CSC 225

Algorithms and Data Structures I

Rich Little
rlittle@uvic.ca
ECS 516

Algorithm Design Technique
Divide and Conquer

- Best-known 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

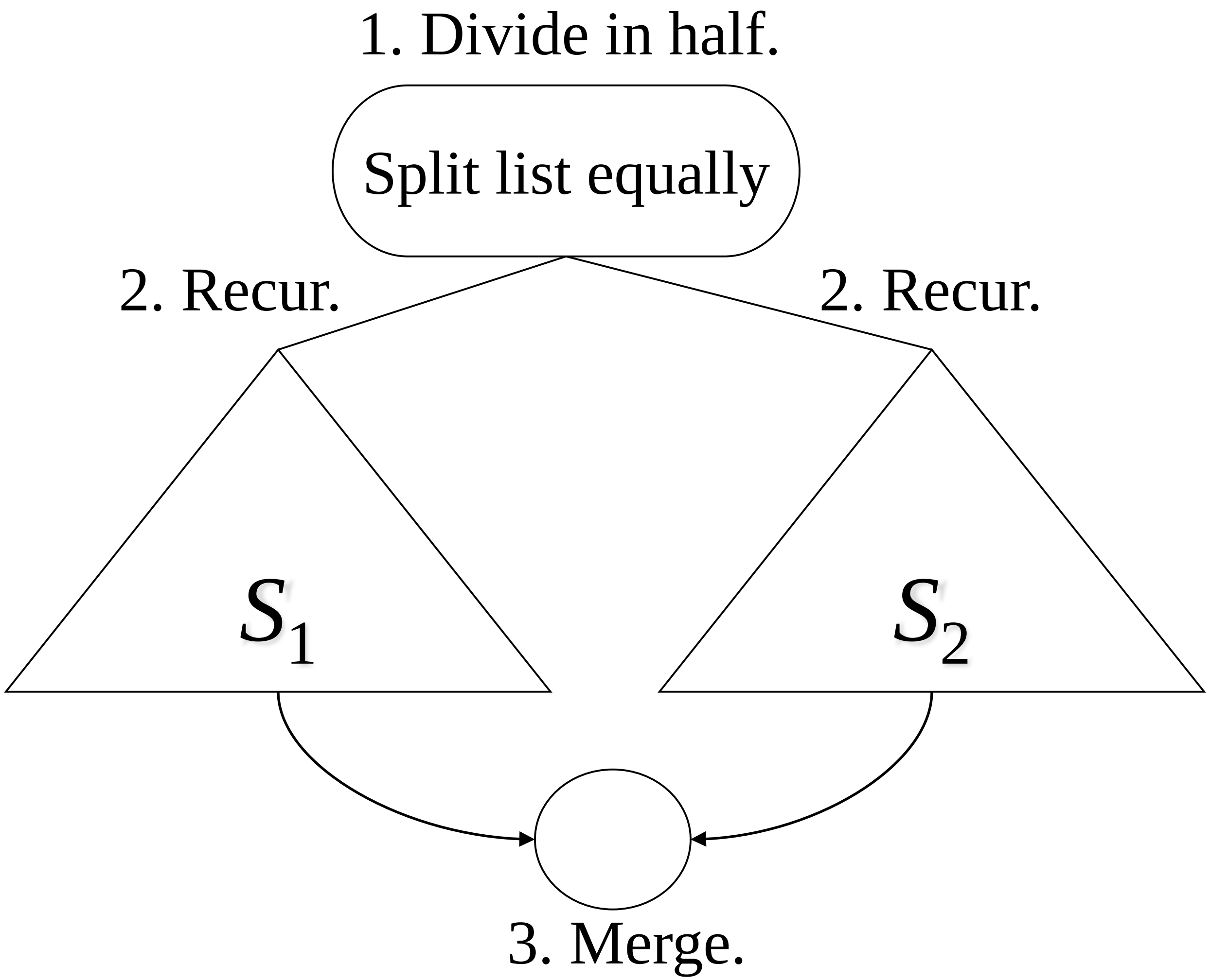
Merge-Sort

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 Algorithm



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.

Example

Let $S = [8,1,11,4,12,3,7,5]$ and sort using merge-sort.

Algorithm mergeSort(S)

```
if S.size() < 2 then
    return S
divide(S1, S2 , S)
S1 ← mergeSort(S1)
S2 ← mergeSort(S2)
merge(S1, S2, S)
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 ← 0 to ⌊n/2⌋ do
   S1[i] ← S[i]
end
for i ← ⌊n/2⌋ + 1 to n-1 do
   S2[i] ← S[i]
end
```

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: Arrays S_1 and S_2 sorted in non-decreasing order; an empty output arrray S.

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

```
i ← 1
j ← 1
while i ≤ n1 and j ≤ n2 do
   if S1[i] ≤ S2[j] then
         S[i+j-1] ← S1[i]
         i ← i + 1
   else
         S[i+j-1] ← S2[j]
         j ← j + 1
while i ≤ n1 do
   S[i+j-1] ← S1[i]
   i ← i + 1
while j ≤ n2 do
   S[i+j-1] ← S2[j]
   j ← j + 1
```

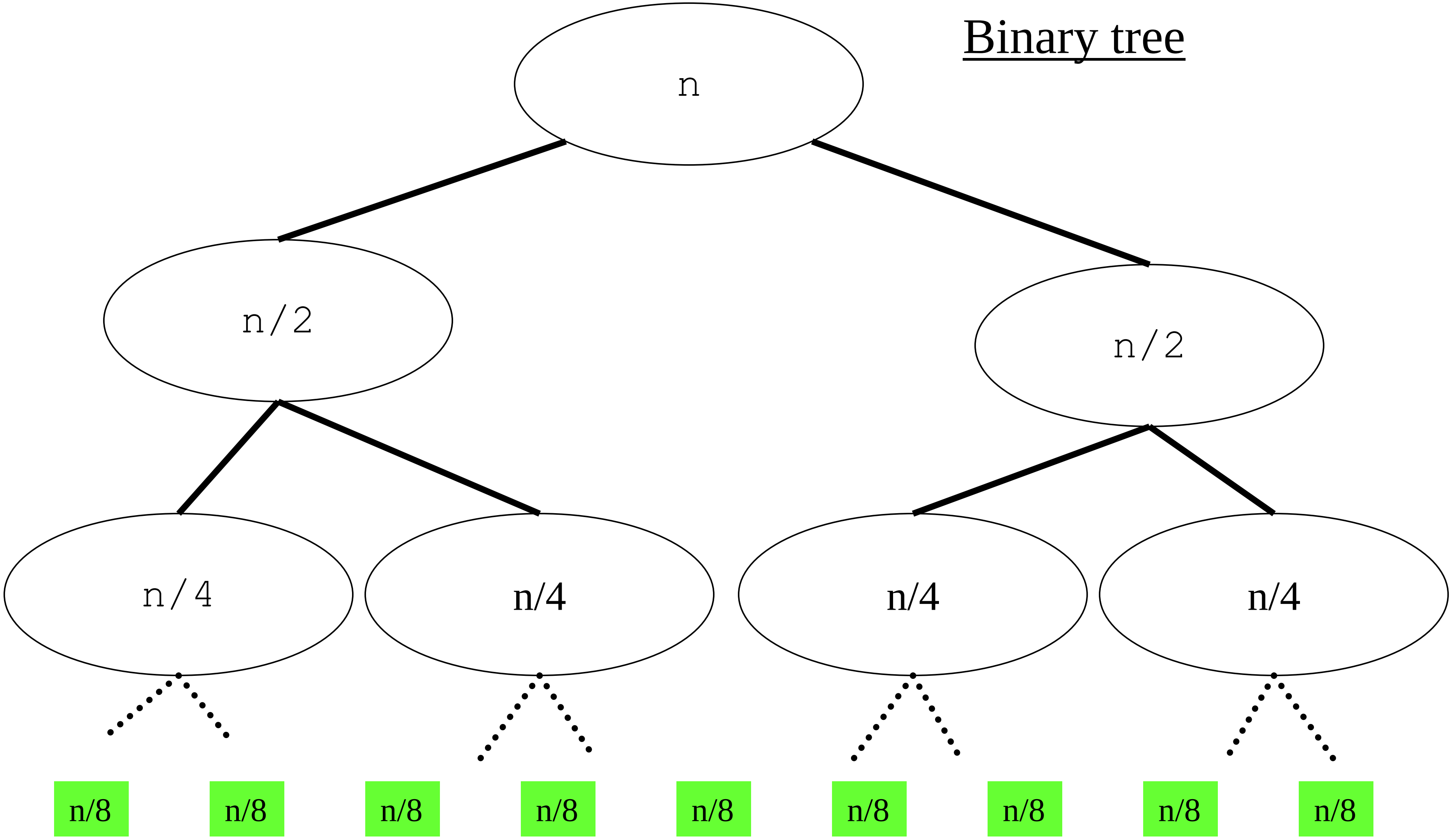
Worst-case Running Time of Merge-Sort

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

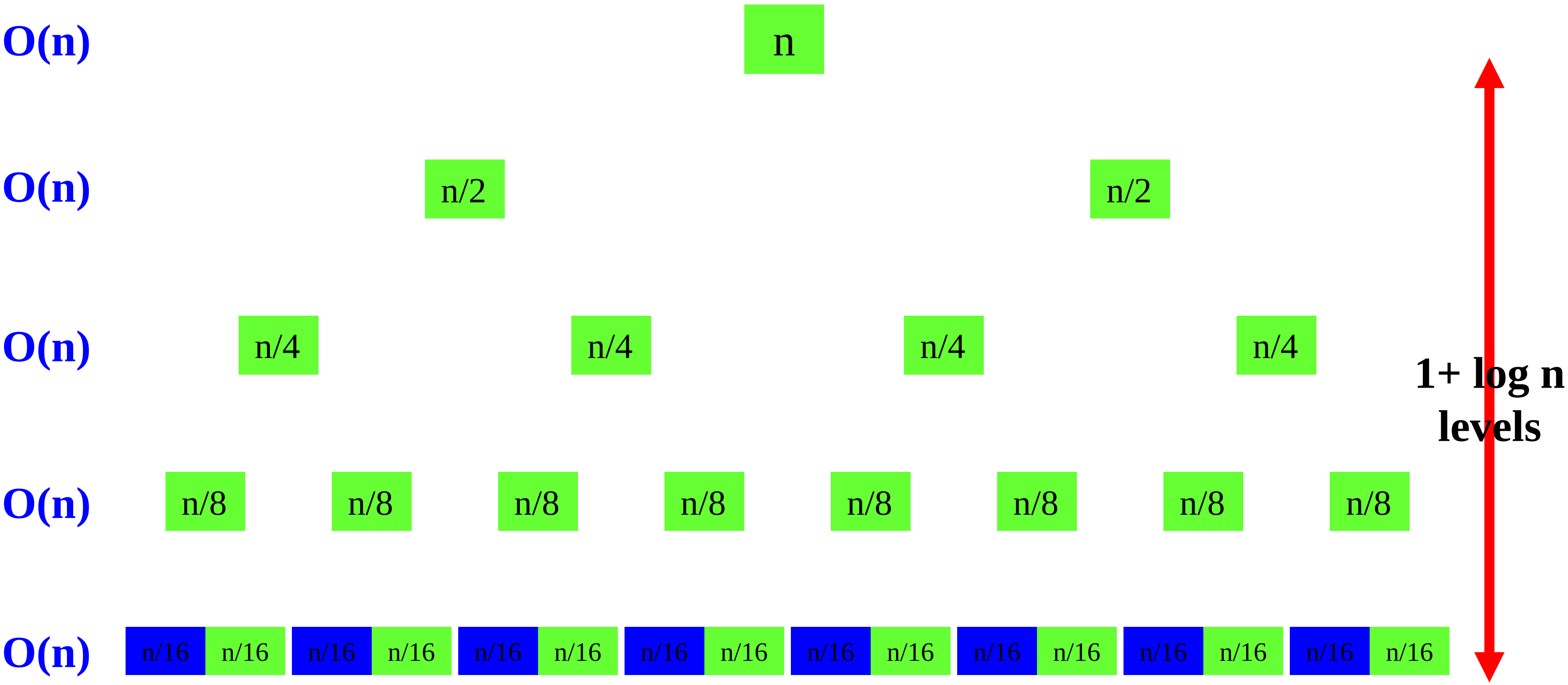
Solve Recurrence Equation by Repeated Substitution

Another Substitution

Depth of Recursion of Merge Sort



Depth of Recursion of Merge Sort



n leaves nodes
n-1 interior nodes

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