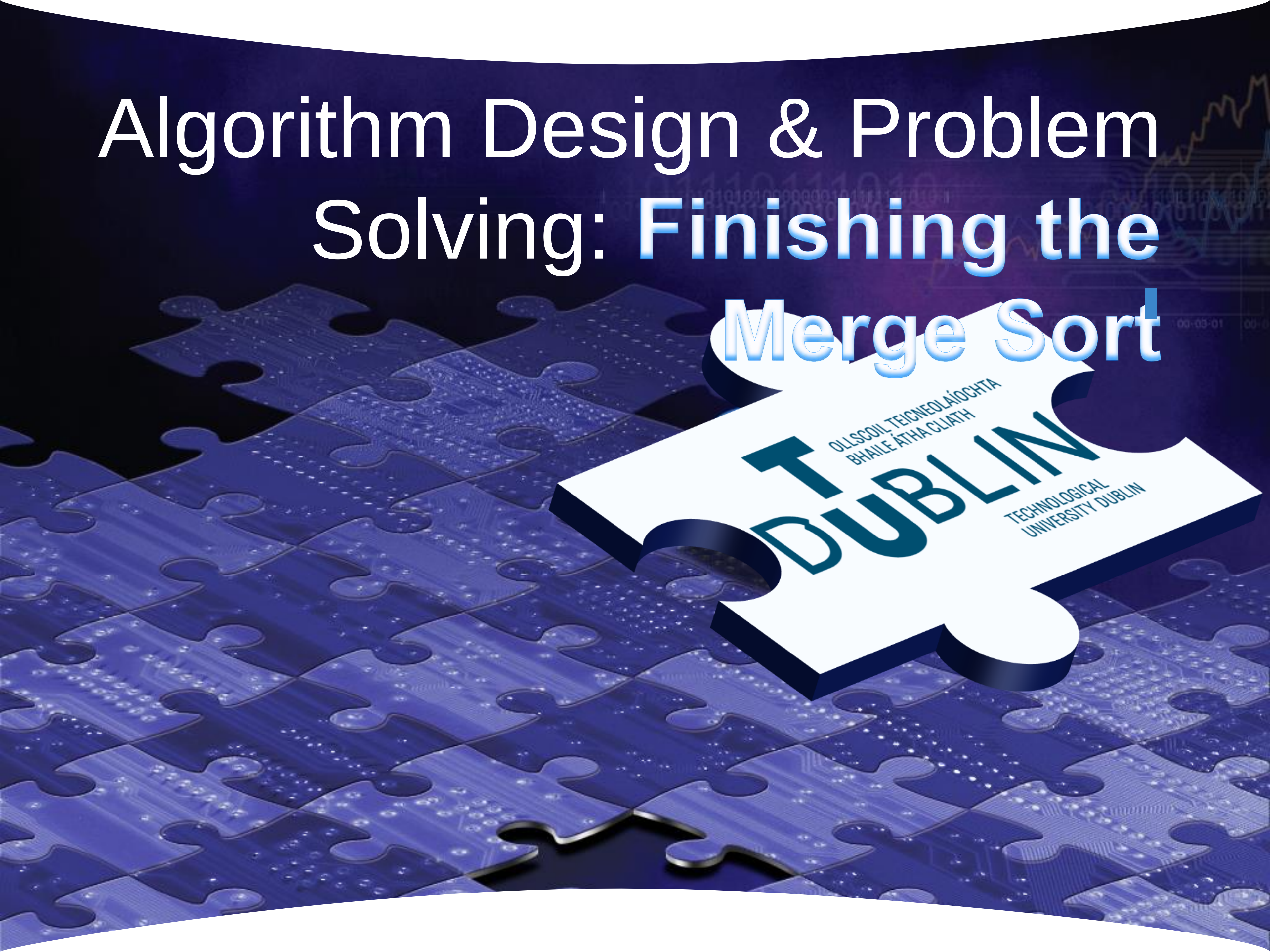
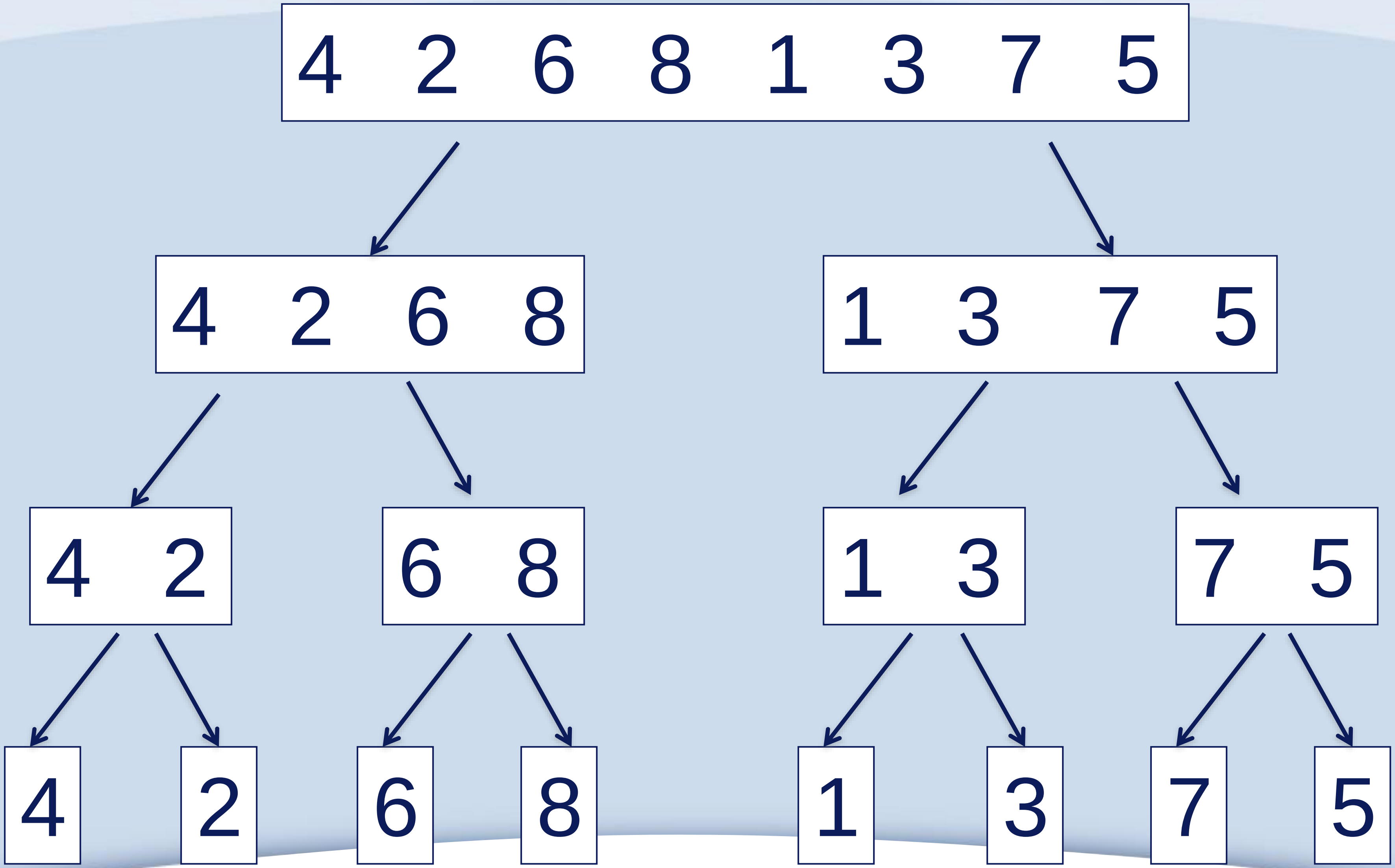
Algorithm Design & Problem Solving: **Finishing the Merge Sort**

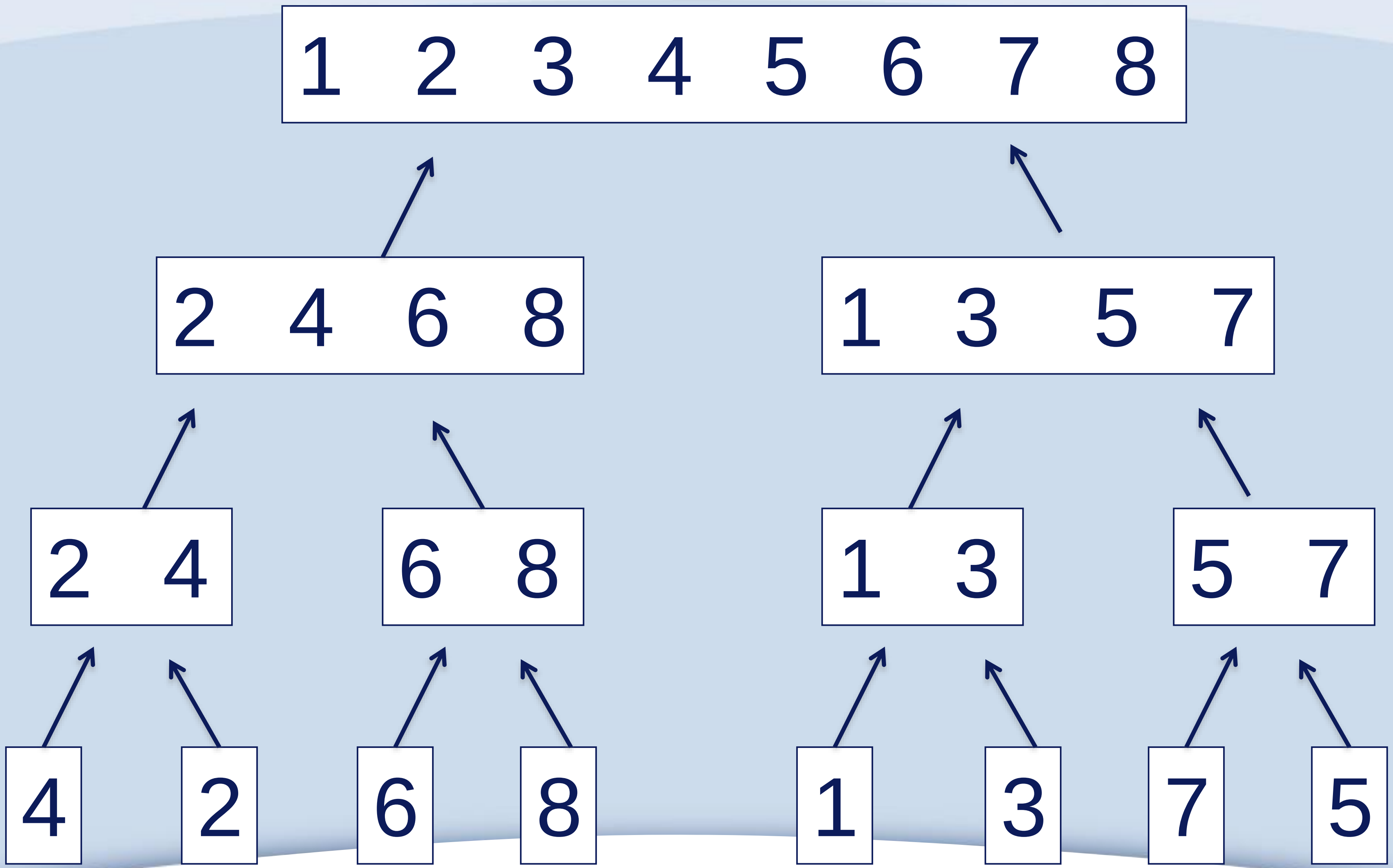

OLLSCOIL TEICNEOLAÍOCHTA BHAILE ÁTHA CLIATH
TU DUBLIN
TECHNOLOGICAL UNIVERSITY DUBLIN

Contents

Merge Sort (Divide & Conquer)



Merge Sort (Combine)



Merge Sort

- Algorithm “divides” list into 2,
- “conquers” - breaks down list into smallest possible elements – pairs of single numbers
- Then builds it back up in stack order combining each pair of numbers into the sorted list
- How many divisions to get down to a single number?
  - Starting with 8 = 4->2->1 =3
  - Starting with 16 = 8->4->2->1 = 4
  - Pattern?
  - Log_2N

Reminder: Merge Sort

1. **Divide**
2. Conquer
3. Combine

Finds middle of array

Merge Sort

1. Divide
2. **Conquer**
3. Combine

Recursively solve two sub problems

Merge Sort

1. Divide
2. Conquer
3. **Combine**

Merge to recreate array

Merge Sort

```
On input of n elements:
If n < 2
        Return
Else
        Sort left half of elements.
        Sort right half of elements.
        Merge sorted halves.
```

Algorithm Merge_Sort (A, low, high):

```
If n < 2
        Return
Else
        mid = (low+high)/2                    Divide
        Merge_Sort(A, low, mid)
        Merge_Sort(A, mid+1, high)            Conquer
        Merge(A, low, mid, high)              Combine
```

```
1. Merge_Sort (A, low, high):
2. If $n < 2$
3.     Return
4. Else
5.     mid = (low+high)/2                      Divide
6.     Merge_Sort(A, low, mid)
7.     Merge_Sort(A, mid+1, high)              Conquer
8.     Merge(A, low, mid, high)                Combine
```

Merge Sort

```
1. Merge_Sort (A, low, high):
2. If n < 2
3.   Return
4. Else
5.   mid = (low+high)/2
6.   Merge_Sort(A, low, mid)
7.   Merge_Sort(A, mid+1, high)
8.   Merge(A, low, mid, high)
```

- Q: So how does this work?
- A: do the stack trace
- Show how the merge sort works to sort

4	3	1	6

Line Number	Function Calls	Left or right	mid	low	high
main () 0	MergeSort(A, 0, 3)			0	3

Take 10 Minutes and see can you work out what happens next. Look at the code and plug in your arguments.

Merge Sort

4	3	1	6

```
1. Merge_Sort (A, low, high):
2. If n < 2
3.   Return
4. Else
5.   mid = (low+high)/2
6.   Merge_Sort(A, low, mid)
7.   Merge_Sort(A, mid+1, high)
8.   Merge(A, low, mid, high)
```

- Base case fail
- Mid=(0+3)/2=1
- Call left mergesort line 6

Line Number	Function Calls	Left or right	mid	low	high
main () 0	MergeSort(A, 0, 3)			0	3
6	MergeSort(A,0,1)	L	1	0	3

Merge Sort

4	3	1	6

```
1. Merge_Sort (A, low, high):
2. If n < 2
3.   Return
4. Else
5.   mid = (low+high)/2
6.   Merge_Sort(A, low, mid)
7.   Merge_Sort(A, mid+1, high)
8.   Merge(A, low, mid, high)
```

- Base case fail
- Mid=(0+1)/2=0
- Call left mergesort line 6

Line Number	Function Calls	Left or right	mid	low	high
main () 0	MergeSort(A, 0, 3)			0	3
6	MergeSort(A,0,1)	L	1	0	3
6	MergeSort(A,0,0)	L	0	0	1

Merge Sort

4 3 1 6

```
1. Merge_Sort (A, low, high):
2. If n < 2
3.   Return
4. Else
5.   mid = (low+high)/2
6.   Merge_Sort(A, low, mid)
7.   Merge_Sort(A, mid+1, high)
8.   Merge(A, low, mid, high)
```

- Base case true –pop
- Execution resumes on line 7 - R

Line Number	Function Calls	Left or right	mid	low	high
main () 0	MergeSort(A, 0, 3)			0	3
6	MergeSort(A,0,1)	L	1	0	3
6	MergeSort(A,0,0)	L	0	0	1
6	pop				
7	MergeSort(A,1,1)	R	0	0	1

Merge Sort

4	3	1	6

```
1. Merge_Sort (A, low, high):
2. If n < 2
3.   Return
4. Else
5.   mid = (low+high)/2
6.   Merge_Sort(A, low, mid)
7.   Merge_Sort(A, mid+1, high)
8.   Merge(A, low, mid, high)
```

- Base case true –pop
- Execution resumes on line 8 merge
- What does merge do – put A[0] and A[1] in order

Line Number	Function Calls	Left or right	mid	low	high
main () 0	MergeSort(A, 0, 3)			0	3
6	MergeSort(A,0,1)	L	1	0	3
6	MergeSort(A,0,0)	L	0	0	1
6	pop				
7	MergeSort(A,1,1)	R	0	0	1
	Pop				
8	Merge (A,0,0,1)		0	0	1

Merge Sort

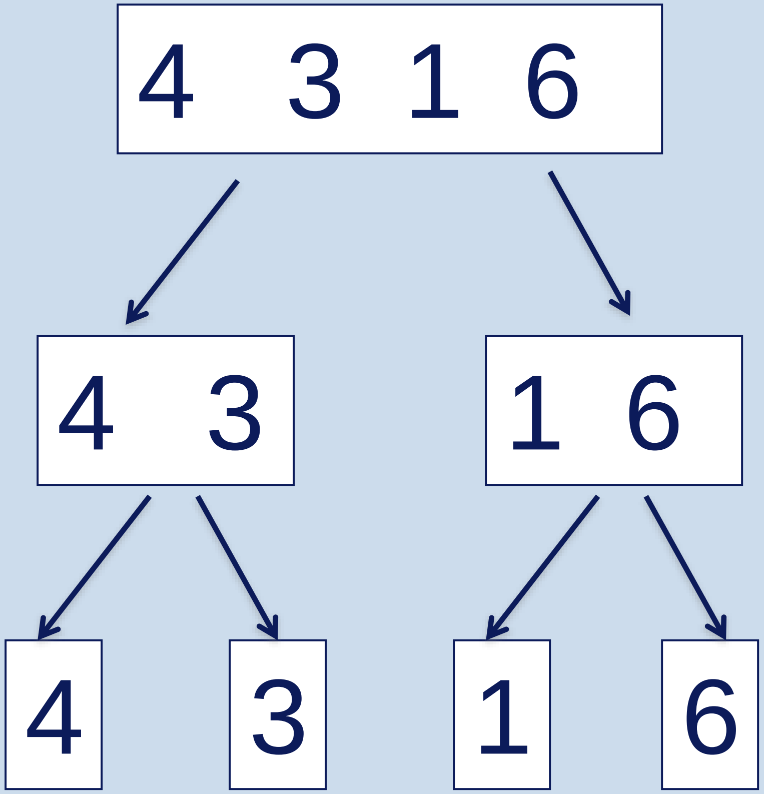
4	3	1	6

```
1. Merge_Sort (A, low, high):
2. If n < 2
3.   Return
4. Else
5.   mid = (low+high)/2
6.   Merge_Sort(A, low, mid)
7.   Merge_Sort(A, mid+1, high)
8.   Merge(A, low, mid, high)
```

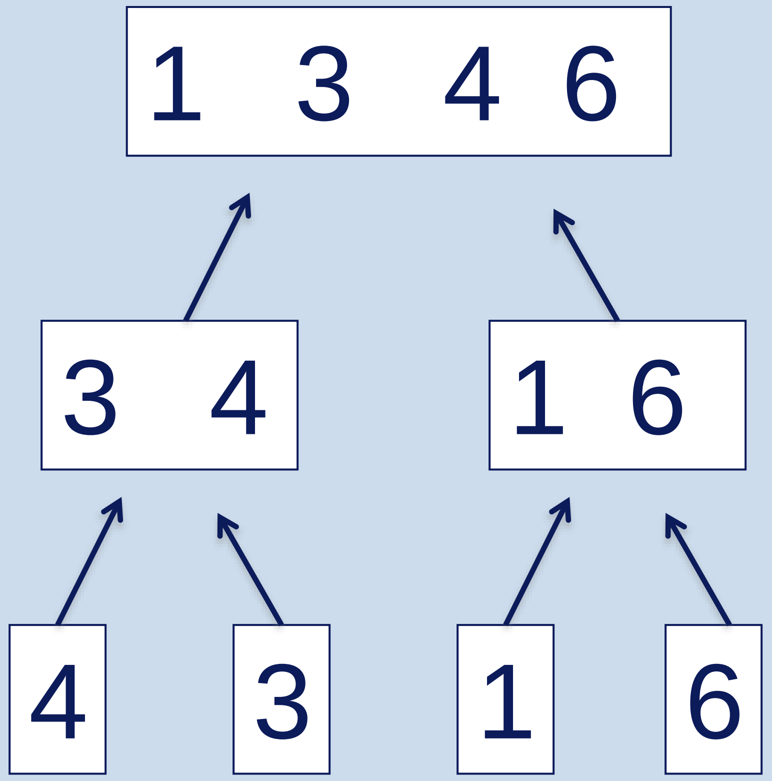
- Mergesort function exits, resumes to do right merge sort from top
- Right Side (16) completes as the first 43
- Then followed by final merge

Line Number	Function Calls	Left or right	mid	low	high
main () 0	MergeSort(A, 0, 3)			0	3
6	MergeSort(A,0,1)	L	1	0	3
6	MergeSort(A,0,0)	L	0	0	1
7	MergeSort(A,1,1)	R	0	0	1
8	Merge (A,0,0,1)		0	0	1
7	MergeSort(A,2,3)	R	1	0	3
6	MergeSort(A,2,2)	L	2	2	3
7	MergeSort(A,3,3)	R	2	2	3
8	Merge (A,2,2,3)				
8	Merge (A,0,1,3)				

Merge Sort (Divide & Conquer)



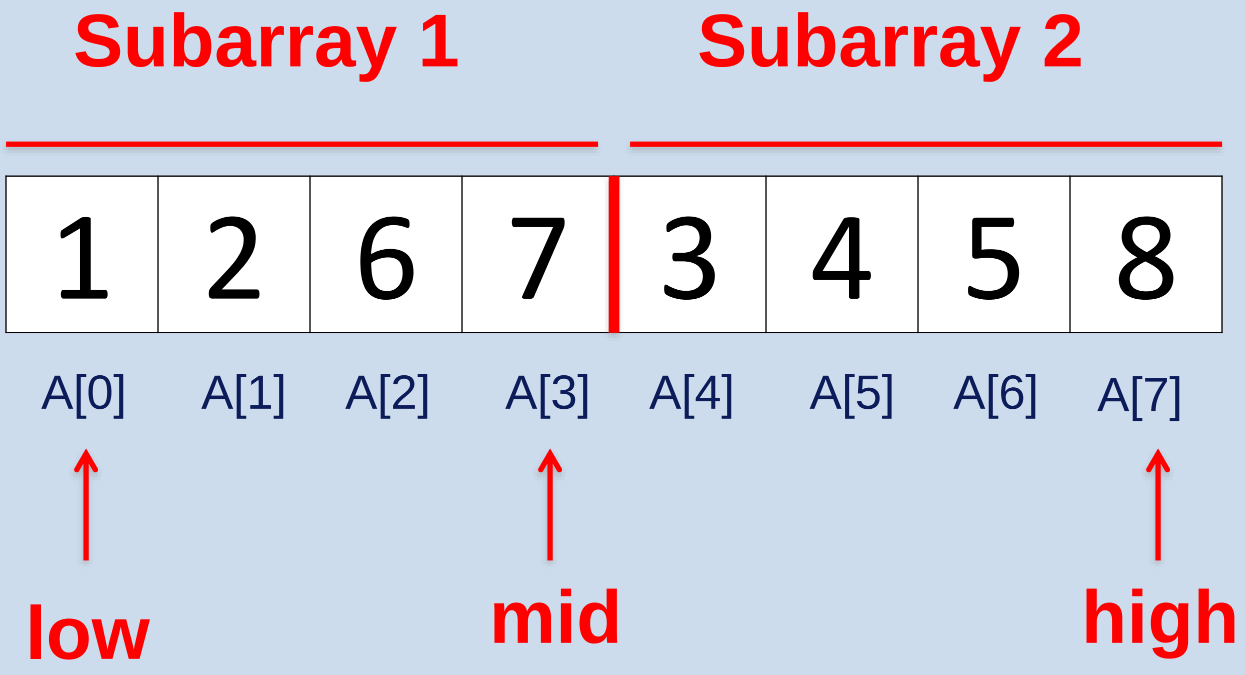
Merge Sort (Combine)



Part II: the Merge

- The merge sort is really two algorithms combined.
- First one divides up the list
- Second does the merge, in order – hence merge sort.

Combining/Merging: the Merge



Part II: the Merge

Merge the subarrays until the end of one is reached.

1. Compare first element of both subarrays. Place the smallest of the 2 first elements into a temp array and move to next space in subarray

Array A	1	2	6	7	3	4	5	8
Array tempArray	1							
Array A	1	2	6	7	3	4	5	8

Part II: the Merge

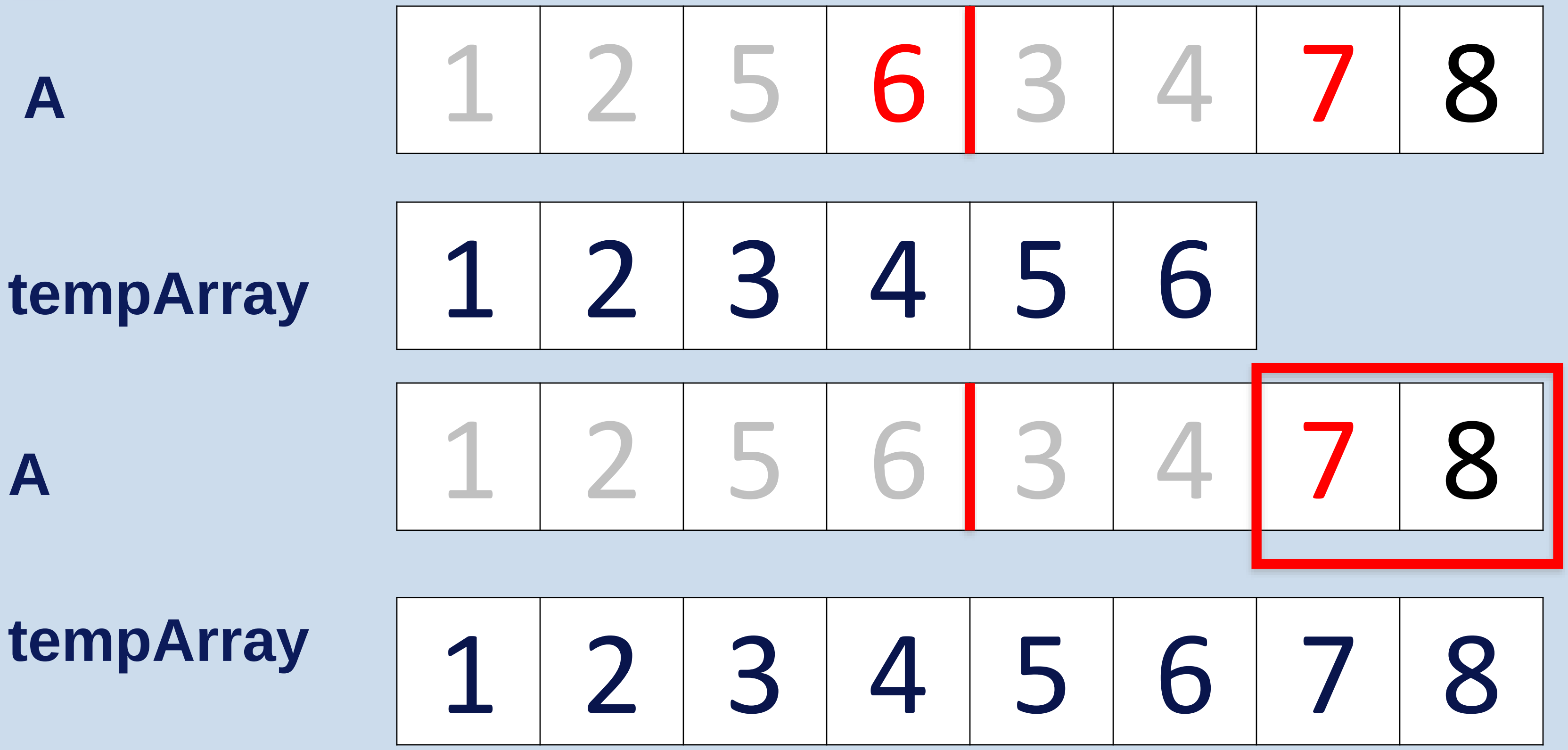
A	1	2	5	6	3	4	7	8
tempArray	1							
A	1	2	5	6	3	4	7	8
tempArray	1	2						
A	1	2	5	6	3	4	7	8
tempArray	1	2	3					

Part II: the Merge

1. Merge the subarrays until the end of one is reached.
    1. Compare first element of both subarrays. Place the smallest of the 2 first elements into a temp array and move to next space in subarray
2. If end of left subarray reached
    - Copy the right subarray into tempArray
    - Otherwise, copy the left subarray into tempArray

1	2	5	6	3	4	7	8

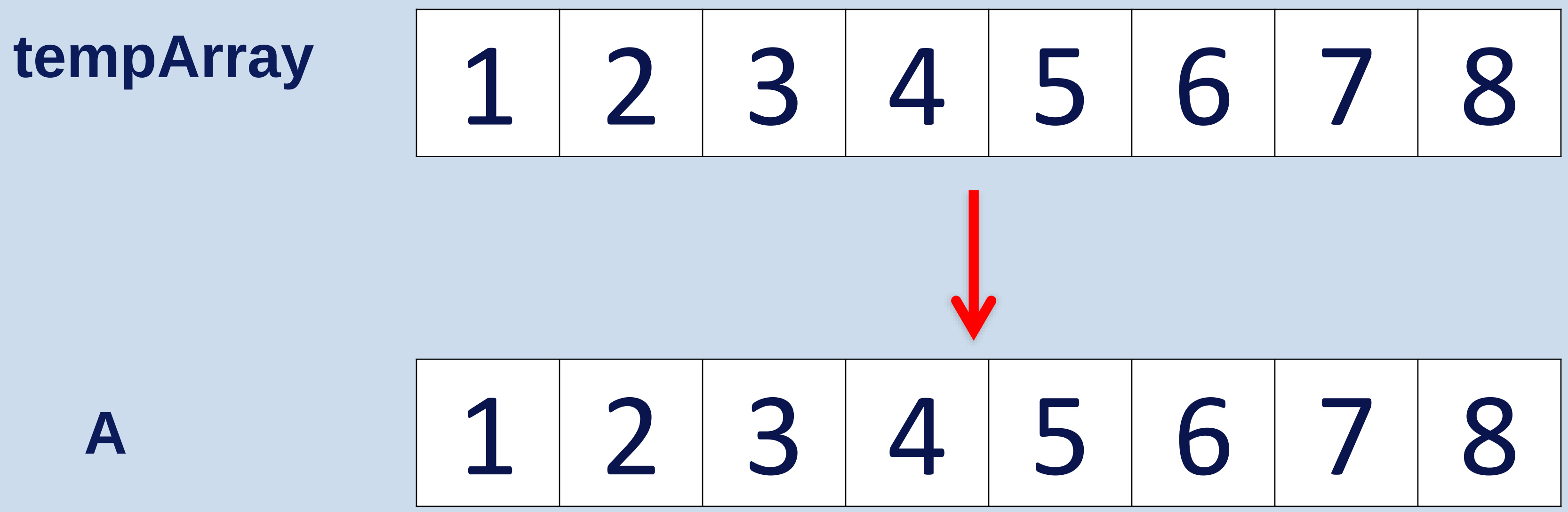
Part II: the Merge



Part II: the Merge

1. Merge the subarrays until the end of one is reached.
    1. Compare first element of both subarrays. Place the smallest of the 2 first elements into a temp array and move to next space in subarray
2. If end of left subarray reached
    - Copy the left subarray
    - Otherwise, copy the right subarray
3. Copy values back into original array

Part II: the Merge



Part II: the Merge

```
Merge(A, low, mid, high)
   left=low
   right=mid+1
   temp=left
   while left<=mid AND right<=high
     if A[left]<=A[right]
       tempA[temp++]=A[left++]
     else
       tempA[temp++]=A[right++]
```

Step 1

Look what's happening here - temp++ means the temp variable increments **after** assignment

Part II: the Merge

Step 2

```
........
if left>mid                  //nothing remains on lhs
    while right<=high
        tempA[temp++]=A[right++]
else                         //nothing remains on rhs
    while left<=mid
        tempA[temp++]=A[left++]
```

Step 3

```
for i=low to i<=high do
    A[i] = tempA[i]   //copy temp to actual
```

Thank You !

OLLSCOIL TEICNEOLAÍOCHTA BHAILE ÁTHA CLIATH
TU DUBLIN
TECHNOLOGICAL UNIVERSITY DUBLIN