Lecture 2: MergeSort

CS 341: Algorithms

Thu, Jan 10th 2019

Outline For Today

1. Example 1: Sorting-Merge Sort-Divide & Conquer

Sorting

- Input: An array of integers in *arbitrary* order

10	2	37	5	9	55	20

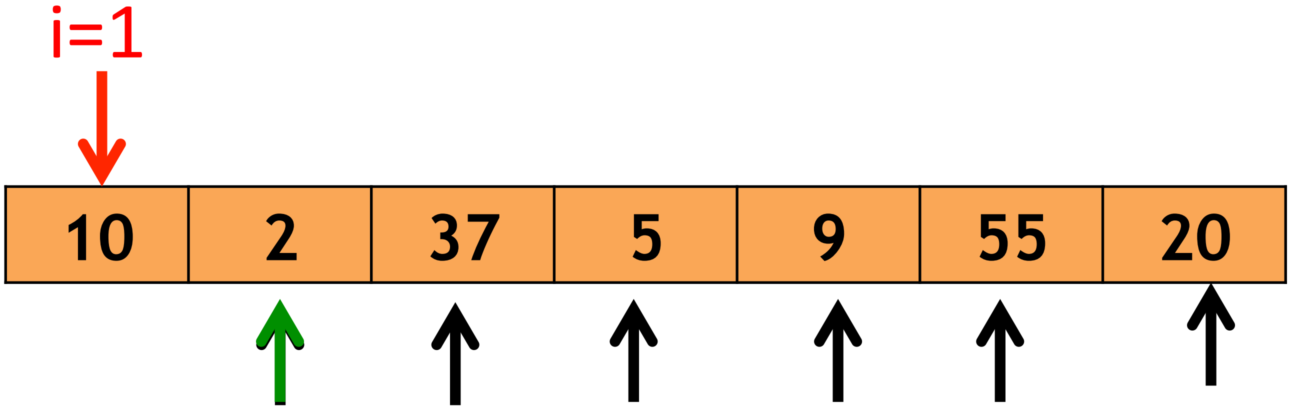
- Output: Same array of integers in *increasing* order

2	5	9	10	20	37	55

Algorithm 1: Selection Sort

```
procedure selectionSort(Array X of size n):
   for i = 1 to n {
      let minIndex = i;
      for j = i+1 to n {
         if X[j] < X[minIndex]
            minIndex = j
      }
      X[i] <--> X[minIndex] (swap in place)
   }
   return X
```

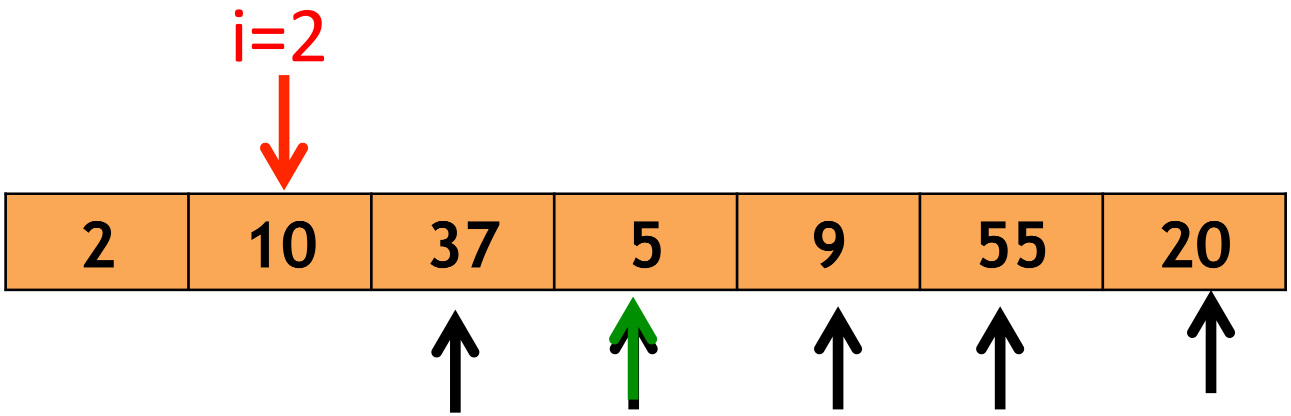
SelectionSort Simulation



minElement: 2

minIndex: 2

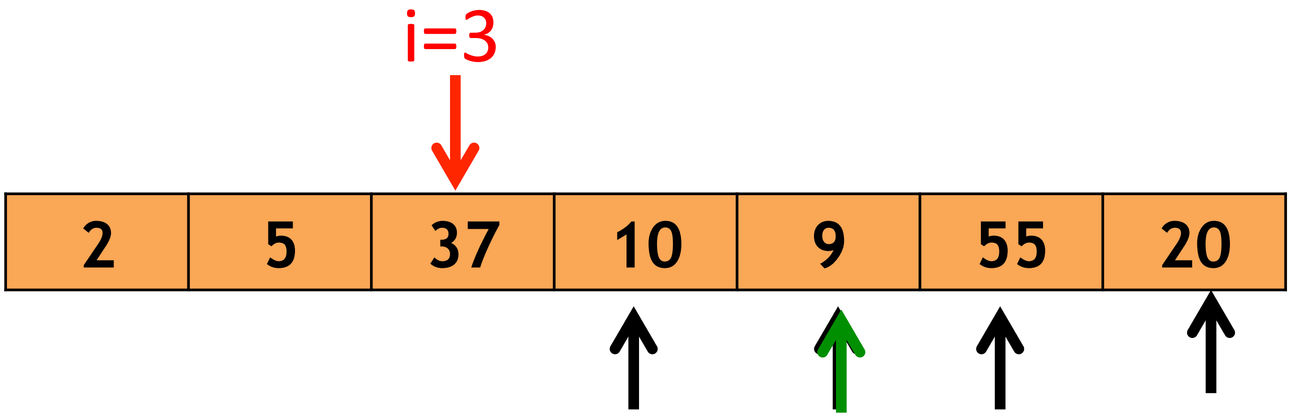
SelectionSort Simulation



minElement: 5

minIndex: 4

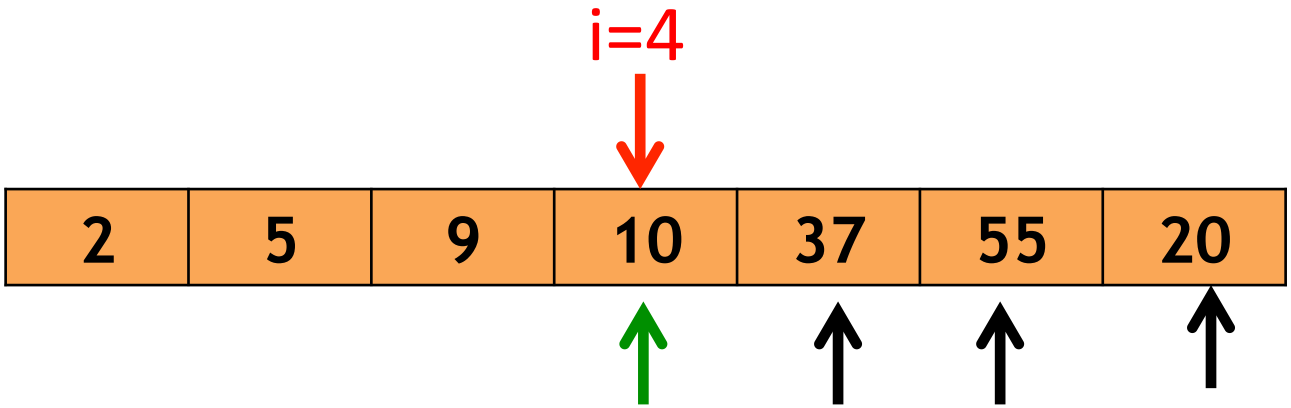
SelectionSort Simulation



minElement: 9

minIndex: 5

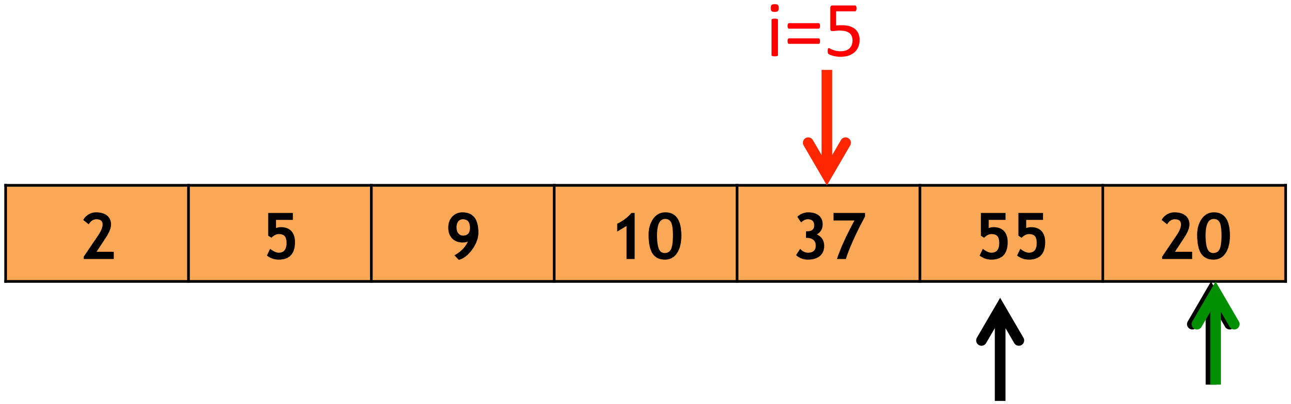
SelectionSort Simulation



minElement: 10

minIndex: 4

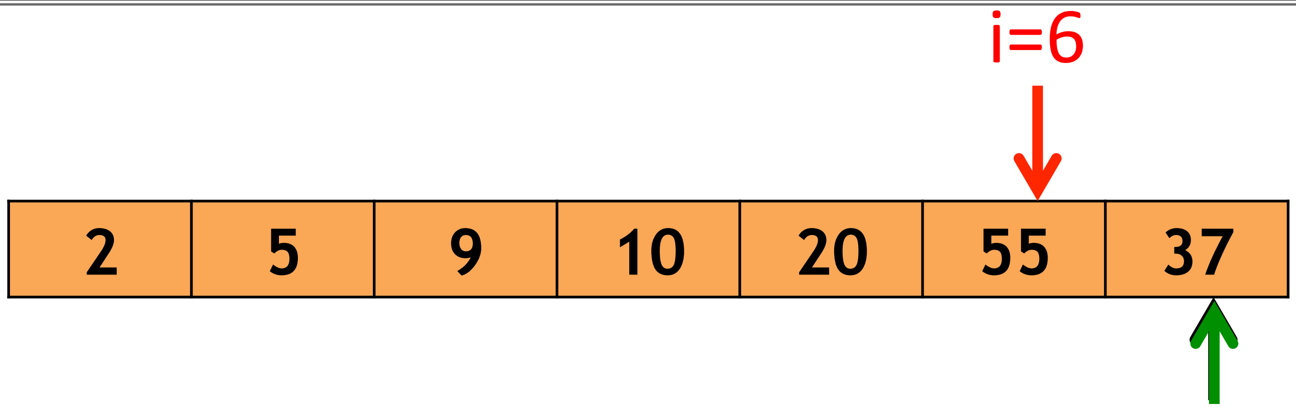
SelectionSort Simulation



minElement: 20

minIndex: 7

SelectionSort Simulation



minElement: 37

minIndex: 7

SelectionSort Simulation

2	5	9	10	20	37	55

Final Output

Analysis of Selection Sort

- Q: How much time (# operations) does SelectionSort take?

```
procedure selectionSort(Array X of size n):
    for i = 1 to n {                       1 Op
        let minIndex = i;                  1 Op
        for j = i+1 to n {                 1 Op
            if X[j] < X[minIndex]          1 Op
                minIndex = j               1 Op     3 Ops
        }
        X[i] <--> X[minIndex]              1 Op
    }
    return X
```

Analysis of Selection Sort

- Inner Loop Block: 3(n-1) + 3(n-2) + … + 3

$$3\sum_{k=1}^{n-1} k = \frac{3(n-1)n}{2} = \frac{3n^2 - 3n}{2}$$

- Outer Loop Line: n
- Initial Assignment Line: n
- Swap Line: n
- Total: $\frac{3n^2 + 3n}{2}$

SelectionSort takes $(3n^2 + 3n)/2$ time on an input of size n.

Criticism of Our Analysis & Sloppiness in CS 341

- Criticism 1: Loop increment is not 1 but 2 operations.
- Criticism 2: Swap is not 1 but 3 operations.
- Criticism 3: At machine level, swap might be 100 operations.

In CS 341, we'll be sloppy in our counting of what constitutes how many operations.

We'll count as "1 operation" high-level operations such as addition/subtraction/comparison/swap, etc.

- Will make more formal with Big-oh notation in a few lectures

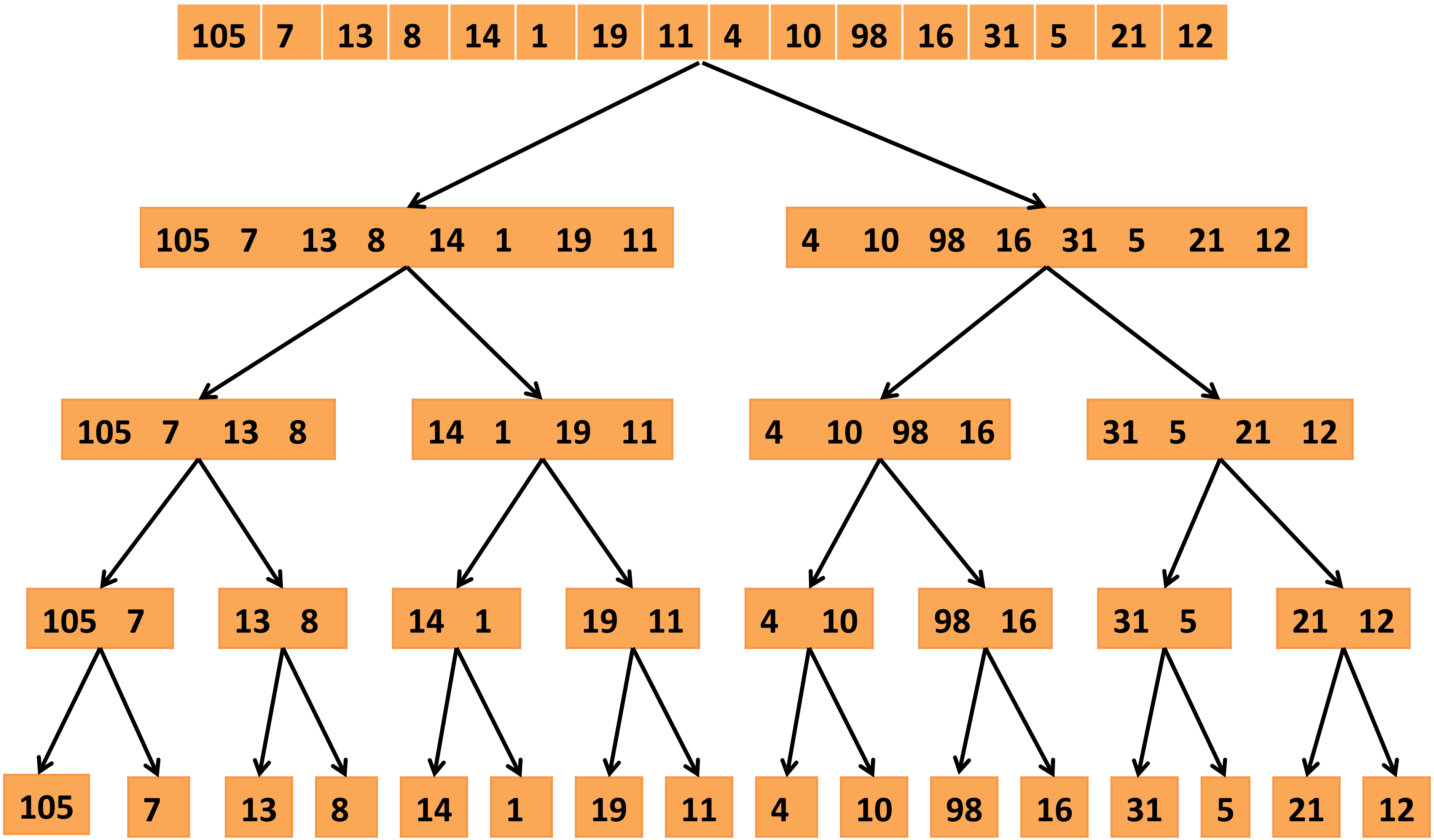
Algorithm 2: MergeSort (Divide & Conquer)

- Assume n is power of 2. (Doesn't really matter)

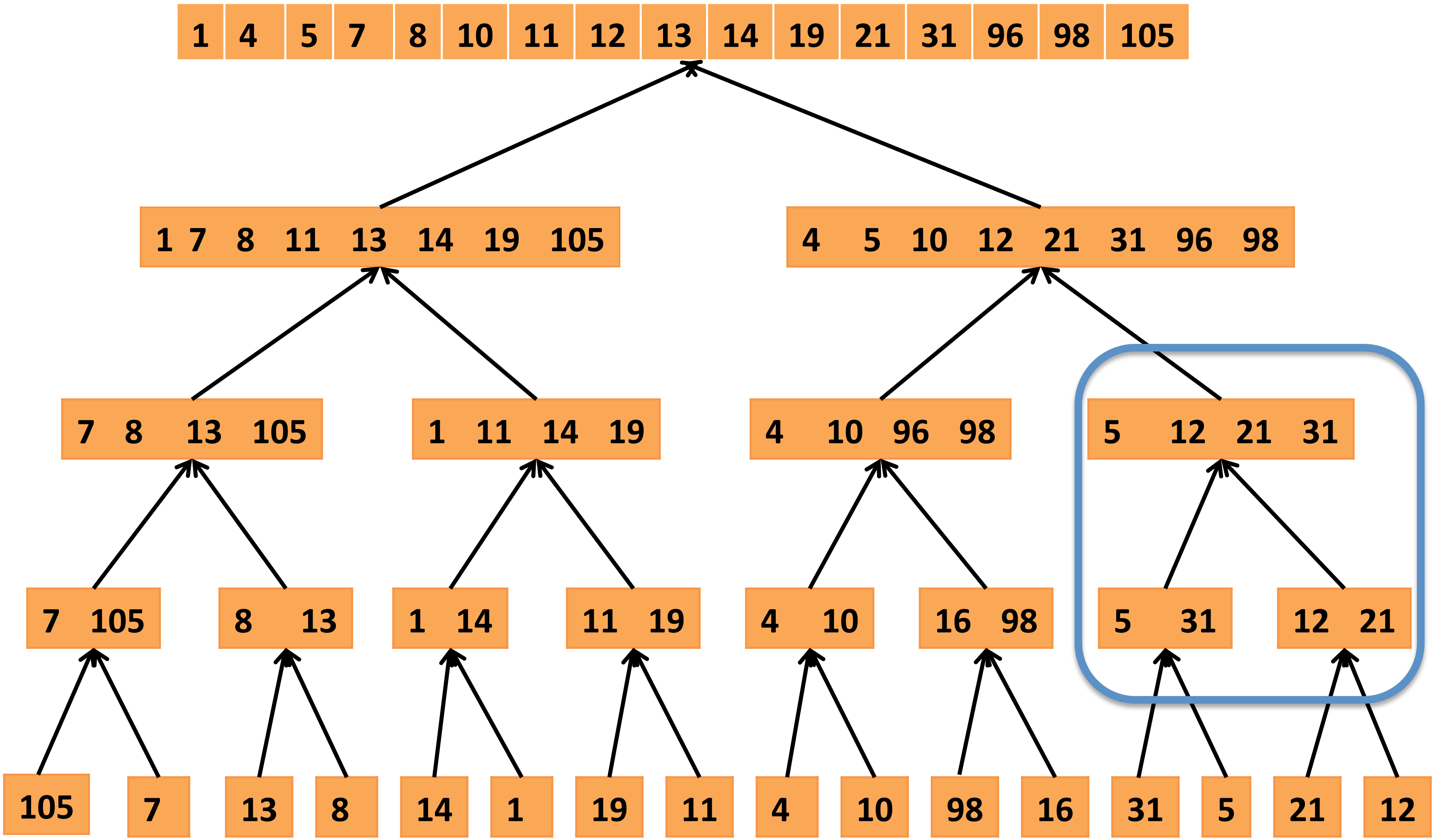
```
procedure mergeSort(Array X of size n):
    1. mergeSort(left subarray X[1,…,n/2])
    2. mergeSort(right subarray X[n/2+1,…,n])
    3. combine the left & right sorted halves
```

Will simulate how the third step is done.

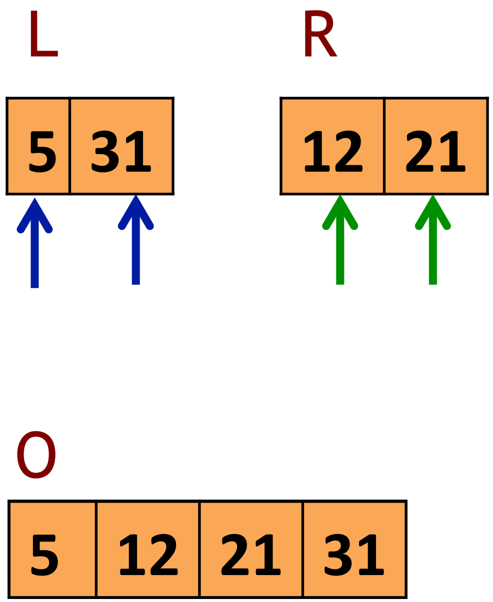
MergeSort Downward-Recursive Steps

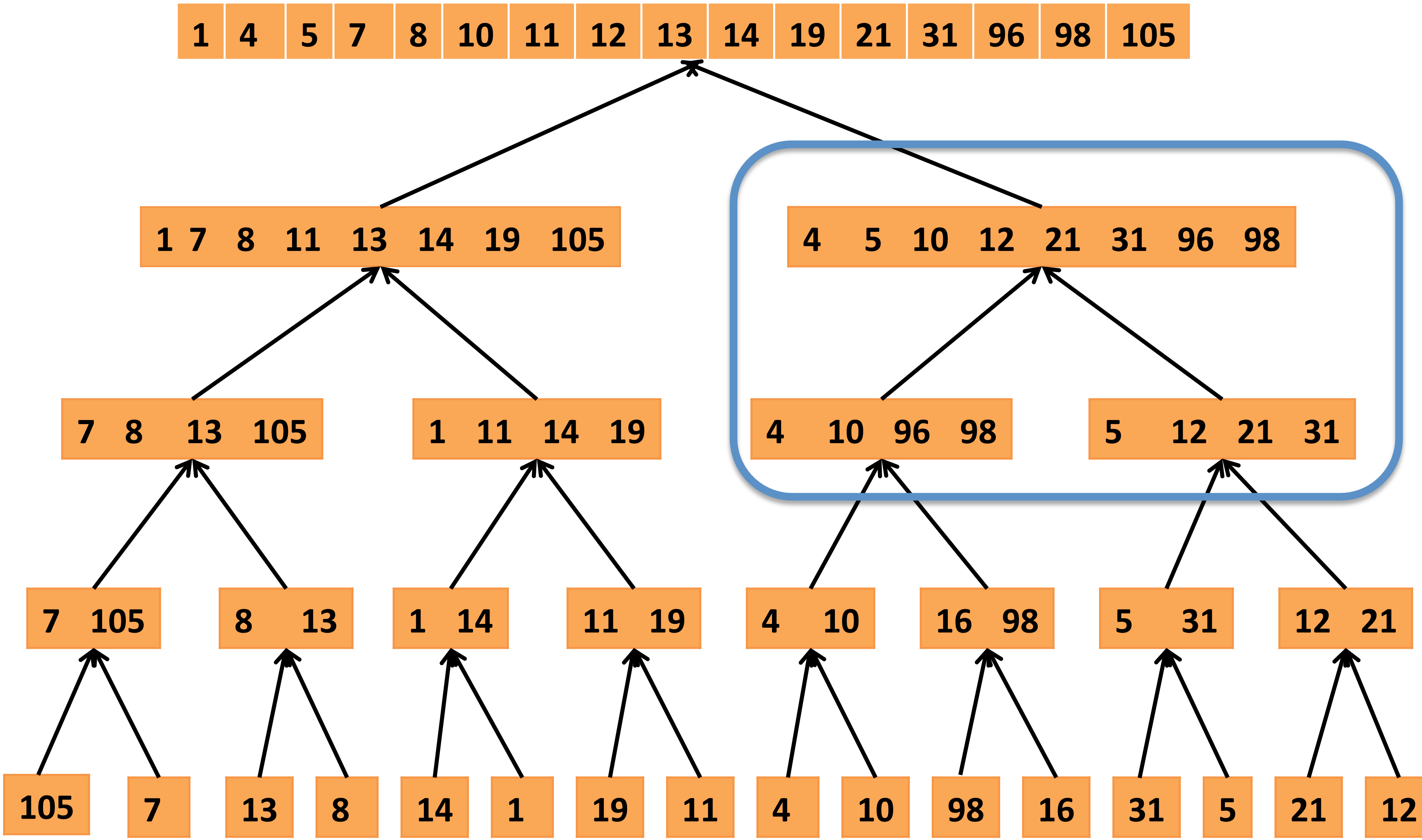


MergeSort Upward-Recursive Steps



Merge Subroutine Simulation 1




1 4 5 7 8 10 11 12 13 14 19 21 31 96 98 105
1 7 8 11 13 14 19 105
4 5 10 12 21 31 96 98
7 8 13 105
1 11 14 19
4 10 96 98
5 12 21 31
7 105
8 13
1 14
11 19
4 10
16 98
5 31
12 21
105
7
13
8
14
1
19
11
4
10
98
16
31
5
21
12

Combine Subroutine Simulation 2

L

4	10	96	98

R

5	12	21	31

O

4	5	10	12	21	31	96	98

Pseudocode for Merge Subroutine

```
procedure merge(sorted lists L,R of size m/2):
   Out = empty array of size m
   i = 1; j = 1;
   for k = 1 to m:
      if L[i] < R[j]:
         Out[k] = L[i];
         i++;
      else:
         Out[k] = R[j];
         j++;
```

Analysis of MergeSort

◆ Q: How much time (# operations) does MergeSort take?

```
procedure mergeSort(Array X of size n):
   1. mergeSort(left subarray X[1,…,n/2])
   2. mergeSort(right subarray X[n/2+1,…,n])
   3. merge the left & right sorted halves
```

Simpler Question: How many operations does the the merge subroutine (step 3 take) on an input of size m?

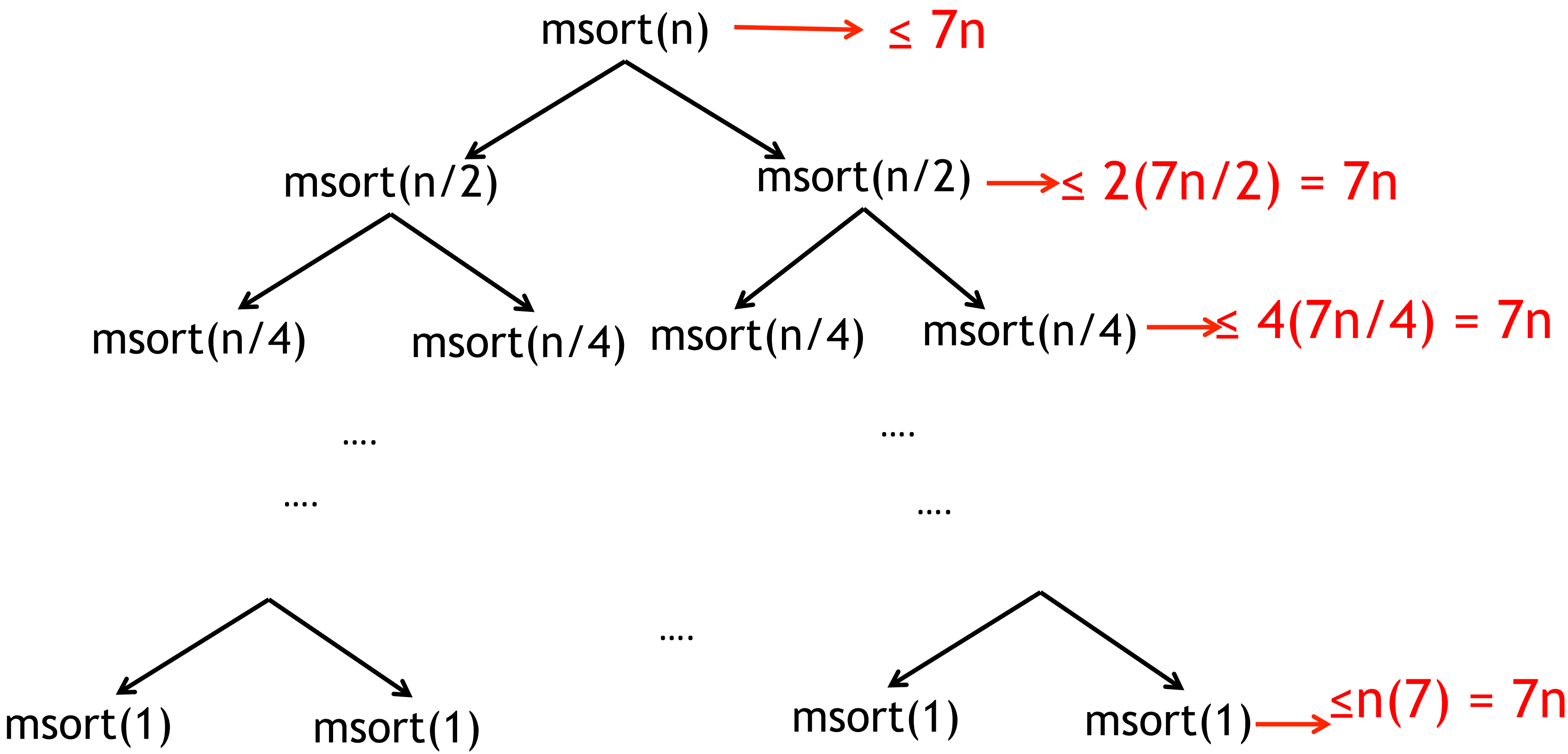
Analysis for Merge Subroutine

```
procedure merge(sorted lists L,R of size m/2):
    Out = empty array of size m          → m ops
    i = 1; j = 1;                        → 2 ops
    for k = 1 to m:                      → 1 op
        if L[i] < R[j]:                  → 1 op
            Out[k] = L[i];               → 1 op
            i++;                         → 1 Op
        else:
            Out[k] = R[j];               → 1 op
            j++;                         → 1 Op
```

4 ops

Total: $m + 2 + 4m = 5m + 2 \leq 7m$

Analysis of MergeSort



Total = 7n*#levels

Total = $7n*\log_2(n)$

MergeSort takes $7n\log_2(n) + 7n$ time on an input of

CS 341 Diagram

Fundamental *(& Fast)* Algorithms to Tractable Problems

- MergeSort
- Strassen's MM
- BFS/DFS
- Dijkstra's SSSP
- Kosaraju's SCC
- Kruskal's MST
- Floyd Warshall APSP
- Topological Sort
- …

Common Algorithm Design Paradigms

- Divide-and-Conquer
- Greedy
- Dynamic Programming

Mathematical Tools to Analyze Algorithms

- Big-oh notation
- Recursion Tree
- Master method
- Substitution method
- Exchange Arguments

Intractable Problems

- P vs NP
- Poly-time Reductions
- Undecidability

Other (Last Lecture)

- Randomized/Online/ Parallel Algorithms

CS 341 Assumptions & Justifications (1)

1. Worst-case Runtime Analysis
   - Justification 1: Easier to make worst-case analysis
   - Justification 2: Holds under any input => Very strong statement
2. "Sloppy" in counting
   - Justification: Can agree on high-level ops but impossible to agree on low-level ops
     - ✧ Will be different from language to language/architecture to architecture/compiler to compiler

CS 341 Assumptions & Justifications (2)

3. Interested in very large inputs
   - Mathematically can't say $3n^2+3n/2 > 7n\log_2(n)$ => depends on n
   - But for large n, can say that $3n^2+3n/2 > 7n\log_2(n)$
   - So we'll say: *MergeSort is better than SelectionSort*