

Department of Computer Science and Engineering

Data Structures and Object-Oriented Design

(CSE - 2050)

Hasan Baig

Office: UConn (Stamford), 305C email: hasan.baig@uconn.edu

Module 7 Divide and Conquer

Divide and Conquer

Divide and Conquer is a paradigm for algorithm design and consists of the following three steps:

- 1. Divide: Divide the input data D into two or more disjoint subsets, D_1 and D_2 .
- 2. Conquer: Recursively solve the subproblems associated with the subsets, D_1 and D_2 .
- 3. Combine: Take the solutions to the subproblems, D_1 and D_2 , and merge them into a solution to the original problem D.

Base case: Base case for the recursion are subproblems of size 0 or 1.



It is a sorting algorithm based on the divide-and-conquer paradigm

Algorithm Development

- Divide: Partition D into two sequences D₁ and D₂, having n/2 elements in each.
 - If D has zero or one item, return
 D as it is considered sorted
- Conquer: Recursively sort D_1 and D_2 .
- Combine: Merge D_1 and D_2 into a sorted sequence.

```
Algorithm mergeSort(D)
Input sequence D with n
elements
Output sequence D sorted

if D.size() > 1
(D₁, D₂) ← partition(D, n/2)
mergeSort(D₁)
mergeSort(D₂)
D ← merge(D₁, D₂)
```



```
85 24 63 45 17 31 96 50
```

```
Algorithm mergeSort(D)
Input sequence D with n
        elements
Output sequence D sorted

if D.size() > 1
  (D<sub>1</sub>, D<sub>2</sub>) ← partition(D, n/2)
  mergeSort(D<sub>1</sub>)
  mergeSort(D<sub>2</sub>)
  D ← merge(D<sub>1</sub>, D<sub>2</sub>)
```



Divide and Conquer

Divide

```
85 24 63 45 17 31 96 50
```

```
Algorithm mergeSort(D)

Input sequence D with n
elements

Output sequence D sorted

if D.size() > 1

(D₁, D₂) ← partition(D, n/2)

mergeSort(D₁)

mergeSort(D₂)

D ← merge(D₁, D₂)
```



CSE-2050 – Data Structures and Object-Oriented Design

Divide and Conquer

Merge-Sort Algorithm

Conquer

```
    17
    31
    96
    50

    85
    24
    63
    45
```

```
Algorithm mergeSort(D)

Input sequence D with n
elements

Output sequence D sorted

if D.size() > 1
(D₁, D₂) ← partition(D, n/2)

mergeSort(D₁)

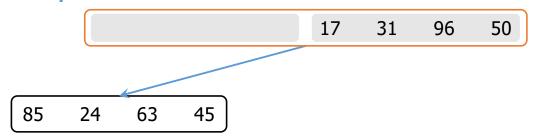
mergeSort(D₂)
D←merge(D₁, D₂)
```



CSE-2050 – Data Structures and Object-Oriented Design

Merge-Sort Algorithm

Conquer



```
Algorithm mergeSort(D)
Input sequence D with n
elements
Output sequence D sorted

if D.size() > 1
(D<sub>1</sub>, D<sub>2</sub>) ← partition(D, n/2)
mergeSort(D<sub>1</sub>)
mergeSort(D<sub>2</sub>)
D ← merge(D<sub>1</sub>, D<sub>2</sub>)
```



Divide

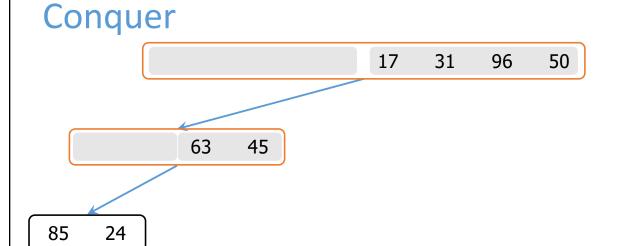
```
    17
    31
    96
    50

    85
    24
    63
    45
```

```
Algorithm mergeSort(D)
Input sequence D with n
elements
Output sequence D sorted

if D.size() > 1
(D<sub>1</sub>, D<sub>2</sub>) ← partition(D, n/2)
mergeSort(D<sub>1</sub>)
mergeSort(D<sub>2</sub>)
D ← merge(D<sub>1</sub>, D<sub>2</sub>)
```





```
Algorithm mergeSort(D)
Input sequence D with n
elements
Output sequence D sorted

if D.size() > 1
(D<sub>1</sub>, D<sub>2</sub>) ← partition(D, n/2)
mergeSort(D<sub>1</sub>)
mergeSort(D<sub>2</sub>)
D ← merge(D<sub>1</sub>, D<sub>2</sub>)
```



```
Conquer

17 31 96 50

63 45
```

```
Algorithm mergeSort(D)
Input sequence D with n
elements
Output sequence D sorted

if D.size() > 1

(D<sub>1</sub>, D<sub>2</sub>) ← partition(D, n/2)
mergeSort(D<sub>1</sub>)
mergeSort(D<sub>2</sub>)
D ← merge(D<sub>1</sub>, D<sub>2</sub>)
```



Divide 17 31 96 50 63 45

```
Algorithm mergeSort(D)
Input sequence D with n
elements
Output sequence D sorted

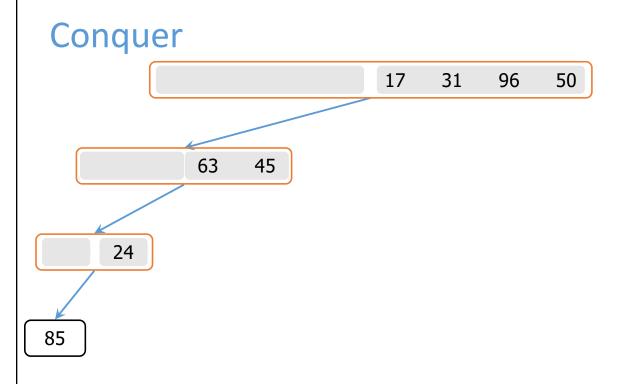
if D.size() > 1

(D<sub>1</sub>, D<sub>2</sub>) ← partition(D, n/2)
mergeSort(D<sub>1</sub>)
mergeSort(D<sub>2</sub>)
D ← merge(D<sub>1</sub>, D<sub>2</sub>)
```



85

24



```
Algorithm mergeSort(D)
Input sequence D with n
elements
Output sequence D sorted

if D.size() > 1
(D1, D2) ← partition(D, n/2)
mergeSort(D1)
mergeSort(D2)
D ← merge(D1, D2)
```



```
Base case

17 31 96 50

63 45
```

```
Algorithm mergeSort(D)
Input sequence D with n
elements
Output sequence D sorted

if D.size() > 1
(D<sub>1</sub>, D<sub>2</sub>) ← partition(D, n/2)
mergeSort(D<sub>1</sub>)
mergeSort(D<sub>2</sub>)
D ← merge(D<sub>1</sub>, D<sub>2</sub>)
```



```
Base case

17 31 96 50

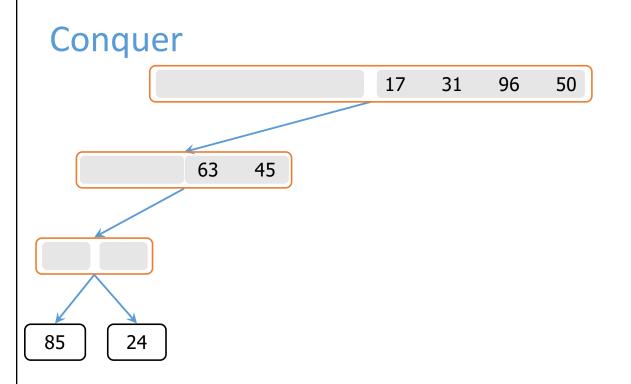
63 45

24
```

```
Algorithm mergeSort(D)
Input sequence D with n
elements
Output sequence D sorted

if D.size() > 1
(D<sub>1</sub>, D<sub>2</sub>) ← partition(D, n/2)
mergeSort(D<sub>1</sub>)
mergeSort(D<sub>2</sub>)
D ← merge(D<sub>1</sub>, D<sub>2</sub>)
```





```
Algorithm mergeSort(D)
Input sequence D with n
elements
Output sequence D sorted

if D.size() > 1
(D1, D2) ← partition(D, n/2)
mergeSort(D1)
mergeSort(D2)
D ← merge(D1, D2)
```



```
Base case
```

```
17 31 96 50
63 45
85 24
```

```
Algorithm mergeSort(D)
Input sequence D with n
elements
Output sequence D sorted

if D.size() > 1

(D<sub>1</sub>, D<sub>2</sub>) ← partition(D, n/2)
mergeSort(D<sub>1</sub>)
mergeSort(D<sub>2</sub>)
D ← merge(D<sub>1</sub>, D<sub>2</sub>)
```



Base case

```
17 31 96 50
63 45
```

```
Algorithm mergeSort(D)
Input sequence D with n
elements
Output sequence D sorted

if D.size() > 1

(D<sub>1</sub>, D<sub>2</sub>) ← partition(D, n/2)
mergeSort(D<sub>1</sub>)
mergeSort(D<sub>2</sub>)
D ← merge(D<sub>1</sub>, D<sub>2</sub>)
```



```
Merge

17 31 96 50

85 24

85 24
```

```
Algorithm mergeSort(D)
Input sequence D with n
elements
Output sequence D sorted

if D.size() > 1
(D₁, D₂) ← partition(D, n/2)
mergeSort(D₁)
mergeSort(D₂)
D ← merge(D₁, D₂)
```



```
Merge

17 31 96 50

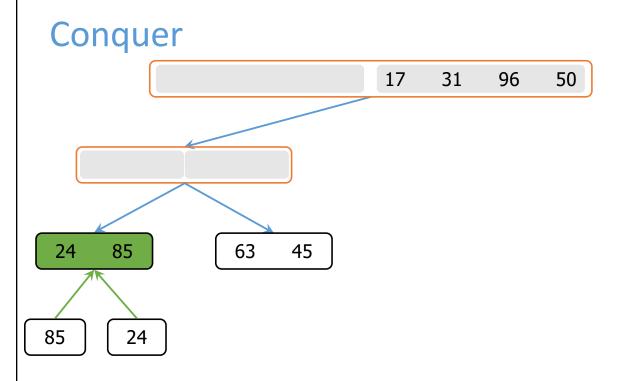
63 45

85 24
```

```
Algorithm mergeSort(D)
Input sequence D with n
elements
Output sequence D sorted

if D.size() > 1
(D1, D2) ← partition(D, n/2)
mergeSort(D1)
mergeSort(D2)
D←merge(D1, D2)
```

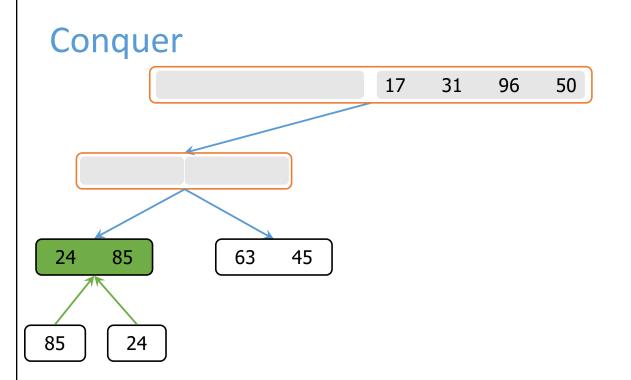




```
Algorithm mergeSort(D)
Input sequence D with n
elements
Output sequence D sorted

if D.size() > 1
(D1, D2) ← partition(D, n/2)
mergeSort(D1)
mergeSort(D2)
D←merge(D1, D2)
```

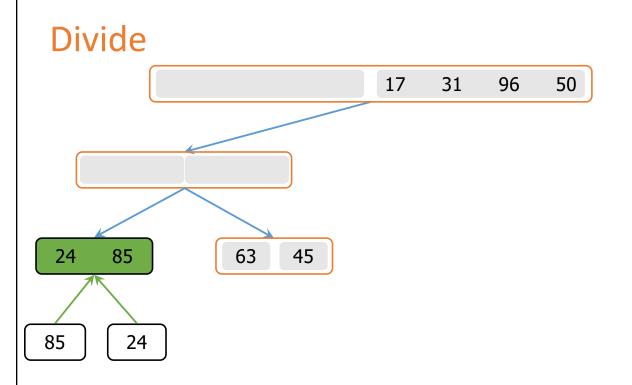




```
Algorithm mergeSort(D)
Input sequence D with n
elements
Output sequence D sorted

if D.size() > 1
(D<sub>1</sub>, D<sub>2</sub>) ← partition(D, n/2)
mergeSort(D<sub>1</sub>)
mergeSort(D<sub>2</sub>)
D ← merge(D<sub>1</sub>, D<sub>2</sub>)
```





```
Algorithm mergeSort(D)
Input sequence D with n
elements
Output sequence D sorted

if D.size() > 1

(D<sub>1</sub>, D<sub>2</sub>) ← partition(D, n/2)
mergeSort(D<sub>1</sub>)
mergeSort(D<sub>2</sub>)
D ← merge(D<sub>1</sub>, D<sub>2</sub>)
```



```
Conquer

17 31 96 50

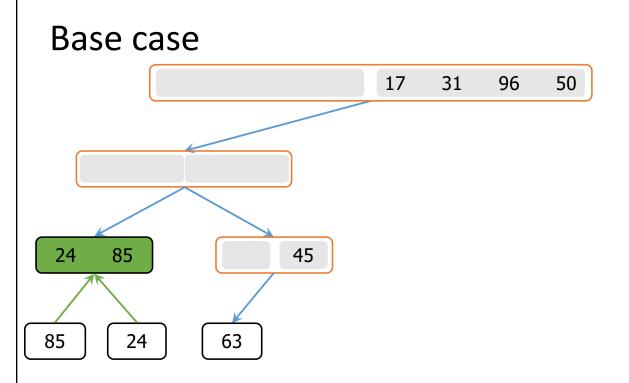
24 85 45

85 24 63
```

```
Algorithm mergeSort(D)
Input sequence D with n
elements
Output sequence D sorted

if D.size() > 1
(D1, D2) ← partition(D, n/2)
mergeSort(D1)
mergeSort(D2)
D ← merge(D1, D2)
```



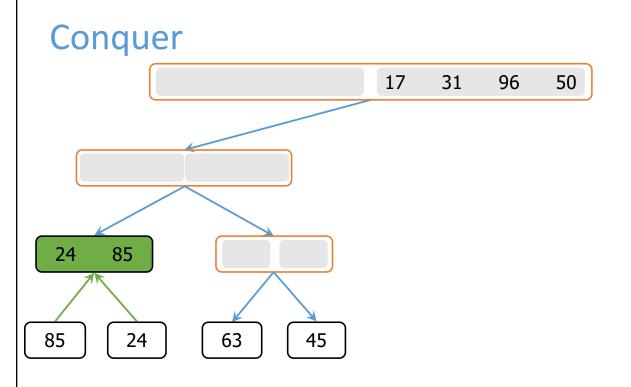


```
Algorithm mergeSort(D)
Input sequence D with n
elements
Output sequence D sorted

if D.size() > 1

(D<sub>1</sub>, D<sub>2</sub>) ← partition(D, n/2)
mergeSort(D<sub>1</sub>)
mergeSort(D<sub>2</sub>)
D ← merge(D<sub>1</sub>, D<sub>2</sub>)
```

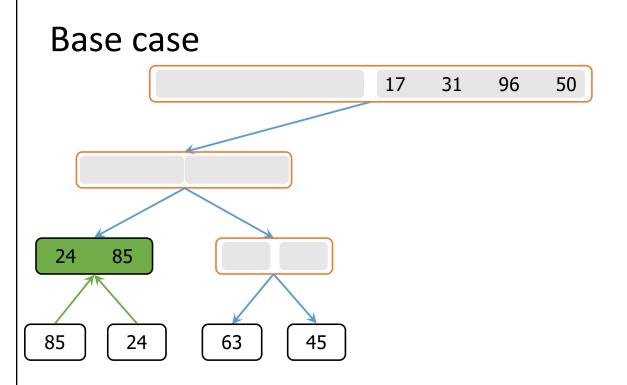




```
Algorithm mergeSort(D)
Input sequence D with n
elements
Output sequence D sorted

if D.size() > 1
(D₁, D₂) ← partition(D, n/2)
mergeSort(D₁)
mergeSort(D₂)
D ← merge(D₁, D₂)
```





```
Algorithm mergeSort(D)
Input sequence D with n
elements
Output sequence D sorted

if D.size() > 1
(D1, D2) ← partition(D, n/2)
mergeSort(D1)
mergeSort(D2)
D ← merge(D1, D2)
```



```
Merge

17 31 96 50

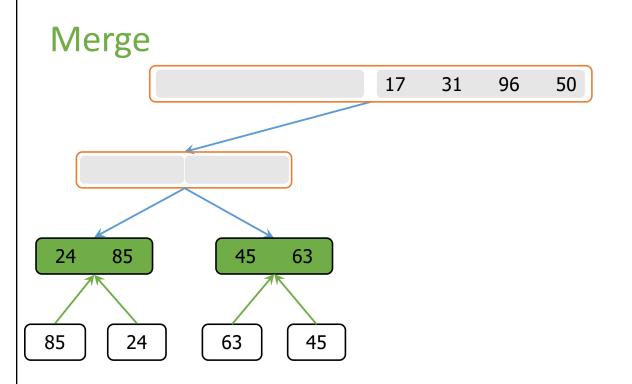
24 85 63 45

85 24 63 45
```

```
Algorithm mergeSort(D)
Input sequence D with n
elements
Output sequence D sorted

if D.size() > 1
(D1, D2) ← partition(D, n/2)
mergeSort(D1)
mergeSort(D2)
D ← merge(D1, D2)
```

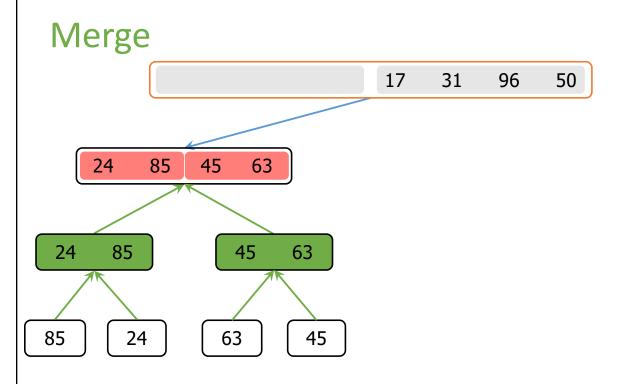




```
Algorithm mergeSort(D)
Input sequence D with n
elements
Output sequence D sorted

if D.size() > 1
(D₁, D₂) ← partition(D, n/2)
mergeSort(D₁)
mergeSort(D₂)
D ← merge(D₁, D₂)
```





```
Algorithm mergeSort(D)
Input sequence D with n
elements
Output sequence D sorted

if D.size() > 1
(D1, D2) ← partition(D, n/2)
mergeSort(D1)
mergeSort(D2)
D←merge(D1, D2)
```

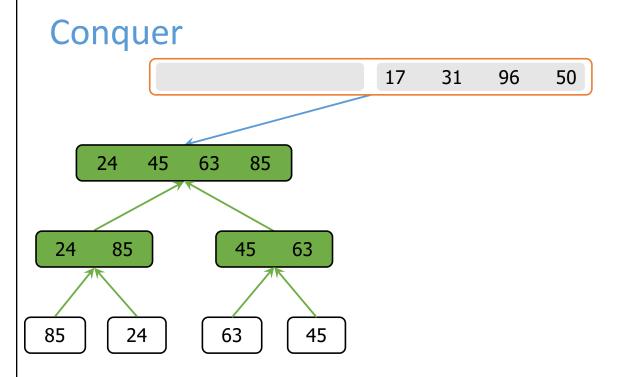


```
Merge
                                      31
                                            96
                                                  50
                                 17
          45
               63
     24
                    85
 24
      85
                  45
                        63
85
        24
                 63
                         45
```

```
Algorithm mergeSort(D)
Input sequence D with n
elements
Output sequence D sorted

if D.size() > 1
(D₁, D₂) ← partition(D, n/2)
mergeSort(D₁)
mergeSort(D₂)
D←merge(D₁, D₂)
```

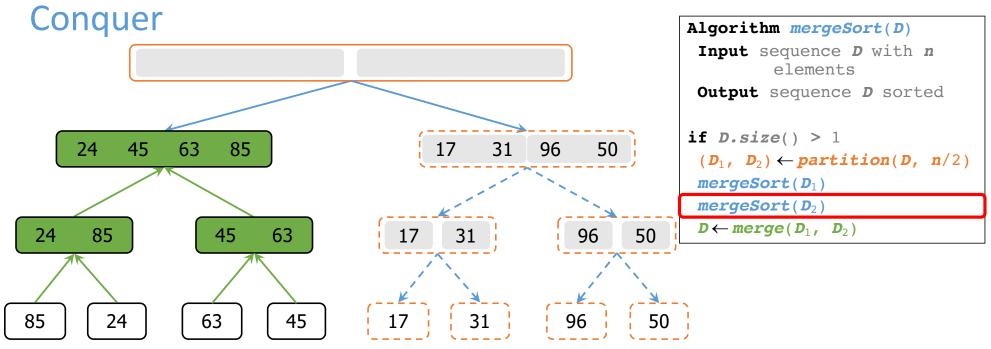




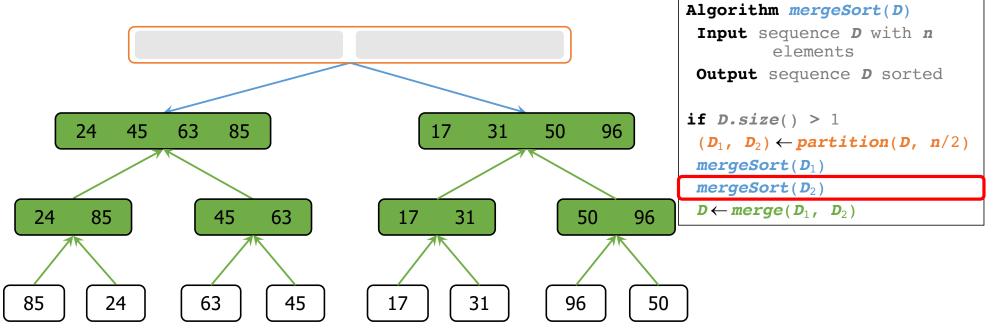
```
Algorithm mergeSort(D)
Input sequence D with n
elements
Output sequence D sorted

if D.size() > 1
(D₁, D₂) ← partition(D, n/2)
mergeSort(D₁)
mergeSort(D₂)
D ← merge(D₁, D₂)
```

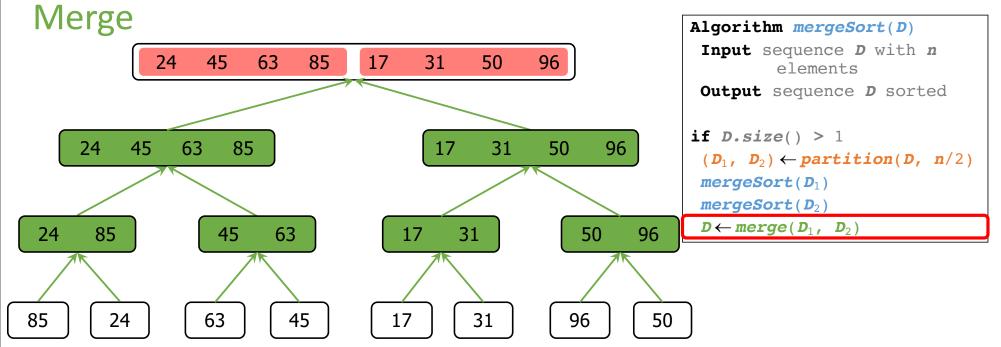




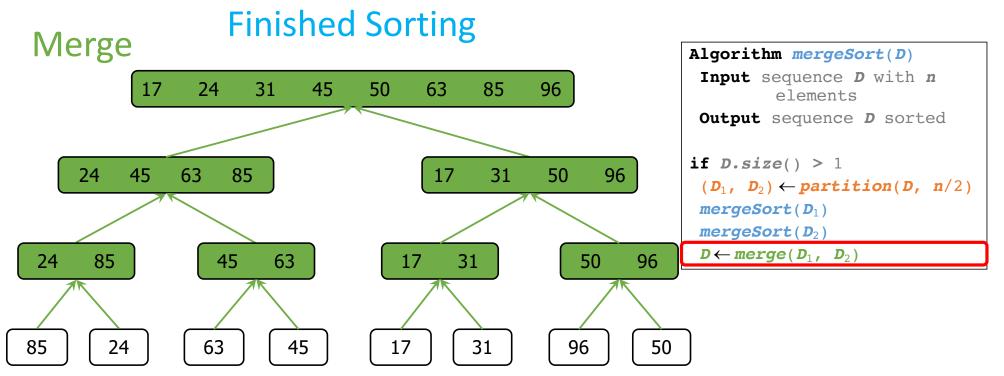














Activity

Write a python code for merge-sort algorithm

```
Algorithm mergeSort(D)
Input sequence D with n
elements
Output sequence D sorted

if D.size() > 1
(D1, D2) ← partition(D, n/2)
mergeSort(D1)
mergeSort(D2)
D←merge(D1, D2)
```



Activity Solution

• Write a python code for merge-sort algorithm

```
def mergesort(D):
    n = len(D)
    if n > 1:
        mid = n // 2
        #DIVIDING the data into half
        D1 = D[ : mid]
        D2 = D[mid : ]

#CONQUERING through recursion
        mergesort(D1)
        mergesort(D2)

#Combining/merging the data in order
        merge(D1, D2, D)
```

```
Algorithm mergeSort(D)
Input sequence D with n
elements
Output sequence D sorted

if D.size() > 1
(D1, D2) ← partition(D, n/2)
mergeSort(D1)
mergeSort(D2)
D←merge(D1, D2)
```



Divide and Conquer

Merge-Sort Algorithm

merge() Algorithm

D1 D2

 24
 45
 63
 85
 17
 31
 50
 96

j=0 j=0

D



Divide and Conquer

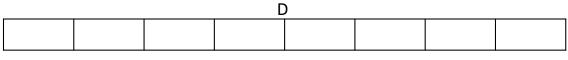
Merge-Sort Algorithm

merge() Algorithm

D1 D2

24 45 63 85 17 31 50 96

i=0 j=0



i+j=0

```
merge(D1, D2, D):
i=j=0

if D1[i] < D2[j]:
    D[i+j] = D1[i]
    i += 1
else:
    D[i+j] = D2[j]
    j += 1
```



Divide and Conquer

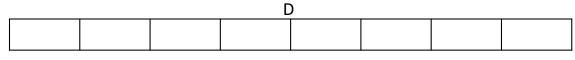
Merge-Sort Algorithm

merge() Algorithm

D1 D2

24 45 63 85 17 31 50 96

i=0 j=0



i+j=0

```
merge(D1, D2, D):
i=j=0

if D1[i] < D2[j]:
    D[i+j] = D1[i]
    i += 1
else:
    D[i+j] = D2[j]
    j += 1
```



Divide and Conquer

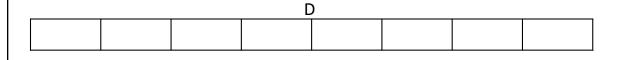
Merge-Sort Algorithm

merge() Algorithm

D1 D2

24 45 63 85 17 31 50 96

i=0 j=0



```
merge(D1, D2, D):

i=j=0

if D1[i] < D2[j]:

D[i+j] = D1[i]

i += 1

else:

D[i+j] = D2[j]

j += 1
```



Hasan Baig

i+j=0

Divide and Conquer

Merge-Sort Algorithm

merge() Algorithm

D1 D2

24 45 63 85 17 31 50 96

i=0 j=0

17 D

i+j=0

merge(D1, D2, D):
i=j=0

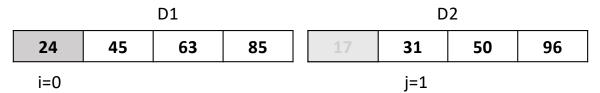
if D1[i] < D2[j]:
 D[i+j] = D1[i]
 i += 1
else:
 D[i+j] = D2[j]
 j += 1

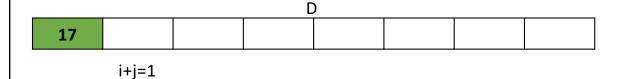


Divide and Conquer

Merge-Sort Algorithm

merge() Algorithm





```
merge(D1, D2, D):
i=j=0

if D1[i] < D2[j]:
    D[i+j] = D1[i]
    i += 1
else:
    D[i+j] = D2[j]
    j += 1
```



Divide and Conquer

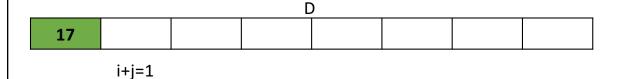
Merge-Sort Algorithm

merge() Algorithm

D1 D2

24 45 63 85 17 31 50 96

i=0 j=1



```
merge(D1, D2, D):
i=j=0

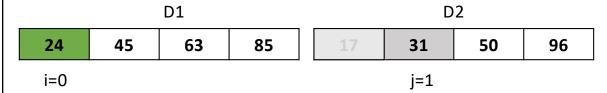
if D1[i] < D2[j]:
    D[i+j] = D1[i]
    i += 1
else:
    D[i+j] = D2[j]
    j += 1
```

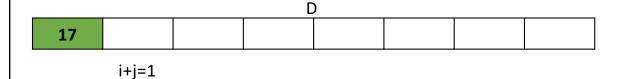


Divide and Conquer

Merge-Sort Algorithm

merge() Algorithm





```
merge(D1, D2, D):
i=j=0

if D1[i] < D2[j]:
    D[i+j] = D1[i]
    i += 1
else:
    D[i+j] = D2[j]
    j += 1
```



Divide and Conquer

Merge-Sort Algorithm

merge() Algorithm

D1 D2

24 45 63 85 17 31 50 96

i=0 j=1

D

17 24 | i+j=1

```
merge(D1, D2, D):
i=j=0

if D1[i] < D2[j]:
    D[i+j] = D1[i]
    i += 1
else:
    D[i+j] = D2[j]
    j += 1
```



Divide and Conquer

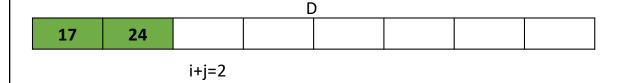
Merge-Sort Algorithm

merge() Algorithm

D1 D2

24 45 63 85 17 31 50 96

i=1 j=1



```
merge(D1, D2, D):
i=j=0

if D1[i] < D2[j]:
    D[i+j] = D1[i]
    i += 1
else:
    D[i+j] = D2[j]
    j += 1
```

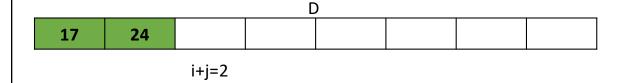


Divide and Conquer

Merge-Sort Algorithm

merge() Algorithm





```
merge(D1, D2, D):
i=j=0

if D1[i] < D2[j]:
    D[i+j] = D1[i]
    i += 1
else:
    D[i+j] = D2[j]
    j += 1
```

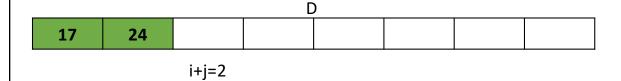


Divide and Conquer

Merge-Sort Algorithm

merge() Algorithm





```
merge(D1, D2, D):
i=j=0

if D1[i] < D2[j]:
    D[i+j] = D1[i]
    i += 1
else:
    D[i+j] = D2[j]
    j += 1
```

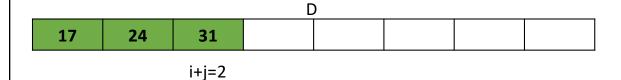


Divide and Conquer

Merge-Sort Algorithm

merge() Algorithm





```
merge(D1, D2, D):
i=j=0

if D1[i] < D2[j]:
    D[i+j] = D1[i]
    i += 1
else:
    D[i+j] = D2[j]
    j += 1
```



Divide and Conquer

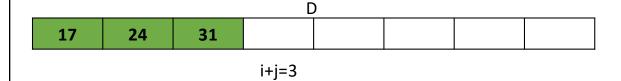
Merge-Sort Algorithm

merge() Algorithm

D1 D2

24 45 63 85 17 31 50 96

i=1 j=2



```
merge(D1, D2, D):
i=j=0

if D1[i] < D2[j]:
    D[i+j] = D1[i]
    i += 1
else:
    D[i+j] = D2[j]
    j += 1
```



Divide and Conquer

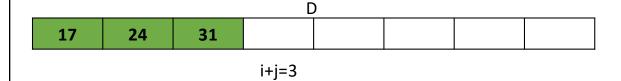
Merge-Sort Algorithm

merge() Algorithm

D1 D2

24 45 63 85 17 31 50 96

i=1 j=2



```
merge(D1, D2, D):
i=j=0

if D1[i] < D2[j]:
    D[i+j] = D1[i]
    i += 1
else:
    D[i+j] = D2[j]
    j += 1
```

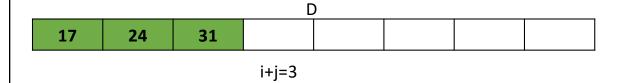


Divide and Conquer

Merge-Sort Algorithm

merge() Algorithm





```
merge(D1, D2, D):
i=j=0

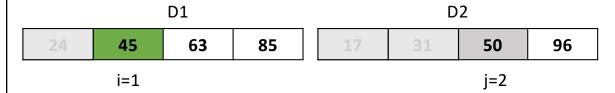
if D1[i] < D2[j]:
    D[i+j] = D1[i]
    i += 1
else:
    D[i+j] = D2[j]
    j += 1
```



Divide and Conquer

Merge-Sort Algorithm

merge() Algorithm





```
merge(D1, D2, D):
i=j=0

if D1[i] < D2[j]:
    D[i+j] = D1[i]
    i += 1
else:
    D[i+j] = D2[j]
    j += 1
```



Divide and Conquer

Merge-Sort Algorithm

merge() Algorithm

D1 D2

24 45 63 85 17 31 50 96

i=2 j=2



```
merge(D1, D2, D):
i=j=0

if D1[i] < D2[j]:
    D[i+j] = D1[i]
    i += 1
else:
    D[i+j] = D2[j]
    j += 1
```



Divide and Conquer

Merge-Sort Algorithm

merge() Algorithm

D1 D2

24 45 63 85 17 31 50 96

i=2 j=2



```
merge(D1, D2, D):
i=j=0

if D1[i] < D2[j]:
    D[i+j] = D1[i]
    i += 1
else:
    D[i+j] = D2[j]
    j += 1
```



Divide and Conquer

Merge-Sort Algorithm

merge() Algorithm





```
merge(D1, D2, D):
i=j=0

if D1[i] < D2[j]:
    D[i+j] = D1[i]
    i += 1
else:
    D[i+j] = D2[j]
    j += 1
```

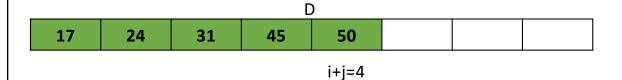


Divide and Conquer

Merge-Sort Algorithm

merge() Algorithm





```
merge(D1, D2, D):
i=j=0

if D1[i] < D2[j]:
    D[i+j] = D1[i]
    i += 1
else:
    D[i+j] = D2[j]
    j += 1
```



Divide and Conquer

Merge-Sort Algorithm

merge() Algorithm

D1 D2

24 45 63 85 17 31 50 96

i=2 j=3

D
17 24 31 45 50 i+j=5

```
D[i+j] = D1[i]

i += 1

else:

D[i+j] = D2[j]

j += 1
```

merge(D1, D2, D):

if D1[i] < D2[j]:

i=j=0

60



Divide and Conquer

Merge-Sort Algorithm

merge() Algorithm

D1 D2

24 45 63 85 17 31 50 96

i=2 j=3

17 24 31 45 50

i+j=5

merge(D1, D2, D): i=j=0 if D1[i] < D2[j]: D[i+j] = D1[i] i += 1 else: D[i+j] = D2[j] j += 1

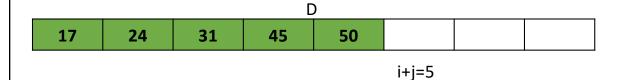


Divide and Conquer

Merge-Sort Algorithm

merge() Algorithm





```
merge(D1, D2, D):
i=j=0

if D1[i] < D2[j]:
    D[i+j] = D1[i]
    i += 1
else:
    D[i+j] = D2[j]
    j += 1
```

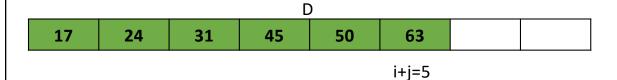


Divide and Conquer

Merge-Sort Algorithm

merge() Algorithm





```
merge(D1, D2, D):
i=j=0

if D1[i] < D2[j]:
    D[i+j] = D1[i]
    i += 1
else:
    D[i+j] = D2[j]
    j += 1
```



Divide and Conquer

Merge-Sort Algorithm

merge() Algorithm

D1 D2

24 45 63 85 17 31 50 96

i=3 j=3

17 24 31 45 50 63

i+j=6

merge(D1, D2, D):
i=j=0

if D1[i] < D2[j]:
 D[i+j] = D1[i]
 i += 1
else:
 D[i+j] = D2[j]
 j += 1



Divide and Conquer

Merge-Sort Algorithm

merge() Algorithm

D1 D2

24 45 63 85 17 31 50 96

i=3 j=3

17 24 31 45 50 63

i+j=6

merge(D1, D2, D):
i=j=0

if D1[i] < D2[j]:
 D[i+j] = D1[i]
 i += 1
else:
 D[i+j] = D2[j]
 j += 1

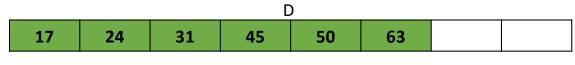


Divide and Conquer

Merge-Sort Algorithm

merge() Algorithm





i+j=6

```
merge(D1, D2, D):
i=j=0

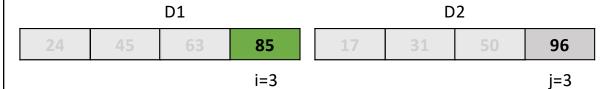
if D1[i] < D2[j]:
    D[i+j] = D1[i]
    i += 1
else:
    D[i+j] = D2[j]
    j += 1
```



Divide and Conquer

Merge-Sort Algorithm

merge() Algorithm



17 24 31 45 50 63 85

i+j=6

merge(D1, D2, D):
i=j=0

if D1[i] < D2[j]:
D[i+j] = D1[i]
i += 1
else:
D[i+j] = D2[j]
j += 1



Divide and Conquer

Merge-Sort Algorithm

merge() Algorithm





i+j=7

```
merge(D1, D2, D):
```

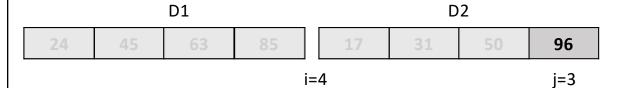
i=j=0



Divide and Conquer

Merge-Sort Algorithm

merge() Algorithm



D 24 31 45 50 63 85

i+j=7

```
merge(D1, D2, D):
    i=j=0

while i < len(D1) and while j < len(D1)
    if D1[i] < D2[j]:
        D[i+j] = D1[i]
        i += 1
    else:
        D[i+j] = D2[j]
        j += 1
```



Hasan Baig

17

Divide and Conquer

Merge-Sort Algorithm

merge() Algorithm

		DI		D2				
24	45	63	85	17	31	50	96	
			i	i=4			j=3	

j=3

D											
17	24	31	45	50	63	85	96				

i+j=7

```
merge(D1, D2, D):
   i=j=0
   while i < len(D1) and while j < len(D1)
       if D1[i] < D2[j]:
            D[i+j] = D1[i]
            i += 1
        else:
            D[i+j] = D2[j]
           i += 1
   D[i+j:] = D1[i:] + D2[j:]
```

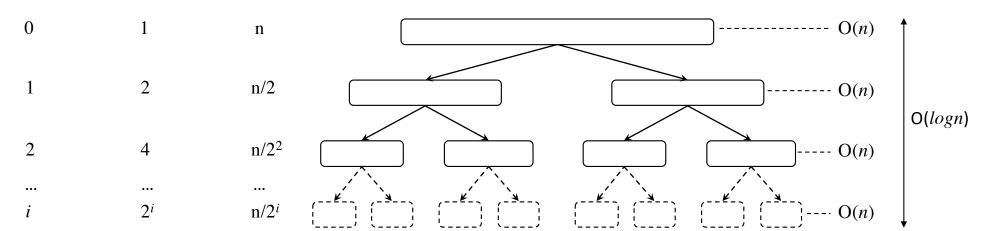


71

Merge-Sort Algorithm

Time Complexity

Depth | # nodes/sequences | size



- The amount of work done at each node is merge + partition
 - Total work done at depth *i* is: number of nodes x size of nodes = $2^i \times n/2^i \rightarrow O(n)$
- What is the stopping condition of recursion? \rightarrow O(logn)
 - \rightarrow Total time complexity = $O(n \log n)$

