are sorted, we never have to go back.
- The total number of comparisons, in the worst case, is $m + n$.

Question: What about data movements?

- It is inefficient to move elements each time an insertion is performed.

The Merge Algorithm (Cont.)

Note:

- Since the b's are sorted, we never have to go back.
- The total number of comparisons, in the worst case, is $m + n$.

Question: What about data movements?

- It is inefficient to move elements each time an insertion is performed.
- Since the merge produces the elements one by one in sorted order, we copy them to a temporary array.

The Merge Algorithm (Cont.)

Note:

- Since the b's are sorted, we never have to go back.
- The total number of comparisons, in the worst case, is $m + n$.

Question: What about data movements?

- It is inefficient to move elements each time an insertion is performed.
- Since the merge produces the elements one by one in sorted order, we copy them to a temporary array.
- Each element is copied exactly once.

The Merge Algorithm (Cont.)

Note:

- Since the b's are sorted, we never have to go back.
- The total number of comparisons, in the worst case, is $m + n$.

Question: What about data movements?

- It is inefficient to move elements each time an insertion is performed.
- Since the merge produces the elements one by one in sorted order, we copy them to a temporary array.
- Each element is copied exactly once.

Complexity:

- **Time:** $\mathcal{O}(n + m)$ comparisons.
- **Space:** $\mathcal{O}(n + m)$ data.

Question?

Can we use the Merge algorithm to sort a given array?

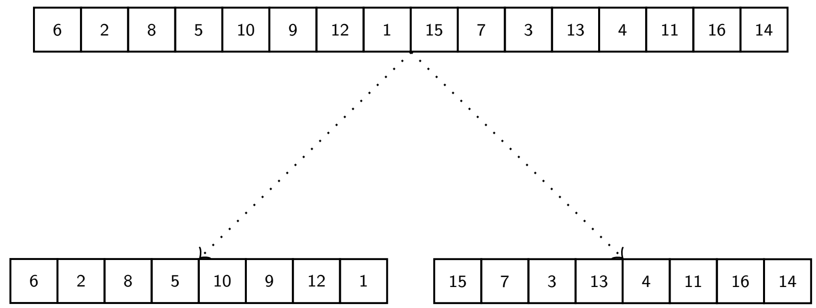
Mergesort: An Example

Splitting Phase:

6	2	8	5	10	9	12	1	15	7	3	13	4	11	16	14

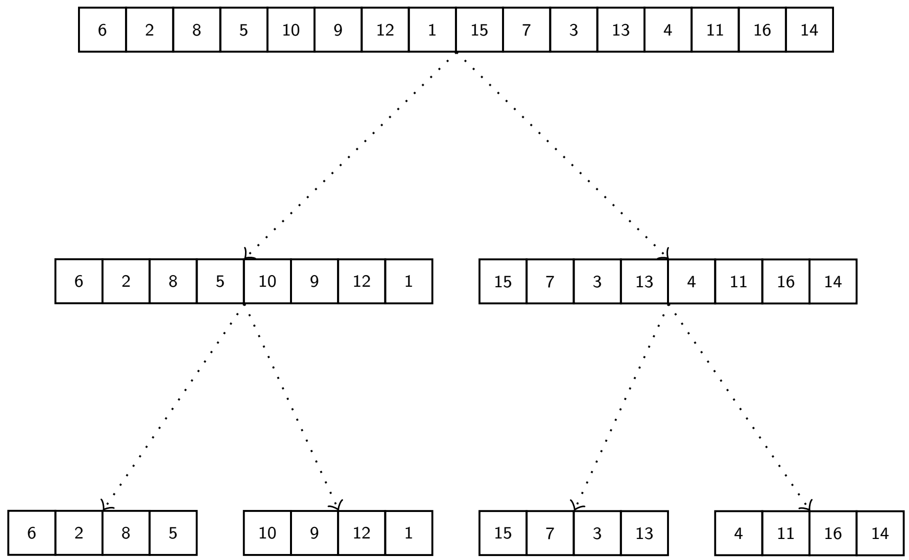
Mergesort: An Example

Splitting Phase:



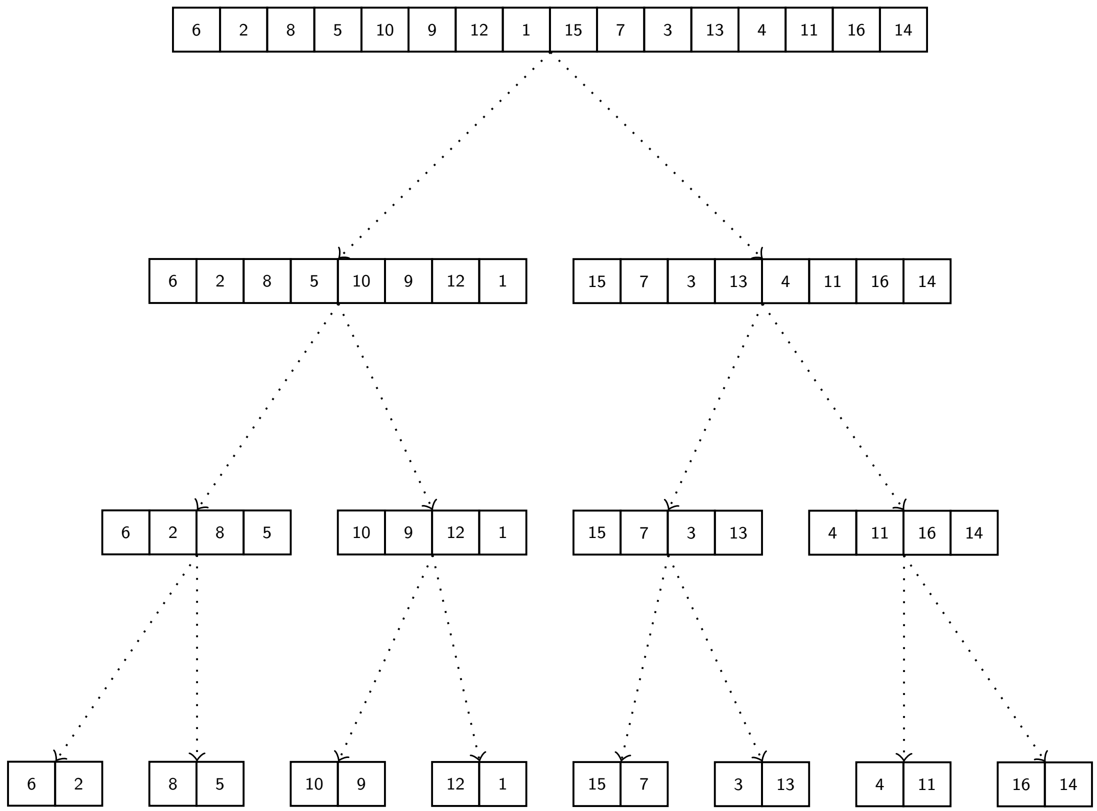
Mergesort: An Example

Splitting Phase:



Mergesort: An Example

Splitting Phase:



Mergesort: An Example

Merging Phase:

6 2	8 5	10 9	12 1	15 7	3 13	4 11	16 14

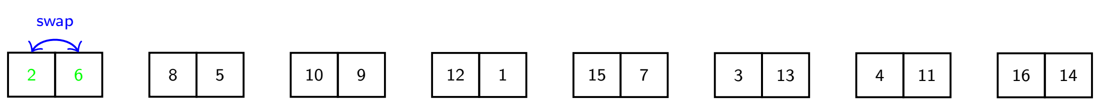
Mergesort: An Example

Merging Phase:

6 2	8 5	10 9	12 1	15 7	3 13	4 11	16 14

Mergesort: An Example

Merging Phase:

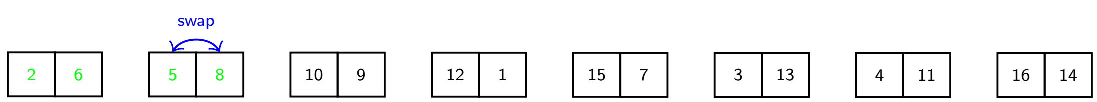


Mergesort: An Example

Merging Phase:

2	6	8	5	10	9	12	1	15	7	3	13	4	11	16	14

Merging Phase:



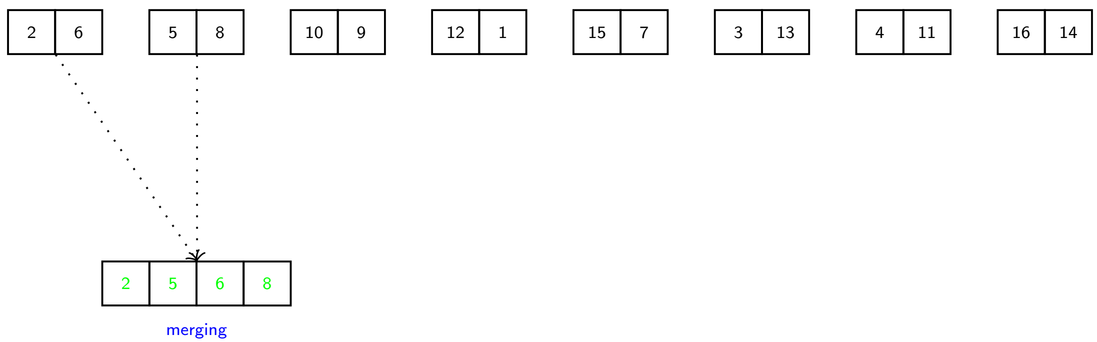
Mergesort: An Example

Merging Phase:

2	6	5	8	10	9	12	1	15	7	3	13	4	11	16	14

Mergesort: An Example

Merging Phase:



Mergesort: An Example

Merging Phase:

10	9		12	1		15	7		3	13

4	11		16	14

2	5	6	8

Mergesort: An Example

Merging Phase:

swap

9	10		12	1		15	7		3	13		4	11		16	14

2	5	6	8

Mergesort: An Example

Merging Phase:

9	10	12	1	15	7	3	13	4	11	16	14

2	5	6	8

Mergesort: An Example

Merging Phase:

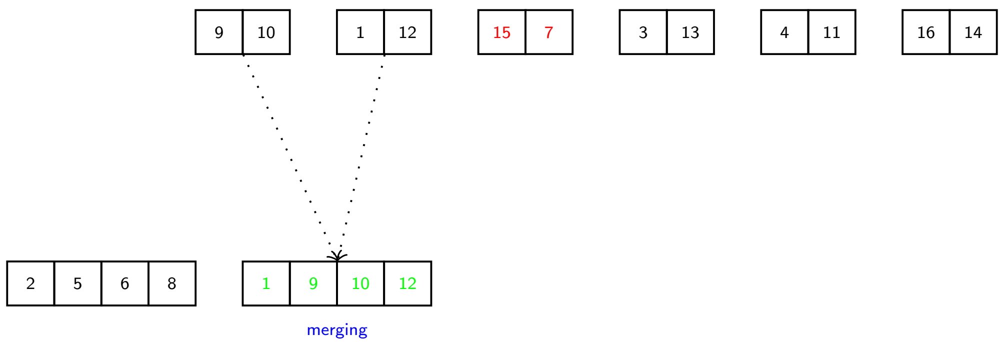
swap

9	10	1	12	15	7	3	13	4	11	16	14

2	5	6	8

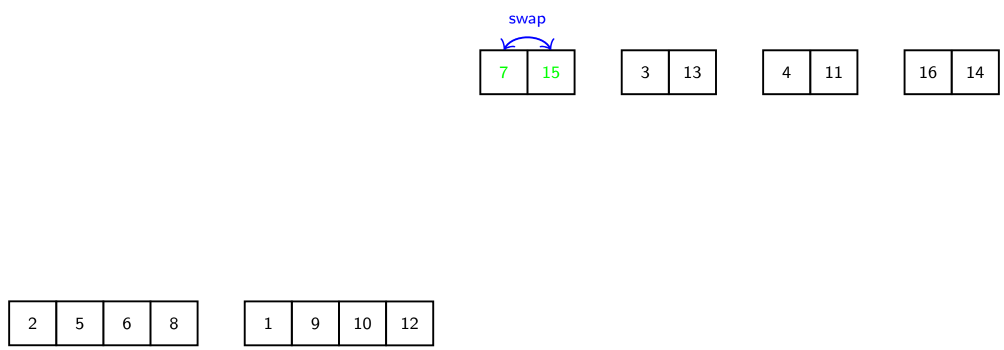
Mergesort: An Example

Merging Phase:



Mergesort: An Example

Merging Phase:



Mergesort: An Example

Merging Phase:

15	7	3	13	4	11	16	14

2	5	6	8	1	9	10	12

Mergesort: An Example

Merging Phase:

15	7

3	13

4	11

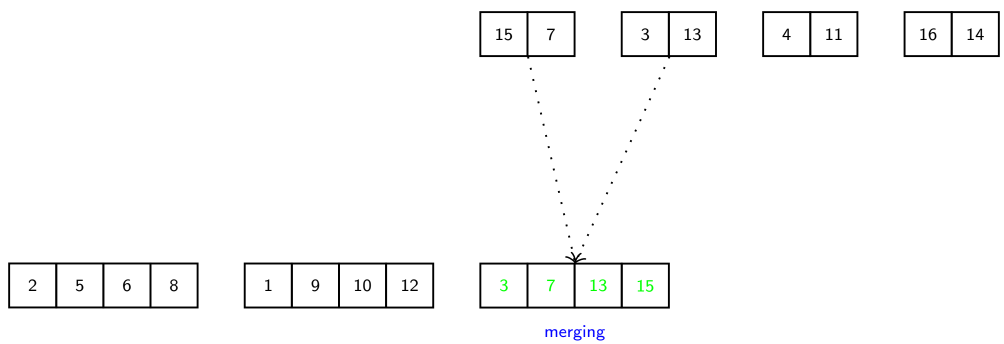
16	14

2	5	6	8

1	9	10	12

Mergesort: An Example

Merging Phase:



Mergesort: An Example

Merging Phase:

4	11

16	14

2	5	6	8

1	9	10	12

3	7	13	15

Mergesort: An Example

Merging Phase:

4	11		16	14	

2	5	6	8		1	9	10	12		3	7	13	15

Mergesort: An Example

Merging Phase:

4	11

16	14

2	5	6	8

1	9	10	12

3	7	13	15

Mergesort: An Example

Merging Phase:

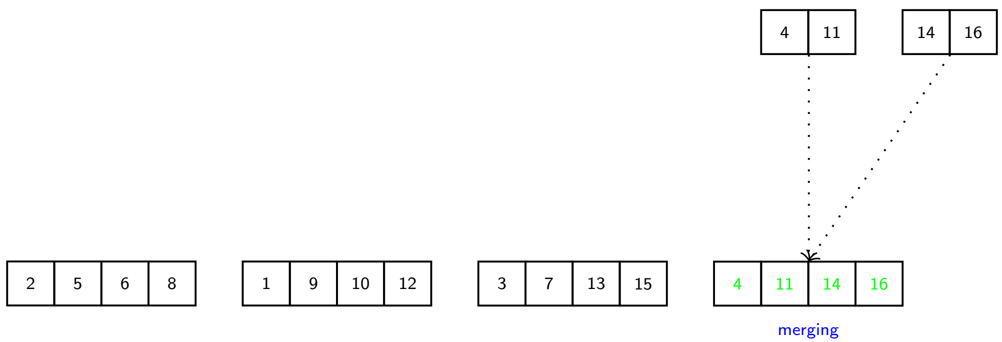
swap

4	11		14	16	

2	5	6	8		1	9	10	12		3	7	13	15

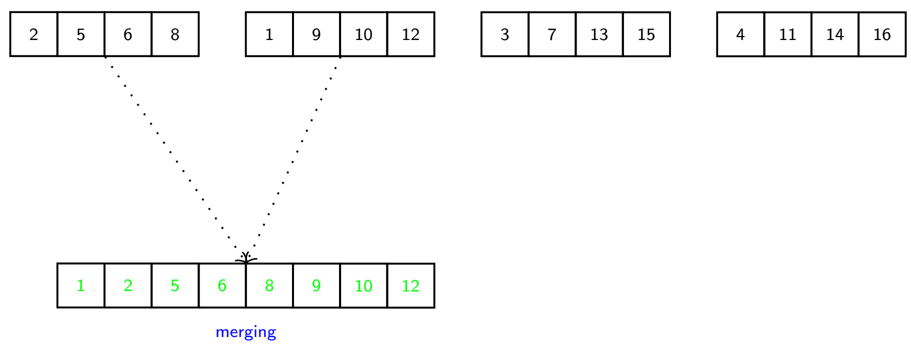
Mergesort: An Example

Merging Phase:



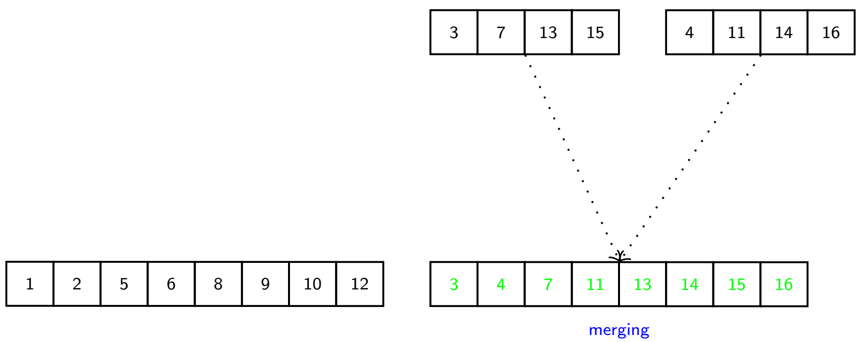
Mergesort: An Example

Merging Phase:



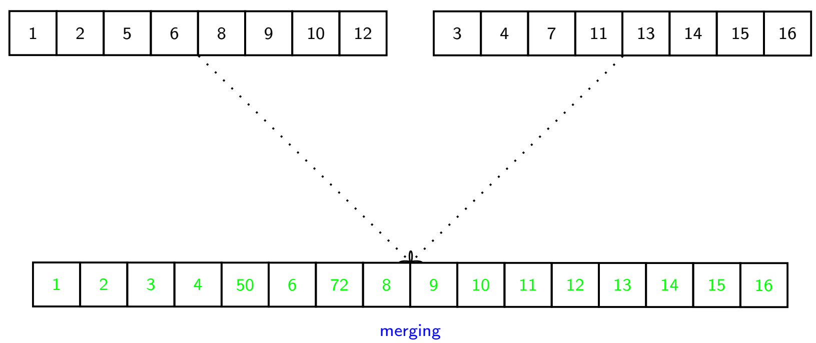
Mergesort: An Example

Merging Phase:



Mergesort: An Example

Merging Phase:



Mergesort

The merge procedure is used as a basis for the divide-and-conquer sorting algorithm, known as the **mergesort**.

Mergesort

The merge procedure is used as a basis for the divide-and-conquer sorting algorithm, known as the **mergesort**.

The Algorithm:

- Divide the sequence into two equal or close-to-equal parts.
- Sort each part separately using *recursion*.
- Merge the two sorted parts into one sorted sequence, using the *merge algorithm*.

Mergesort

The merge procedure is used as a basis for the divide-and-conquer sorting algorithm, known as the **mergesort**.

The Algorithm:

- Divide the sequence into two equal or close-to-equal parts.
- Sort each part separately using *recursion*.
- Merge the two sorted parts into one sorted sequence, using the *merge algorithm*.

Worst-Case Complexity:

$$T(n) \ = \ 2T(n/2) + \mathcal{O}(n)$$

Books Consulted

1. *Introduction to Algorithms: A Creative Approach* by Udi Manber.

2. *Introduction to Algorithms* by Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, Clifford Stein.

Thank You for your kind attention!