CSC 225

Algorithms and Data Structures I
Fall 2014
Rich Little

Algorithm Design Technique
Divide and Conquer

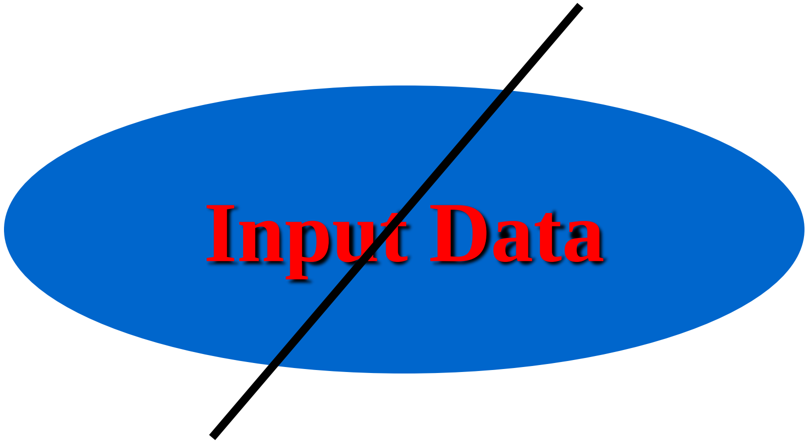
- Best-know general algorithm design technique
- Some very efficient algorithms are direct results of this technique
  - Mergesort
  - Quicksort
  - Linear selection/median

Algorithm Design Technique
Divide and Conquer

- The problem instance is divided into smaller instances of the same problem, ideally of about the same size (typically n/2)
- The smaller instances are solved (typically recursively, though sometimes a different algorithm is employed when instances become small enough)
- If necessary, the solution obtained for the smaller instances are combined to get a solution to the original instance

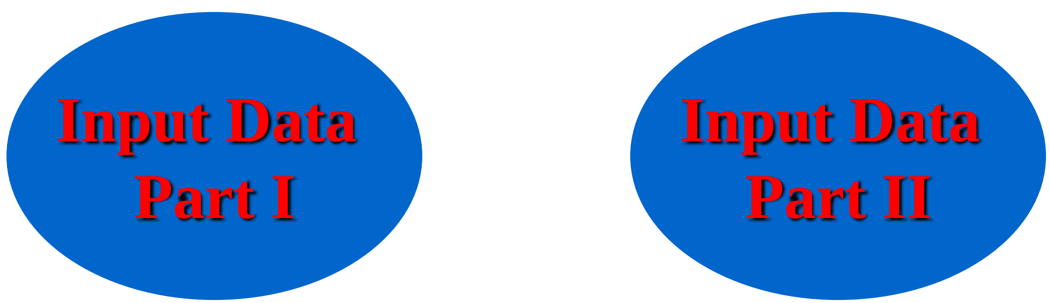
Algorithm Design Technique
Divide and Conquer

1. **Divide:** If the input size is smaller than a certain threshold, solve the problem directly. Otherwise, divide the input data into two or more disjoint subsets.

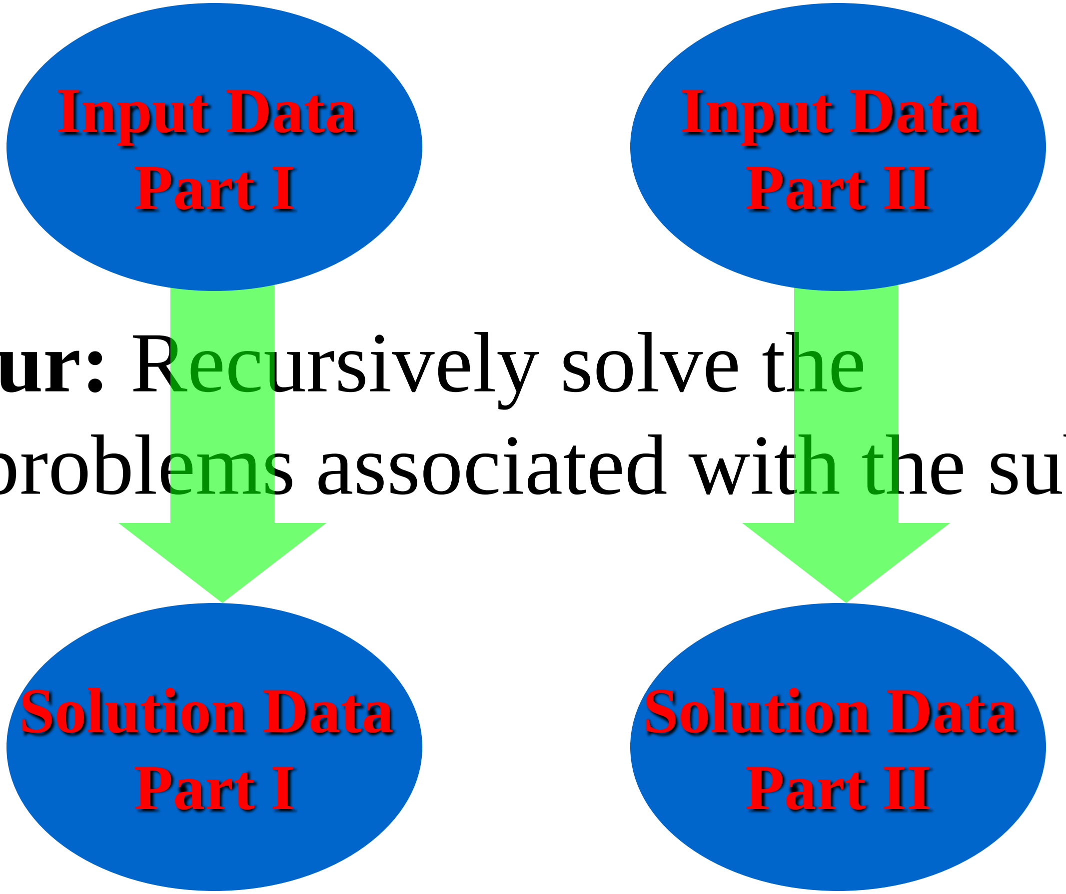


Divide and Conquer

1. **Divide:** If the input size is smaller than a certain threshold, solve the problem directly. Otherwise, divide the input data into two or more disjoint subsets.
2. **Recur:** Recursively solve the subproblems associated with the subsets.



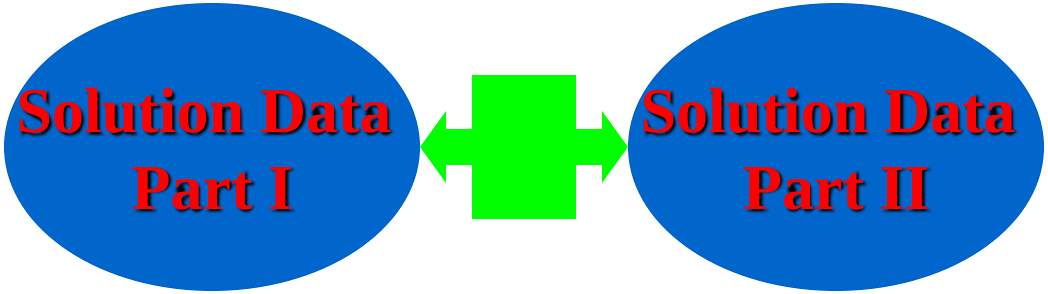
Divide and Conquer



2. **Recur:** Recursively solve the subproblems associated with the subsets.

Divide and Conquer

3. **Conquer:** Take the solutions to the subproblems and merge them into a solution to the original problem.



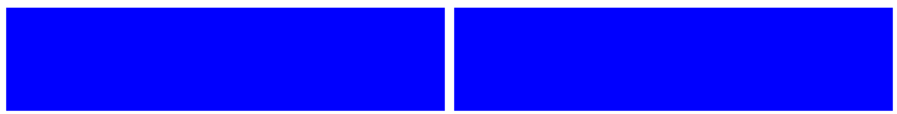
Divide and Conquer

3. **Conquer:** Take the solutions to the subproblems and merge them into a solution to the original problem.

Solution All Data

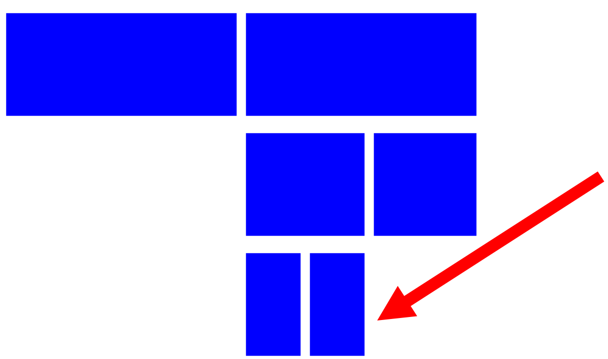
Divide and Conquer
Binary Search

- Assume array is sorted



Sorted array

- Compare search value to an element m in the middle of the array; if the search value is less than m, continue searching in left subarray; otherwise right subarray



Subarray of one element; either search value is found or not in the array

Merge Sort

Input: A collection of *n* objects (stored in a list, vector, array or sequence) and a comparator defining a total order on these objects

Output: An ordered representation of these objects

➔Apply the Divide-and-Conquer technique to the Sorting problem.

Merge Sort

Let S be a sequence with n elements

1. **Divide**
   - ✓ If S has zero or one element, return S since S is sorted.
   - ✓ Otherwise, remove all the elements from S and put them into two sequences S_1 and S_2 such that S_1 and S_2 each contain about half of the elements of S.

Merge Sort

2. **Recur**
   - ✓ Recursively sort sequences S_1 and S_2
3. **Conquer**
   - ✓ Put the elements back into S by ***merging*** the sorted sequences S_1 and S_2 into a sorted sequence.

Merging Two Sorted Sequences

- Assume two sorted sequences S_1 and S_2
- Look up the smallest element of each sequence and compare the two elements
- Remove a smallest element e from these two elements from its sequence and add it to the output sequence S
- Repeat the previous two steps until one of the two sequences is empty
- Copy the remainder of the non-empty sequence to the output sequence

Algorithm merge(S_1, S_2, S)

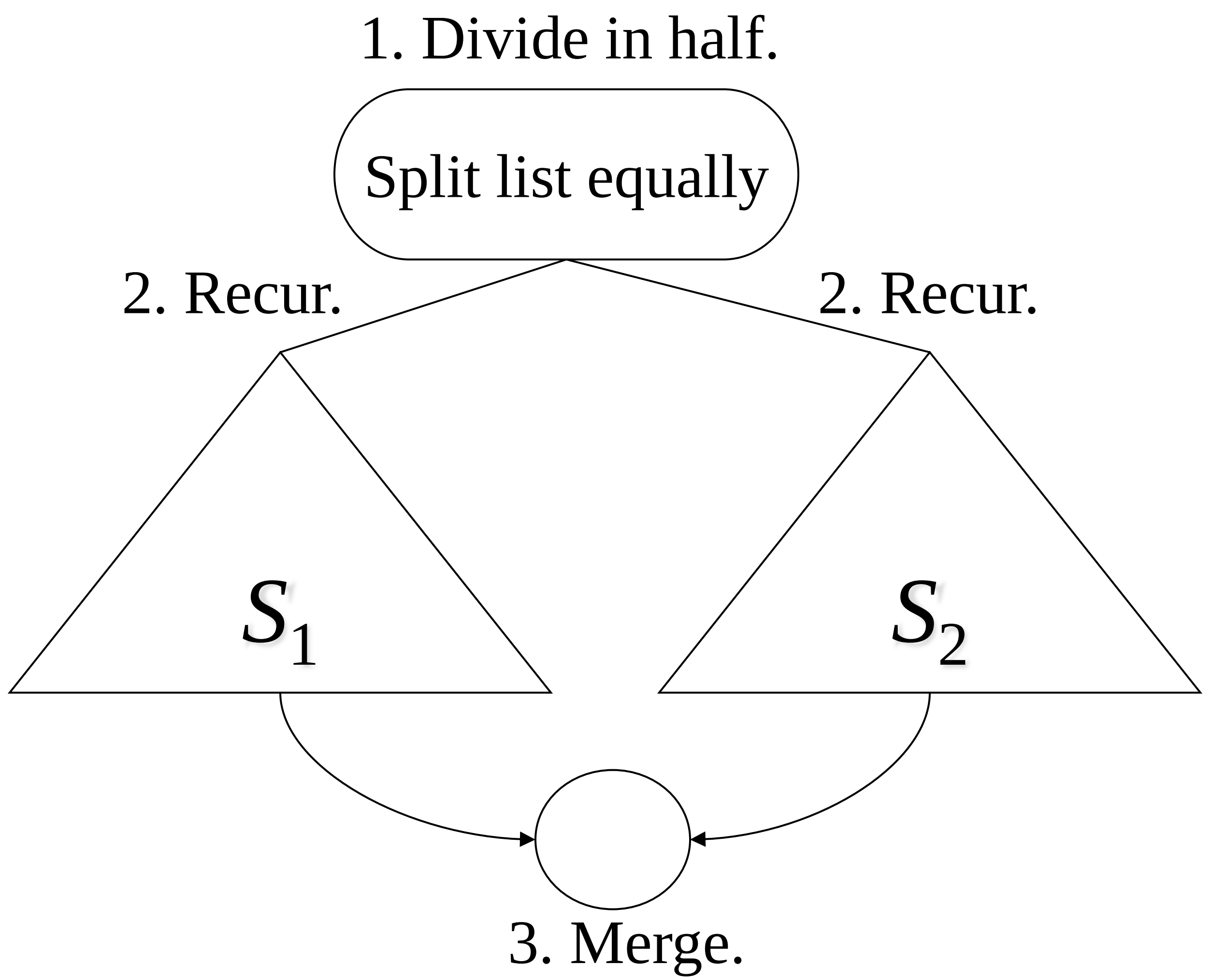
Input: Sequences S_1 and S_2 sorted in non-decreasing order; an empty output sequence S.

Output: Sequence S containing the elements from S_1 and S_2 sorted in non-decreasing order

Algorithm merge(S_1, S_2)

```
while not(S₁.isEmpty() or S₂.isEmpty()) do
  if S₁.first().key() < S₂.first().key() then
      S.insertLast(S₁.removeFirst())
  else
      S.insertLast(S₂.removeFirst()))
  end
end
while not(S₁.isEmpty()) do
  S.insertLast(S₁.removeFirst())
end
while not(S₂.isEmpty()) do
  S.insertLast(S₂.removeFirst())
end
return S
```

Merge Sort Algorithm



Algorithm mergeSort(S)

```
if S.size() < 2 then return S
S1, S2 ← divide(S)
S1 ← mergeSort(S1)
S2 ← mergeSort(S2)
S ← merge(S1, S2)
return S
```

Algorithm divide(S_1, S_2, S)

- ✓ S is a sequence containing n elements
- ✓ Let S_1 and S_2 be empty sequences

for $i \leftarrow 0$ **to** $\lfloor n/2 \rfloor$ **do**
  S_1.insertLast(S.*removeFirst*())
end
for $i \leftarrow \lfloor n/2 \rfloor + 1$ **to** n-*1* **do**
  S_2.insertLast(S.*removeFirst*())
end

Worst-case Running Time of Merge Sort

```
if S.size() < 2 then return S          b
S1, S2 ← divide(S)          n
S1 ← mergeSort(S1)          T(n/2)
S2 ← mergeSort(S2)          T(n/2)
S ← merge(S1, S2)          n
return S
```

$$T(n) = \begin{cases} b & \text{if } n = 1 \\ 2T(\frac{n}{2}) + cn & \text{otherwise} \end{cases}$$

Solve Recurrence Equation by Repeated Substitution

$$\text{For } n \geq 3$$

$$T(n) = 2T\left(\frac{n}{2}\right) + cn$$

$$= 2\left(2T\left(\frac{1}{2}\frac{n}{2}\right) + c\frac{n}{2}\right) + cn$$

$$= 2\left(2T\left(\frac{n}{2^2}\right) + c\frac{n}{2}\right) + cn$$

$$= 2^2 T\left(n/2^2\right) + cn + cn$$

$$= 2^2 T\left(n/2^2\right) + 2cn$$

Another Substitution

$$\text{For } n \geq 4: \; T(n) = 2T\left(\frac{n}{2}\right) + cn$$

$$= 2\left(2T\left(\frac{n}{2^2}\right) + c\frac{n}{2}\right) + cn$$

$$= 2\left(2\left(T\left(\frac{1}{2}\frac{n}{2^2}\right) + c\frac{n}{2}\right) + c\frac{n}{2}\right) + cn$$

$$= 2\left(2\left(T\left(\frac{n}{2^3}\right) + c\frac{n}{2}\right) + c\frac{n}{2}\right) + cn$$

$$= 2^3 T\left(\frac{n}{2^3}\right) + 2cn + cn$$

$$= 2^3 T\left(\frac{n}{2^3}\right) + 3cn$$

Repeated Substitution

For $n \geq i+1$

$$T(n) = 2^i T\left(\frac{n}{2^i}\right) + icn$$

The Substitution Terminates with T(1)

$$T(n) = b \qquad\qquad \text{if } n = 1$$

After how many recursion calls is $n = 1$?

When $2^i = n$ or after $i = \log n$ times.

The Final Steps

Since $T(n) = 2^i T\left(\frac{n}{2^i}\right) + icn \;\; (\text{for } n \geq i+1)$

For $i = \log n$

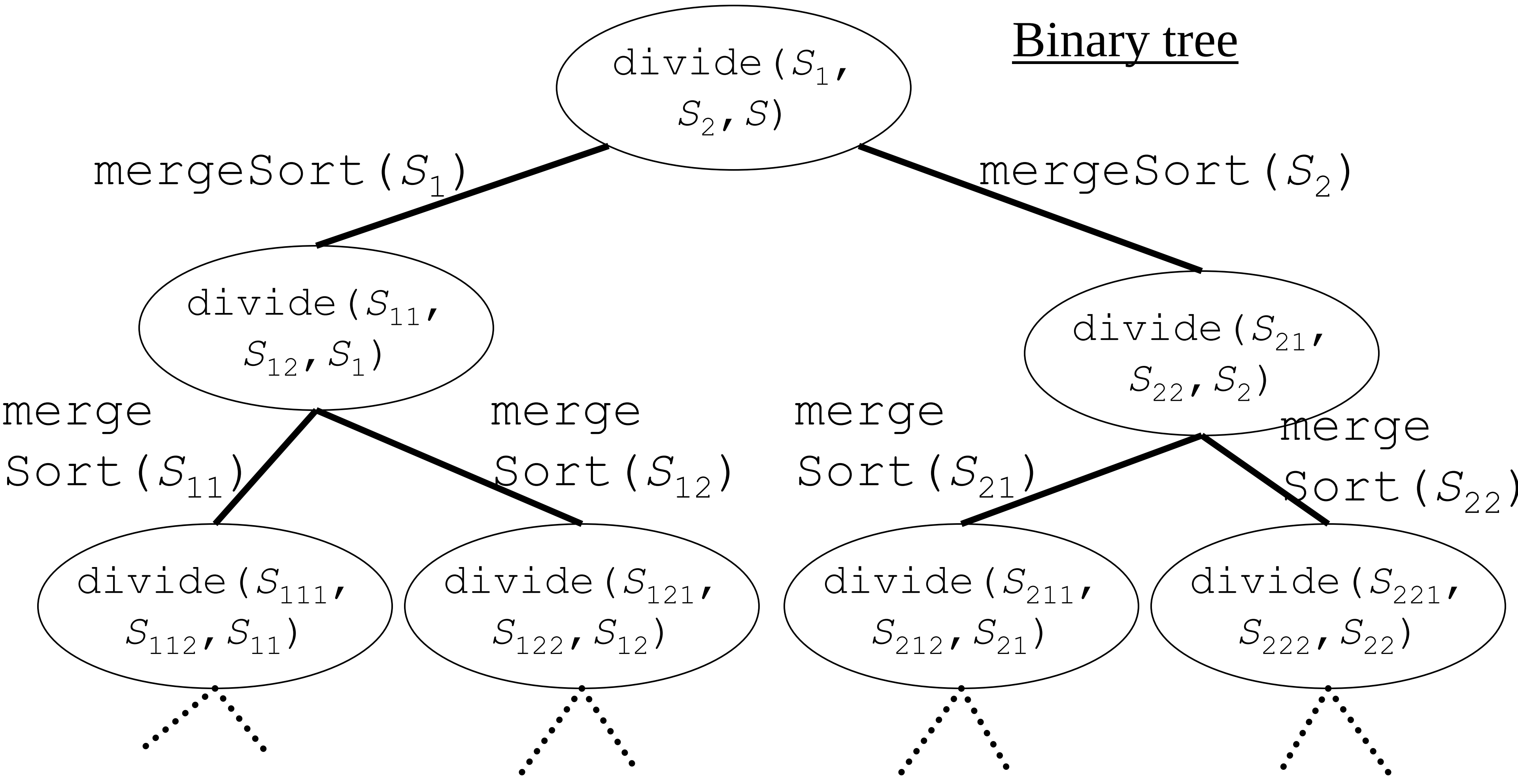
$$T(n) = 2^{\log n} T\left(\frac{n}{2^{\log n}}\right) + cn \log n$$

$$= nT\left(\frac{n}{n}\right) + cn \log n$$

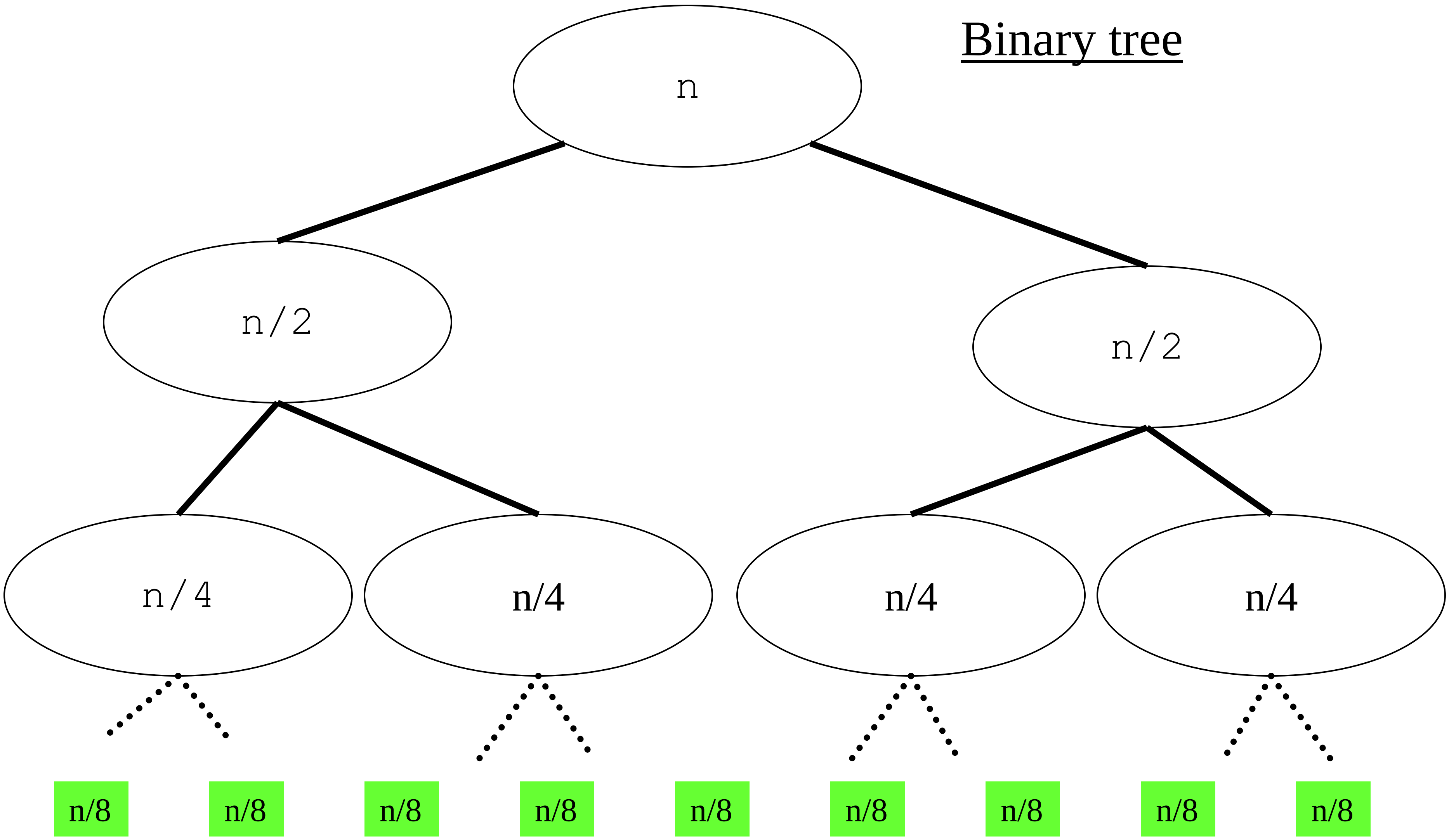
$$= nT(1) + cn \log n$$

$$= nb + cn \log n \quad \text{is } O(n \log n)$$

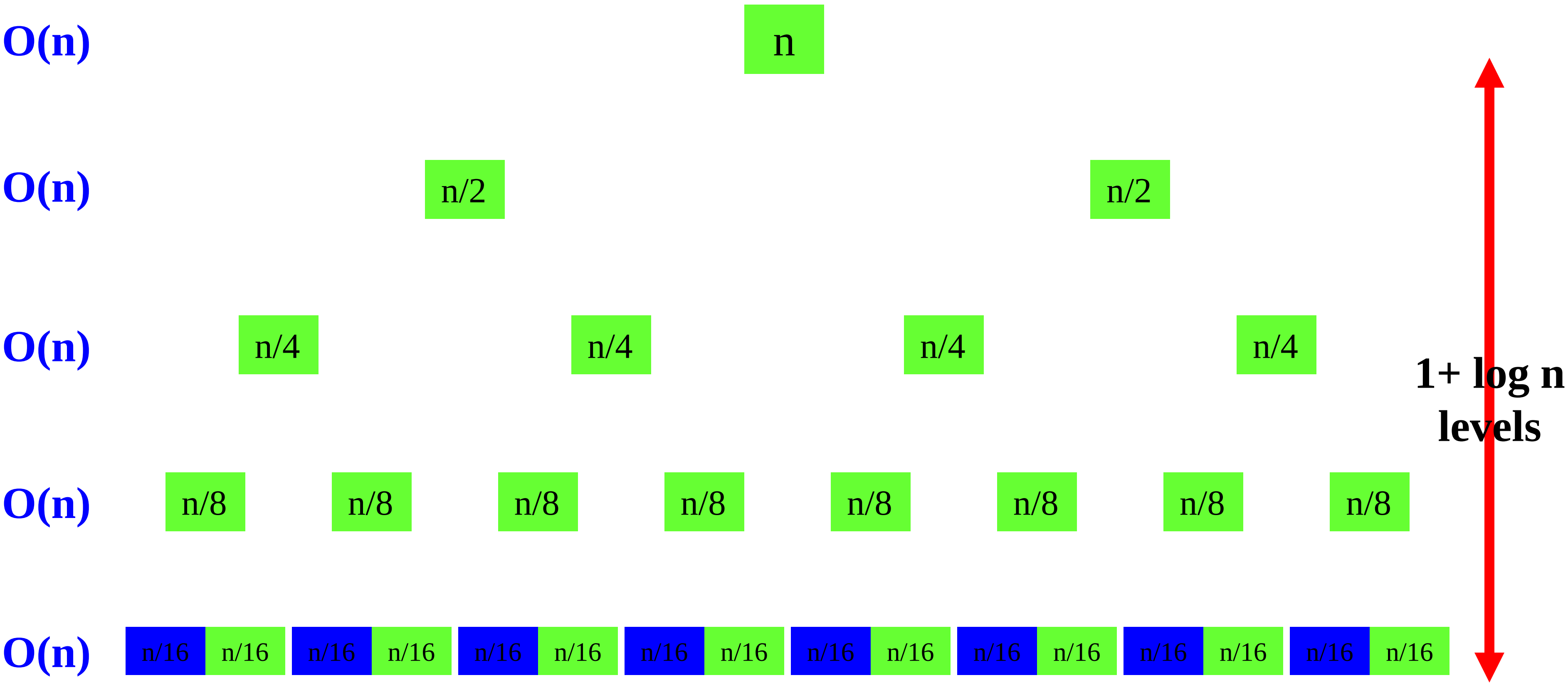
Depth of Recursion of Merge Sort



Depth of Recursion of Merge Sort



Depth of Recursion of Merge Sort



n leaves nodes
n-1 interior nodes

$$\boxed{T(n) = n \log n \in O(n \log n)}$$