Outline

1. Problem Overview
2. Selection Sort
3. Merge Sort
4. Lower Bound for Comparison Based Sorting
5. Non-Comparison Based Sorting Algorithms

Example: merge sort

7	2	5	3	7	13	1	6

Example: merge sort

7	2	5	3	7	13	1	6

split the array into two halves

7	2	5	3

7	13	1	6

Example: merge sort

7	2	5	3	7	13	1	6

split the array into two halves

7	2	5	3

7	13	1	6

sort the halves recursively

2	3	5	7

1	6	7	13

Example: merge sort

7	2	5	3	7	13	1	6

split the array into two halves

7	2	5	3

7	13	1	6

sort the halves recursively

2	3	5	7

1	6	7	13

merge the sorted halves into one array

1	2	3	5	6	7	7	13

MergeSort($A[1 \ldots n]$)

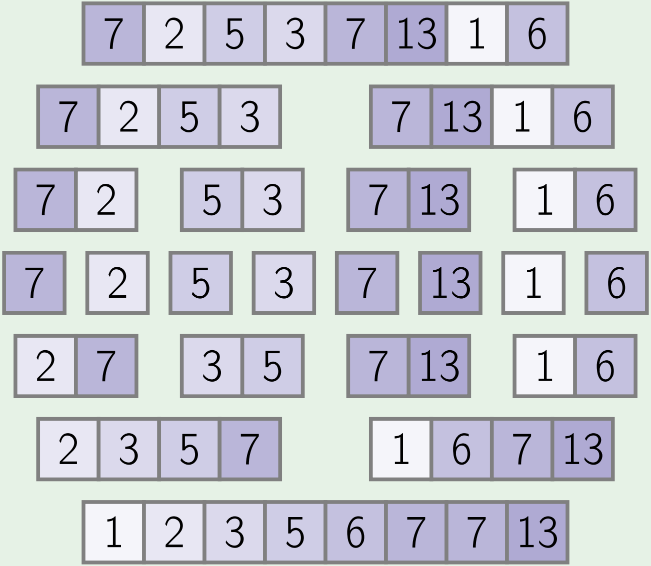
```
if n = 1:
  return A
m ← ⌊n/2⌋
B ← MergeSort(A[1 ... m])
C ← MergeSort(A[m + 1 ... n])
A' ← Merge(B, C)
return A'
```

Merging Two Sorted Arrays

Merge($B[1\dots p], C[1\dots q]$)

```
{B and C are sorted}
D ← empty array of size p + q
while B and C are both non-empty:
  b ← the first element of B
  c ← the first element of C
  if b ≤ c:
    move b from B to the end of D
  else:
    move c from C to the end of D
move the rest of B and C to the end of D
return D
```

Merge sort: example



Lemma

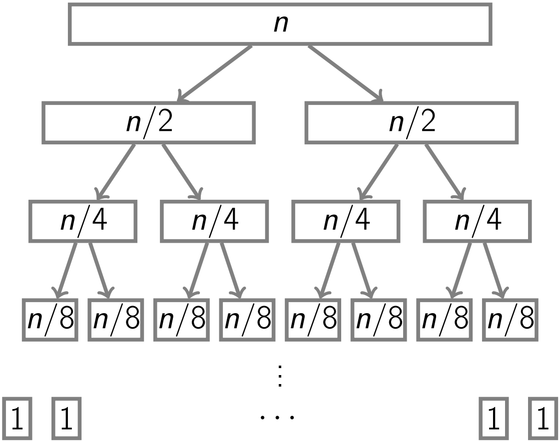
The running time of `MergeSort`$(A[1 \dots n])$ is $O(n \log n)$.

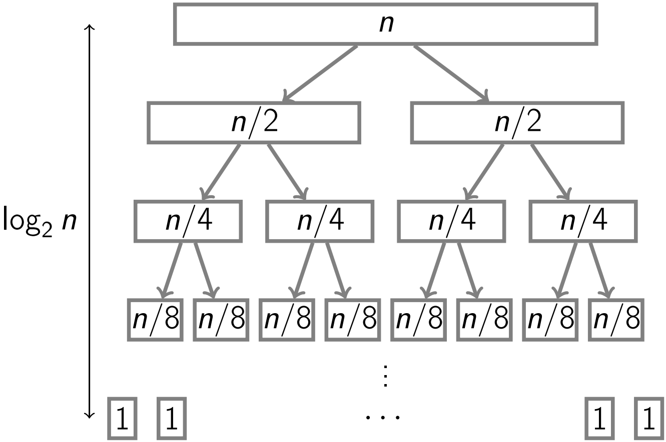
Lemma

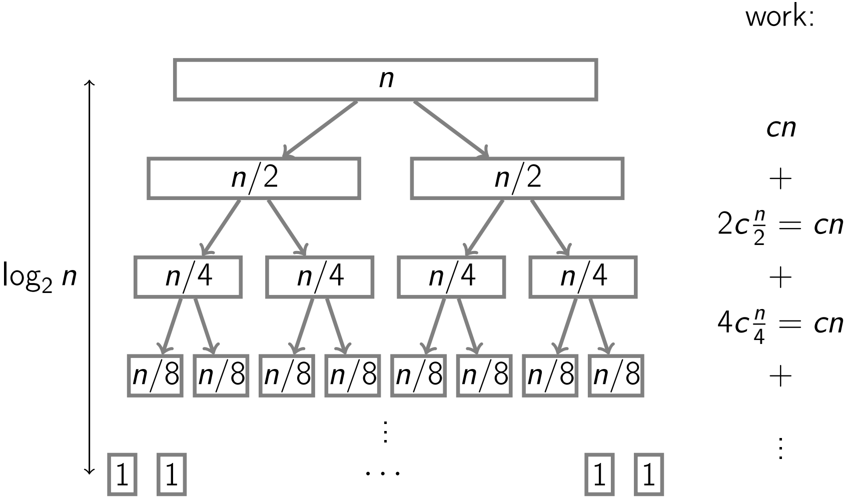
The running time of `MergeSort`$(A[1 \dots n])$ is $O(n \log n)$.

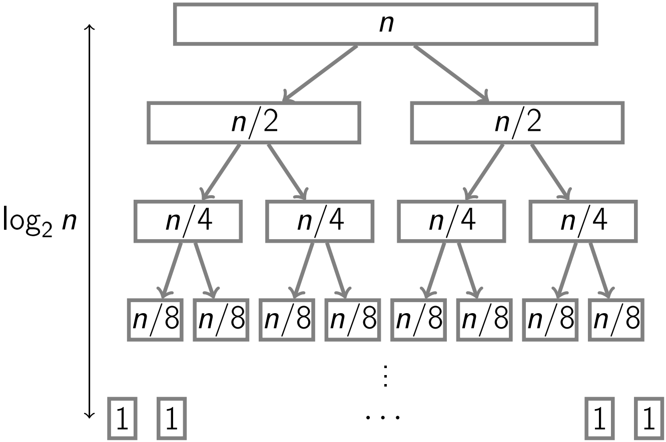
Proof

- The running time of merging B and C is $O(n)$.
- Hence the running time of `MergeSort`$(A[1 \dots n])$ satisfies a recurrence $T(n) \leq 2T(n/2) + O(n)$.


n
$n/2$
$n/2$
$n/4$
$n/4$
$n/4$
$n/4$
$n/8$
$n/8$
$n/8$
$n/8$
$n/8$
$n/8$
$n/8$
$n/8$
1
1
1
1


n
$n/2$
$n/2$
$n/4$
$n/4$
$n/4$
$n/4$
$n/8$
$n/8$
$n/8$
$n/8$
$n/8$
$n/8$
$n/8$
$n/8$
1
1
1
1
$\log_2 n$


work:
n
n/2
n/2
n/4
n/4
n/4
n/4
n/8
n/8
n/8
n/8
n/8
n/8
n/8
n/8
1
1
1
1
log₂ n
cn
+
2c n/2 = cn
+
4c n/4 = cn
+



work:

cn

$+$

$2c\frac{n}{2} = cn$

$+$

$4c\frac{n}{4} = cn$

$+$

$\vdots$

Total: $cn \log_2 n$