Recursive Algorithms Exercise Hints

The Exercise
In the ***InsertionSort***, write a recurrence relating $C(n)$ with the immediately following instance, then solve. **Hint**: Write the <u>sequence</u> of instances generated by the outer loop and note sizes.

Only a simple procedure to insert a key in a sorted list is needed to relate instances n, $n-1$ (diagram).

If lucky, complete sorting after just $O(n)$ key-comps (when?)

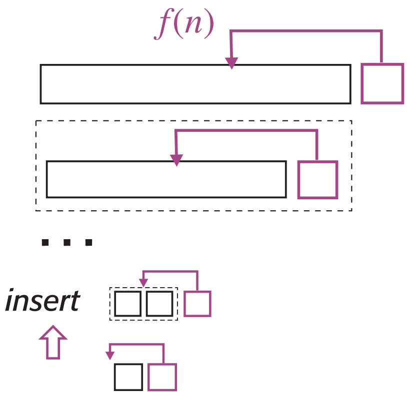
Quiz
How many key-comps are needed in the insertion round <u>when sorting input size</u> is n? *the f(n) in a recurrence (tricky, think about it)*

Generally will have to process the $n-1$ sublist first.

Quiz
What's the name of the returning phase of **recursion**?

Strategy

Insert last element in remaining $n-1$ (hopefully) sorted sublist, <u>repeat</u>



Algorithm *InsertionSort*

```
for i ← 1 to n − 1 do
    v ← A[i]
    j ← i − 1
    while j ≥ 0 and A[j] > v do
        A[j + 1] ← A[j]
        j ← j − 1
    A[j + 1] ← v
```

Insertion Sort Exercise

Algorithm *InsertionSort*

```
for i ← 1 to n − 1 do
    v ← A[i]
    j ← i − 1
    while j ≥ 0 and A[j] > v do
        A[j + 1] ← A[j]
        j ← j − 1
    A[j + 1] ← v
```

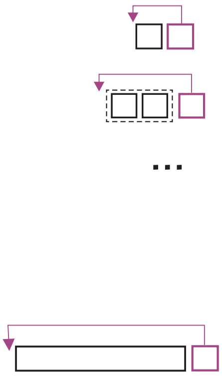
Classic bottom-up

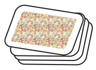
Exercise
Use the pattern diagram to write a summation for the worst-case $C(n)$. **Hint**: write the max #key-comps next to each iteration (the first one is 1 for a sorting size n=2).

Exercise
Write a pseudocode for a recursive insertion sort and code it to test. Print list after each insertion round.

Quiz
Determine the recurrence from the pseudocode.

Pattern





v

89	45 *i*	68	90	29	34	17
45	89	68 *i*	90	29	34	17
45	68	89	90	29	34	17
45	68	89	90	29	34	17
29	45	68	89	90	34	17
29	34	45	68	89	90	17
17	29	34	45	68	89	90

Efficiency described (modeled) by a math sequence of terms based on input sizes.

The count $C(n)$ of a suitable basic operation, as a function of input size n, can be used as a basis of the sequence.

Particularly, conceiving or visualizing how iterated steps are related.

Solving the recurrence gives the generic term of the sequence.

- ✎ Conceptualizing steps helps write either the generic term or a recurrence that relates the terms
- ✎ Non-recursive steps conducive for a generic term, recursive ones favor a recurrence
- ✎ Articulating steps, a pseudocode, can help directly infer a generic term or a recurrence

Sequence of Instances

➪ **[Problem] Instance**

➪ **Basic operation**

Algorithm *Factorial*
Input Integer $n \geq 0$
Output $n!$

```
fact ← 1
for i ← 1 to n do
    fact ← fact × i
return fact
```

```
if n = 0 then
    return 1
else
    return Factorial(n − 1) × n
```

```
Algorithm Factorial (n, val)
if n = 0 then
    return val
val ← val × n
return Factorial (n − 1, val)
```

	instance	instance	instance	instance	instance	
n	0 ↑	1	2	3 ↓	4	
$C(n)$	0	1	2	3	4	$\cdots$

? following term

$$\sum_{1}^{n} 1$$

$$C(n) = ?$$

$$C(n) \in \Theta(n)$$

$$C(n) = C(n-1) + 1, \quad C(0) = 0$$

$f(n)$

Sequence of Instances
Insertion Sort

Exercise
Write the **generic term** of the efficiency sequence.

Algorithm Insertion Sort
Input Array of n orderbale keys $A[0..n-1]$
Output Sorted array

```
for i ← 1 to n − 1 do
    k ← A[i]
    j ← i − 1
    while j ≥ 0 and A[j] > k do
        A[j + 1] ← A[j]
        j ← j − 1
    A[j + 1] ← k
```

```
call ins(A, n − 1)

procedure ins(A, i)
if i > 1 then
    ins(A, i − 1)
k ← A[i]                          f(n)
j ← i − 1
while j ≥ 0 and A[j] > k do
    A[j + 1] ← A[j]
    j ← j − 1
A[j + 1] ← k
```

n	2	3	4	5	6		
$C_{\text{worst}}(n)$	1	3	6	10	15	…	?

Pseudocode indicates possible early exit in the *while*-loop; in the worst-case, loops run to the end.

$$\sum_{i=1}^{n-1} \sum_{j=0}^{i-1} 1$$

$$x(n) = x(n-1) + (n-1),$$

$$x(2) = 1 \qquad f(n)$$

$$x(1) = 0$$

Insertion Sort Efficiency

Quiz
What is the total and per instance number of key comparisons for the example instance? Compare to the worst case?

```
89,45,68,90,29,34,17
45,89,68,90,29,34,17
45,68,89,90,29,34,17
45,68,89,90,29,34,17
29,45,68,89,90,34,17
29,34,45,68,89,90,17
```

Algorithm Insertion Sort

for $i \leftarrow 1$ **to** $n-1$ **do**
  $k \leftarrow A[i]$
  $j \leftarrow i-1$
  while $j \geq 0$ **and** $A[j] > k$ **do**
    $A[j+1] \leftarrow A[j]$
    $j \leftarrow j-1$
  $A[j+1] \leftarrow k$

```
89,45,68,90,29,34,17
45,89,68,90,29,34,17
45,68,89,90,29,34,17
...
29,45,68,89,90,34,17
```

call $ins(A, n-1)$

procedure $ins(A, i)$
if $i > 1$ **then**
  $ins(A, i-1)$
$k \leftarrow A[i]$
$j \leftarrow i-1$
while $j \geq 0$ **and** $A[j] > k$ **do**
  $A[j+1] \leftarrow A[j]$
  $j \leftarrow j-1$
$A[j+1] \leftarrow k$

n	2	3	4	5	6	
$C_{worst}(n)$	1	3	6	10	15	…

Exercise
Determine the best-case efficiency. Describe instances that result in best or worst case efficiency.

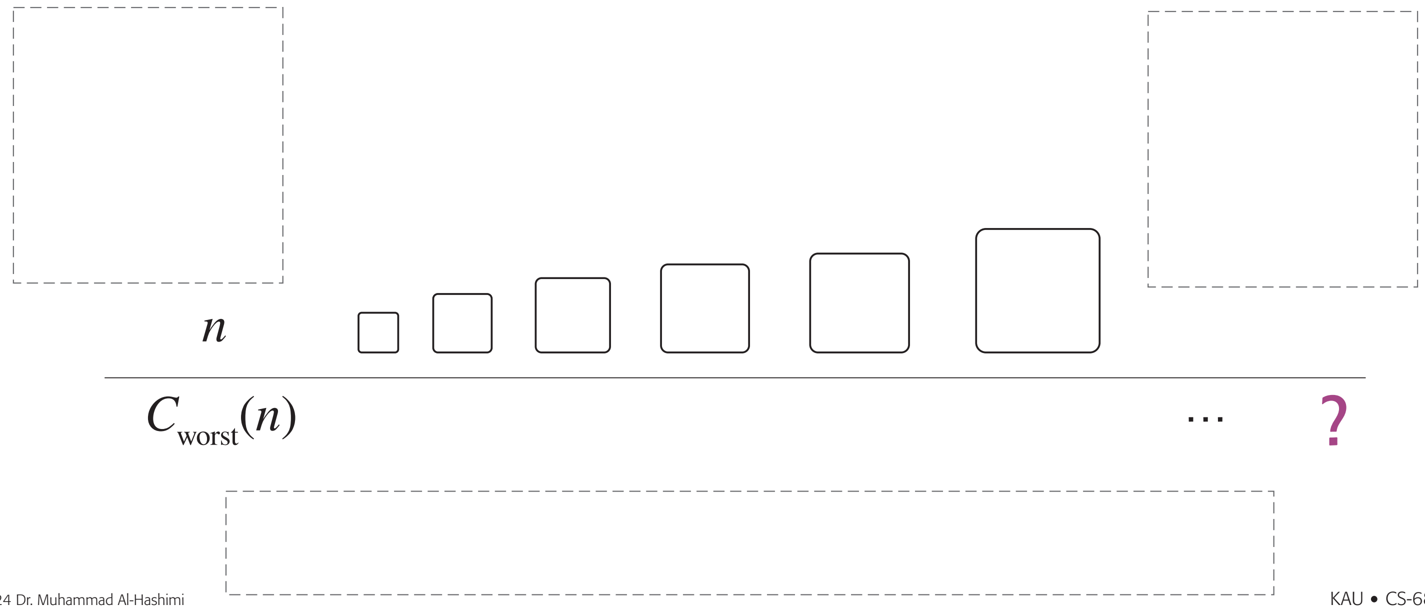
$$C_{worst}(n) \in \Theta(n^2)$$
$$C(n) \in O(n^2)$$

⇨ **Average?**

Sequence of Instances Exercise

Exercise
Write summations and/or recurrences. Characterize the efficiency? Describe instances that result in different efficiency types, if any.

Algorithm *BinarySearch*
Input $A[0..n-1]$ sorted in ascending order
Input Search key K
Output Index of key in A if found, −1 otherwise



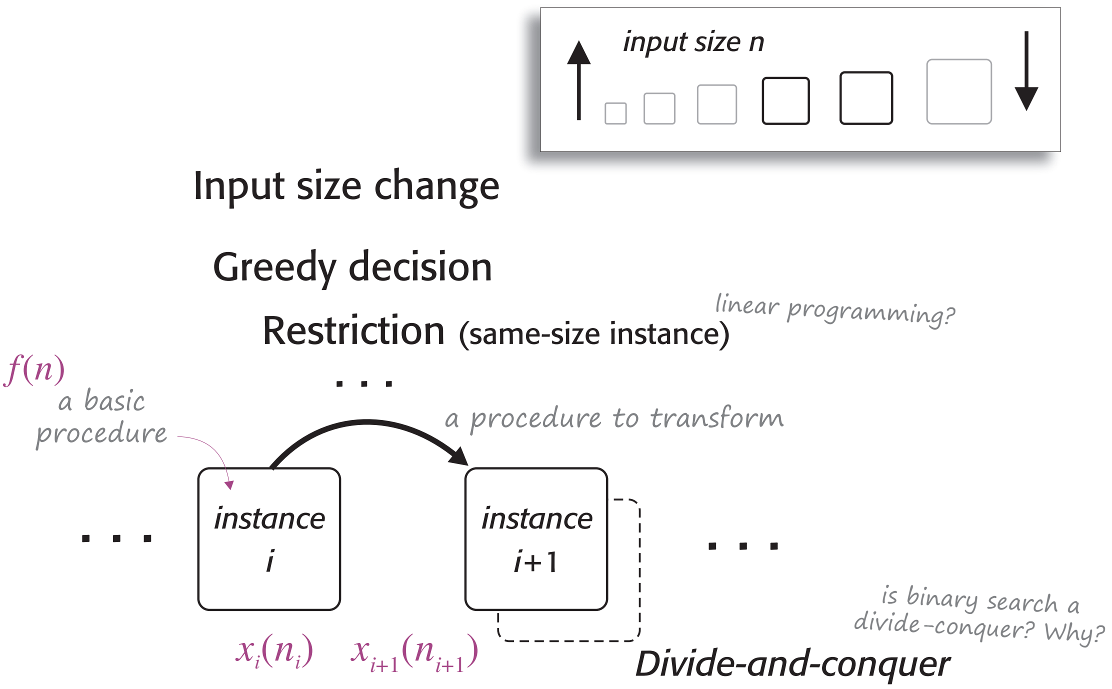
Sequence of Instances
A General View

Quiz
Compare instance size reduction/change pattern in insertion sort and binary search.

Exercise
Suggest example algorithms for each case.

For example, in insertion sorting, inserting a key in a sorted list kept right (high indices) or left (previous examples) of the key.

Exercise
Design a nonrecursive algorithm which keeps the sorted sublist near the end of the array (to the right of *v*). Print list after each round, compare to figures.



cs681fig5-8.cdr
Monday, February 26, 2024 12:22:12 PM
Color profile: Disabled

Composite Default screen

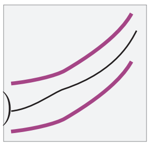
Project 1 Discussion

Another Useful Math Tool
Using Limits

A math limit can investigate relative growth behavior as $n \to \infty$.

Efficiency class of $t(n)$?

$$\lim_{n\to\infty} \frac{t(n)}{g(n)} = \begin{cases} 0 \\ c > 0 \\ \infty \end{cases}$$



Who goes faster to ∞? It must eventually (**asymptotically**) be above.

Exercise
Use Excel to compare growth of functions from exercise Slide 1–11. For example: the truth of $n^3 \in \Omega(n^2)$.

don't care about small n

n	? $t(n)$: $\frac{1}{4}n^2 + 5$	$g(n)$: n^2	$\lim_{n\to\infty} \frac{t(n)}{g(n)}$
1	5.25	1.00	5.25000
2	6.00	4.00	1.50000
3	7.25	9.00	0.80556
4	9.00	16.00	0.56250
5	11.25	25.00	0.45000
6	14.00	36.00	0.38889
7	17.25	49.00	0.35204
8	21.00	64.00	0.32813
…			
97	2357.25	9409.00	0.25053
98	2406.00	9604.00	0.25052
99	2455.25	9801.00	0.25051
100	2505.00	10000.00	0.25050
…			
447	49957.25	199809.00	0.25003
448	50181.00	200704.00	0.25002
449	50405.25	201601.00	0.25002
1000	250005.00	1000000.00	0.25001
…			

Algorithm ≡ steps to result

- ✎ Ordered
- ✎ Unambiguous
- ✎ Computable
- ✎ Terminating

Review
Proper Pseudocode

An arbitrary mix of natural language, math, and programming-like expressions.

Pseudocode should not lead to wrong results due to mis-statement.

➪ Statement vs. correctness

Input specs typically essential in how steps are stated.

Components of statement

➪ Specify legal input instances

❶ Inputs, in terms of parameters used in steps, ❷ limitations ❸ or preconditions

A search may return true/false (or 0/1) or an index in an array, or a pointer in a linked list or the found item itself.

➪ Specify expected results

❶ Output items or effects, ❷ specs (how) if needed

User must be able to unambiguously follow intended steps even if the algorithm is flawed.

➪ Elucidate structure to clarify logic

Iterations, conditionals, indent to show nesting

Review - Pseudocode Examples

Algorithm *InsertionSort*
Input Array $A[0..n-1]$ of orderbale keys
Output Sorted array

```
for i ← 1 to n − 1 do
    v ← A[i]
    j ← i − 1
    while j ≥ 0 and A[j] > v do
        A[j + 1] ← A[j]
        j ← j − 1
    A[j + 1] ← v
```

```
for i ← 1 to n − 1 do
    v ← A[i]
    j ← i − 1
    while j ≥ 0 and A[j] > v do
        A[j + 1] ← A[j]
        j ← j − 1
    end while
    A[j + 1] ← v
end for
```

Algorithm 1 Insertion Sort (Levitan, 3rd)
Input Array of n orderbale keys $A[0..n-1]$
Output Sorted array

```
for i ← 1 to n − 1 do
    k ← A[i]
    j ← i − 1
    while j ≥ 0 and A[j] > k do
        A[j + 1] ← A[j]
        j ← j − 1
    A[j + 1] ← k
```

Algorithm *InsertionSortRec*
Input Array $A[0..n-1]$ of orderbale keys
▶ **Input** Insert index i, initially $n-1$
Output Sorted array

```
if i > 1 then
    InsertionSortRec (A, i − 1)
v ← A[i]
j ← i − 1
while j ≥ 0 and A[j] > v do
    A[j + 1] ← A[j]
    j ← j − 1
A[j + 1] ← v
```

Algorithm 3 Insertion Sort (Recursive)
Input Array of n orderbale keys $A[0..n-1]$
Output Sorted array

▶ **call** $ins(A, n-1)$

```
procedure ins(A, i)                    ▷ i: insert index
if i > 1 then
    ins(A, i − 1)
k ← A[i]
j ← i − 1
while j ≥ 0 and A[j] > k do
    A[j + 1] ← A[j]
    j ← j − 1
A[j + 1] ← k
```

Mergesort

Exercise
Trace in your mind lists of 2 and 3 keys (check element indices of A,B,C).

- ✎ <u>Divide</u> problem into smaller instances
- ✎ Apply solution independently to smaller instances
- ✎ Construct problem solution from solutions to smaller instances

Algorithm *Mergesort*
Input … $A[0\mathinner{..}n-1]$ …
Output …

if $n > 1$ **then**
  copy $A[0\mathinner{..}\lfloor n/2\rfloor - 1]$ to $B[0\mathinner{..}\lfloor n/2\rfloor - 1]$
  copy $A[\lfloor n/2\rfloor\mathinner{..}n-1]$ to $C[0\mathinner{..}\lfloor n/2\rfloor - 1]$
  $Mergesort(B[0\mathinner{..}\lfloor n/2\rfloor - 1])$
  $Mergesort(C[0\mathinner{..}\lfloor n/2\rfloor - 1])$
  $Merge(B, C, A)$

Mergesort Example

➪ Backtracking phase

Levitan, 3rd

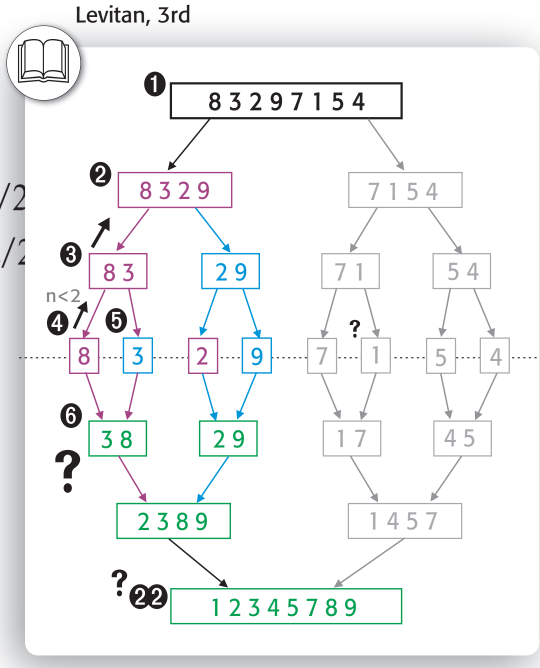
Exercise
List calls in steps 4–6, show input arrays for each (i.e, **serialize** figure), note *list reduction sequence*. **Hint**: print an operation log to study (note recursive call to n<2 instance triggers back-tracking phase).

To visualize backtracking, fold figure along the dotted line to overlay the green box #6 on the magenta box #3.

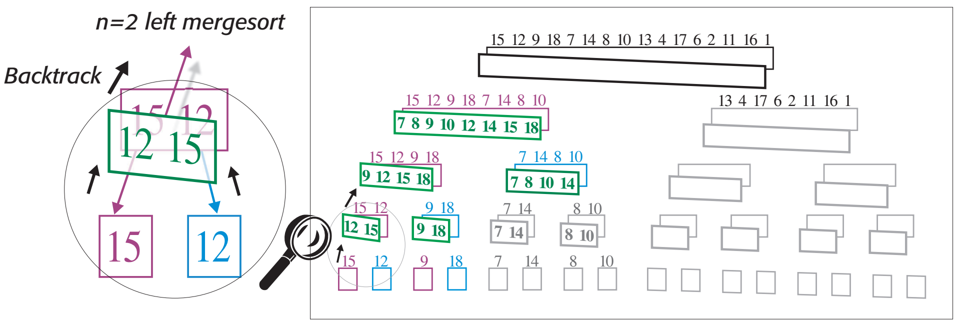
Quiz
How many extra array elements were allocated by *mergesort* during the expansion phase of the recursion? What if n=16?

Algorithm *Mergesort*

1: **if** $n > 1$ **then**
2: copy $A[0 .. \lfloor n/2 \rfloor - 1]$ to $B[0 .. \lfloor n/2$
3: copy $A[\lfloor n/2 \rfloor .. n-1]$ to $C[0 .. \lfloor n/2$
4: $Mergesort(B[0 .. \lfloor n/2 \rfloor - 1])$
5: $Mergesort(C[0 .. \lfloor n/2 \rfloor - 1])$
6: $Merge(B, C, A)$



Mergesort Practice



Algorithm *Mergesort*

1: **if** $n > 1$ **then**

2: copy $A[0 .. \lfloor n/2 \rfloor - 1]$ to $B[0 .. \lfloor n/2 \rfloor - 1]$

3: copy $A[\lfloor n/2 \rfloor .. n-1]$ to $C[0 .. \lfloor n/2 \rfloor - 1]$

4: *Mergesort*$(B[0 .. \lfloor n/2 \rfloor - 1])$

5: *Mergesort*$(C[0 .. \lfloor n/2 \rfloor - 1])$

6: *Merge*(B, C, A)