Prepared by Julian García, based on material by
David Albrecht and María García de la Banda

Lecture 23
Recursive sorting I

FIT 1008&2085
Introduction to Computer Science

MONASH University
Information Technology

COMMONWEALTH OF AUSTRALIA
Copyright Regulations 1969
WARNING
This material has been reproduced and communicated to you by or on behalf of Monash University pursuant to Part VB of the Copyright Act 1968 (the Act).The material in this communication may be subject to copyright under the Act.Any further reproduction o
communication of this material by you may be the subject of copyright protection under the Act.
Do not remove this notice.

For a list of size N, the worst-case time complexity for insertion sort is…

A) O(log N)

B) O(N)

C) O(N log N)

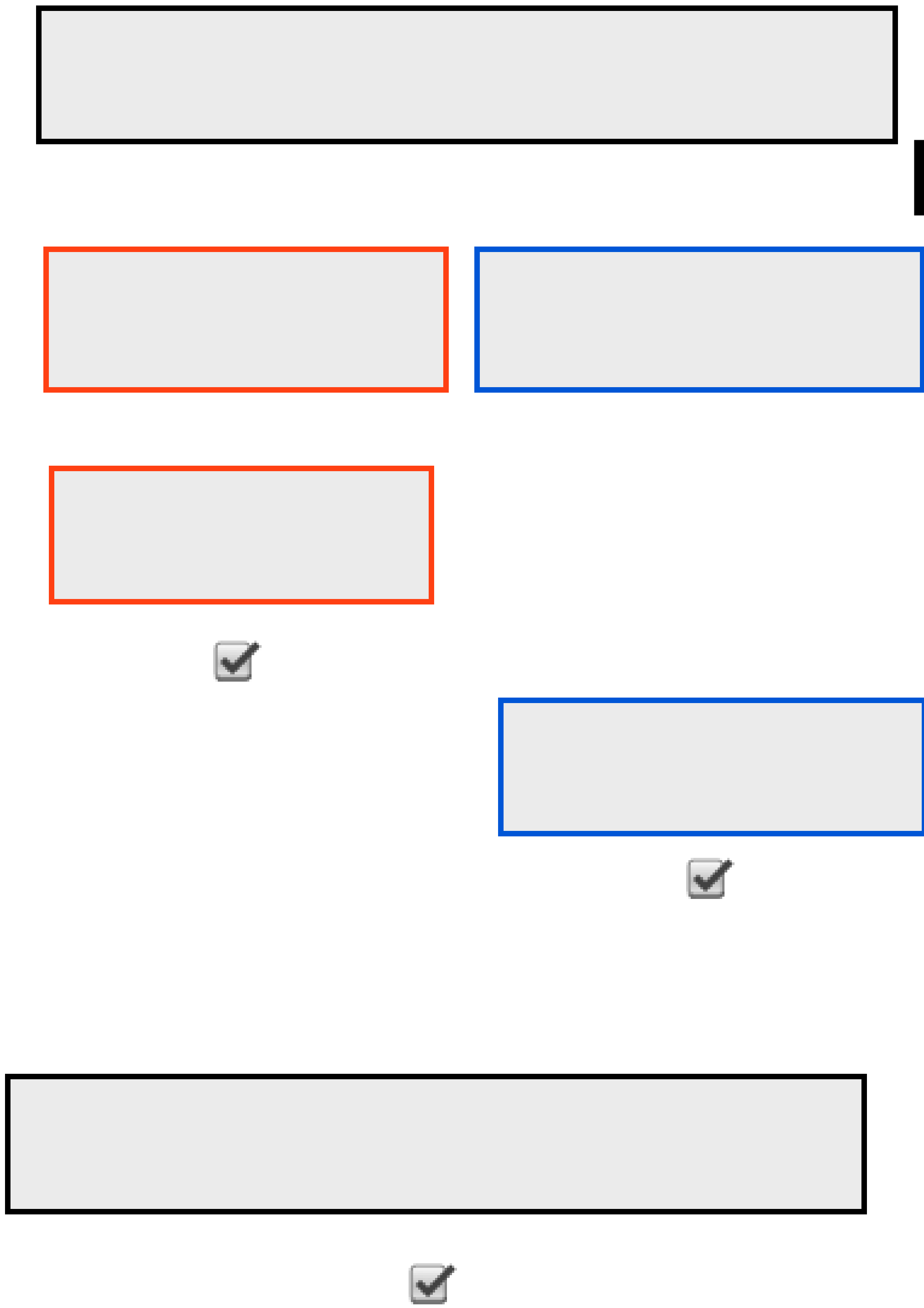
D) $O(N^2)$

Can we do better?

Overview

- To review what a "**divide and conquer**" algorithm is
- To review in more depth two different "divide and conquer" sorting algorithms:
  - **Merge Sort**
  - **Quick Sort**
- To be able to **implement** them and compare their efficiency for different classes of inputs

Divide and Conquer: **Sorting**



Divide the array into **2 parts**

Sort the first part

Sort the second part

Combine

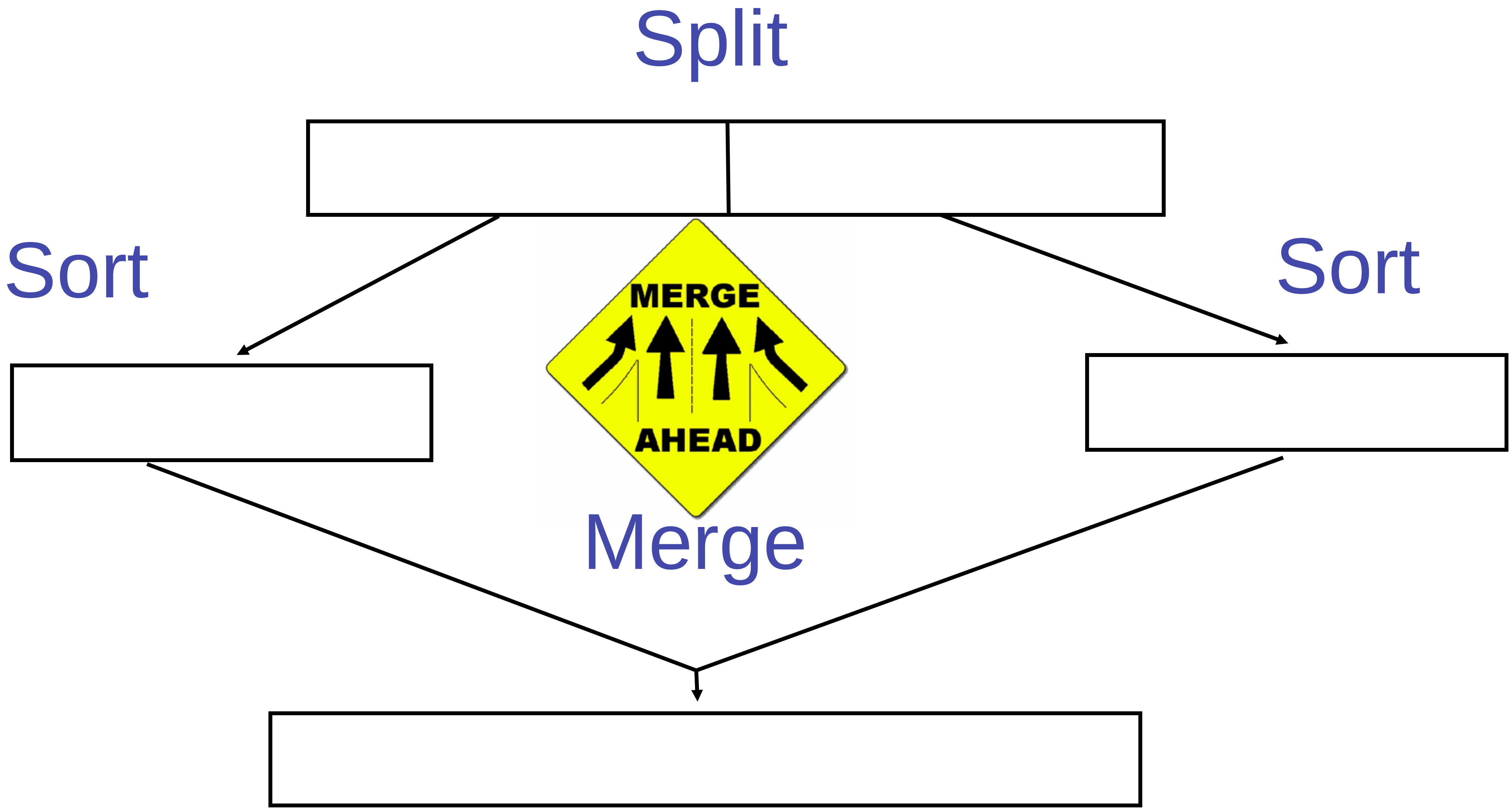
Divide and Conquer: **Sorting**

General Idea

```python
def sort(array):
    if len(array) > 1:
        split(array, first_part, second_part)
        sort(first_part)
        sort(second_part)
        combine(first_part, second_part)
```

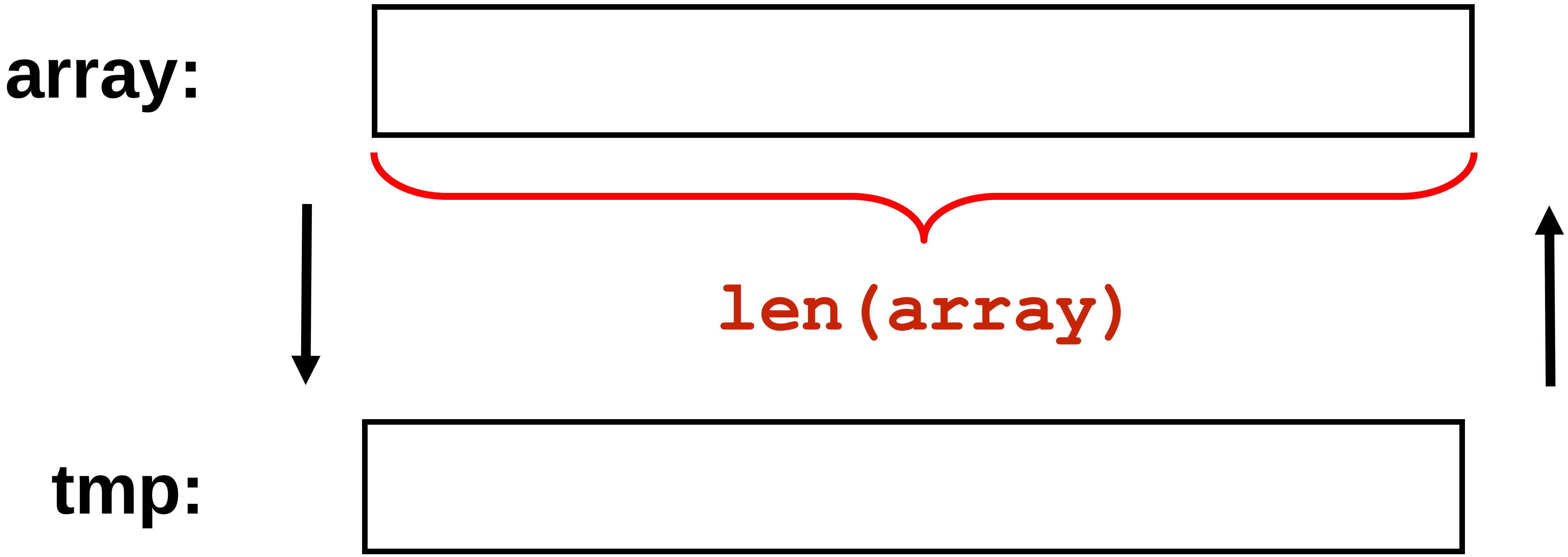
- **Merge Sort** has a simple split and a elaborate combine
- **Quick Sort** has a elaborate split and a simple combine

Merge Sort



- **Split:** In half.
- **Merge:** Take two unsorted arrays, produce a sorted one.

Merge Sort



Use a temporary array, then copy back to original array.

Merge Sort

```
def merge_sort(array):
```

Merge Sort

```python
def merge_sort(array):
    tmp = [None] * len(array)
    start = 0
    end = len(array)-1
    merge_sort_aux(array, start, end, tmp)
```

the array   start index   end index   temporary array

```python
def merge_sort_aux(array, start, end, tmp):
```

Merge Sort

```
def merge_sort_aux(array, start, end, tmp):
```

Merge Sort

```python
def merge_sort_aux(array, start, end, tmp):
    if start < end: # 2 or more still to sort
        mid = (start + end)//2

        # split into two halves
        merge_sort_aux(array, start, mid, tmp)
        merge_sort_aux(array, mid+1, end, tmp)

        # merge
        merge_arrays(array, start, mid, end, tmp)

        # copy tmp back into the original
        for i in range(start,end+1):
            array[i] = tmp[i]
```

Merge

L:

3	5	15	28	30	32

R:

10	14	22	43	50

Sorted Lists

start: 0, mid: 5, end: 10

tmp:

Merge

L:

3	5	15	28	30	32

R:

10	14	22	43	50

i=0

j=0

tmp:

3										

Merge

`L:`

3	5	15	28	30	32

`R:`

10	14	22	43	50

`i=1`

`j=0`

`tmp:`

3	5									

Merge

L:

3	5	15	28	30	32

R:

10	14	22	43	50

`i=2`

`j=0`

tmp:

3	5	10								

Merge

L:

3	5	15	28	30	32

R:

10	14	22	43	50

`i=2`

`j=1`

tmp:

3	5	10	14							

Merge

L:

3	5	15	28	30	32

R:

10	14	22	43	50

i=2

j=2

tmp:

3	5	10	14	15						

Merge

L:

3	5	15	28	30	32

R:

10	14	22	43	50

`i=3`

`j=2`

tmp:

3	5	10	14	15	22					

Merge

L:

3	5	15	28	30	32

R:

10	14	22	43	50

i=3

j=3

tmp:

3	5	10	14	15	22	28				

Merge

L:

3	5	15	28	30	32

i=4

R:

10	14	22	43	50

j=3

tmp:

3	5	10	14	15	22	28	30			

Merge

`L:`

3	5	15	28	30	32

`i=5`

`R:`

10	14	22	43	50

`j=3`

`tmp:`

3	5	10	14	15	22	28	30	32		

Merge

L:

3	5	15	28	30	32

R:

10	14	22	43	50

i=6

j=3

tmp:

3	5	10	14	15	22	28	30	32	43	

Merge

L:

3	5	15	28	30	32

R:

10	14	22	43	50

i=6

j=4

tmp:

3	5	10	14	15	22	28	30	32	43	50

Merge

L:

3	5	15	28	30	32

R:

10	14	22	43	50

`i=6`

`j=5`

tmp:

3	5	10	14	15	22	28	30	32	43	50

```python
def merge_arrays(array, start, mid, end, tmp):
```

```python
def merge_arrays(array, start, mid, end, tmp):
    i = start
    j = mid+1
    for k in range(start, end+1):
        if i > mid: # left finished, copy right
            tmp[k] = array[j]
            j += 1
        elif j > end: # right finished, copy left
            tmp[k] = array[i]
            i += 1
        elif array[i] <= array[j]: # array[i] is the item to copy
            tmp[k] = array[i]
            i += 1
        else:
            tmp[k] = array[j] # array[j] is the item to copy
            j += 1
```

Take remaining elements

Take the smaller from either list and advance

Invariant: After the kth iteration, the first k items in tmp are in order

Invariant(2): After the kth iteration tmp holds the smallest k items from array

If N= end+1-start, what is the time complexity of merge_arrays?

```python
def merge_arrays(array, start, mid, end, tmp):
    i = start
    j = mid+1
    for k in range(start, end+1):
        if i > mid: # left finished, copy right
            tmp[k] = array[j]
            j += 1
        elif j > end: # right finished, copy left
            tmp[k] = array[i]
            i += 1
        elif array[i] <= array[j]: # array[i] is the item to copy
            tmp[k] = array[i]
            i += 1
        else:
            tmp[k] = array[j] # array[j] is the item to copy
            j += 1
```

A) O(log N)

B) O(N)

C) O(N log N)

D) O(N^2)

Merge Sort Analysis

- **Natural:** Typically the method that you would use when sorting a pile of books, CDs cards, etc.
- Most of the work is in the **merging**
- Uses more **space** than other sorts
- Close to optimal in number of comparisons. Good for languages where comparison is expensive.

If N=end+1-start, what is the (worst-case) time complexity of merge sort?

```python
def merge_sort_aux(array, start, end, tmp):
    if start < end: # 2 or more still to sort
        mid = (start + end)//2

        # split into two halves
        merge_sort_aux(array, start, mid, tmp)
        merge_sort_aux(array, mid+1, end, tmp)

        # merge
        merge_arrays(array, start, mid, end, tmp)

        # copy tmp back into the original
        for i in range(start,end+1):
            array[i] = tmp[i]
```

A) O(log N)

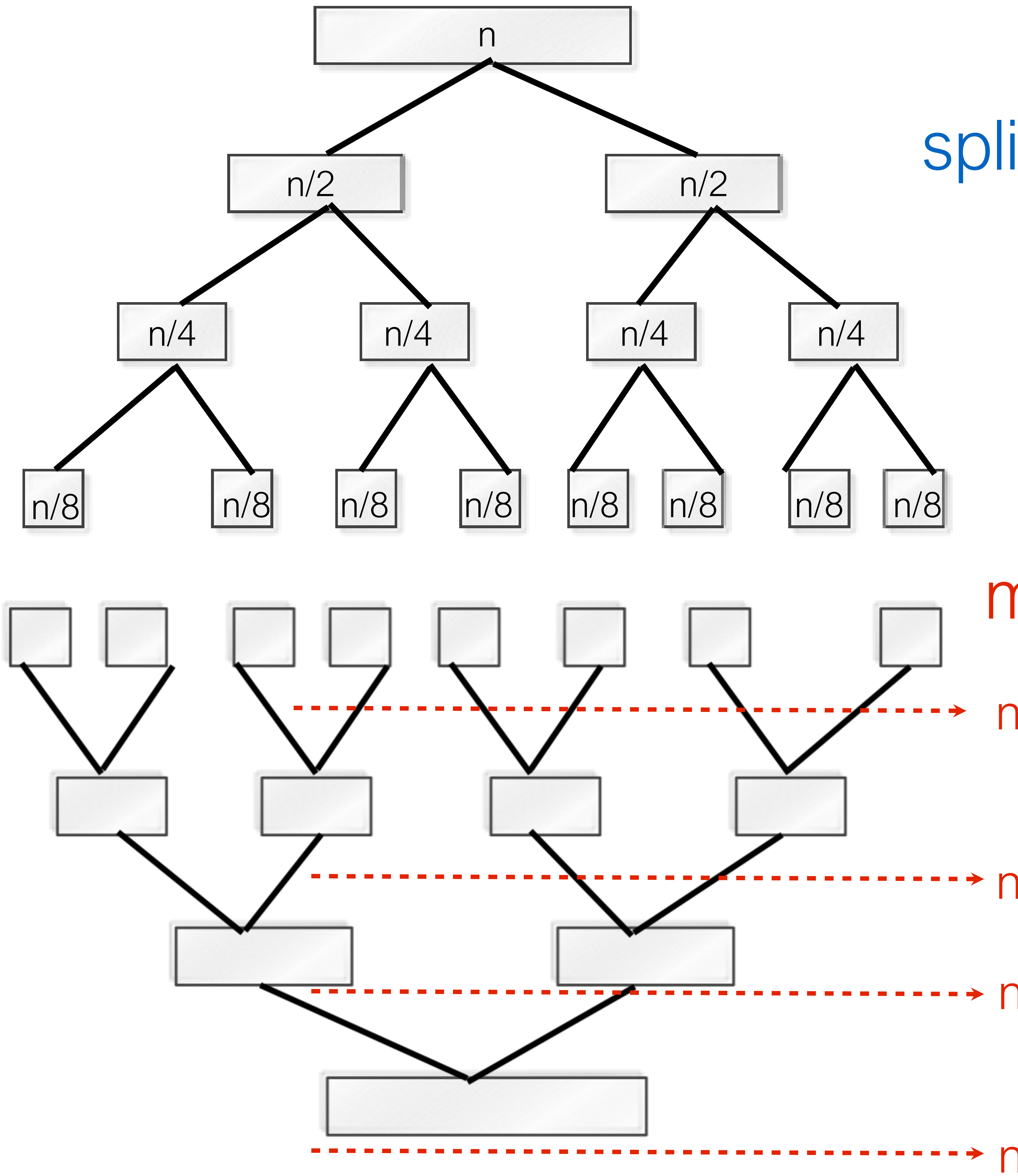
B) O(N)

C) O(N log N)

D) $O(N^2)$

Merge sort

height is O(log n)



splitting is O(1)

merging is O(n)

height is O(log n)

Total Running Time: O(n log n)

If N=end+1-start, what is the **best-case** time complexity of merge sort?

```python
def merge_sort_aux(array, start, end, tmp):
    if start < end: # 2 or more still to sort
        mid = (start + end)//2

        # split into two halves
        merge_sort_aux(array, start, mid, tmp)
        merge_sort_aux(array, mid+1, end, tmp)

        # merge
        merge_arrays(array, start, mid, end, tmp)

        # copy tmp back into the original
        for i in range(start,end+1):
            array[i] = tmp[i]
```

A) O(log N)

B) O(N)

C) O(N log N)

D) $O(N^2)$

Summary

	Best case	Worst case
Quicksort	O(?)	O(?)
Mergesort	O(n log n)	O(n log n)

Summary

Divide and Conquer and Recursive Algorithms (for sorting).

Merge Sort

- Easy: Split
- Elaborate: merge method