CENG 218
Design and Analysis of Algorithms

Izmir Institute of Technology

Lecture 1: Introduction

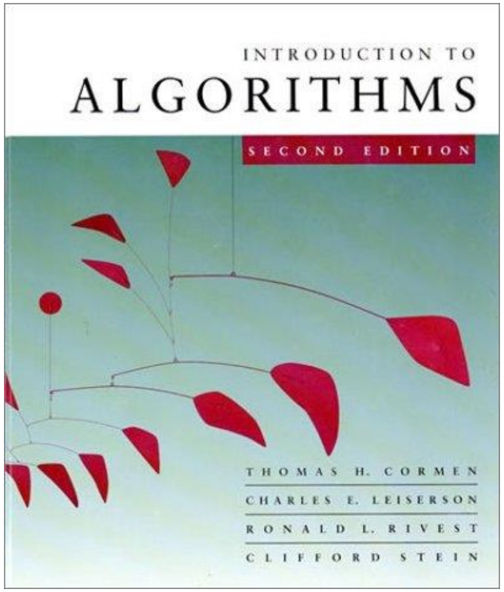
Slides were mostly prepared using the material provided by Prof. Charles E. Leiserson and Prof. Erik Demaine from MIT

Textbook

Cormen, T.H., Leiserson, C.E., Rivest, R.L. & Stein, C. *Introduction to Algorithms*, 3rd Ed., MIT Press.

– 3rd Ed. is available in our library as eBook and downloadable chapters.

– 2nd Ed. is available in our library as hardcopy.



Video lectures for the textbook can be viewed at and downloaded from:

http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-046j-introduction-to-algorithms-sma-5503-fall-2005/

Course Activities and Grading

- **~2 Homeworks** (15% total). You can help each other but you are not allowed to copy the homeworks. Teaching assistants **Altuğ Yiğit** and **N. Furkan Pala** will grade the assignments.
- **Course Material** (Slides, assignments and grades) will be posted via MS-Teams.
- **One Midterm Exam** (35%).
- **One Final Exam** (40%).
- **~5 Unannounced Quizzes** (10%).
  Each student’s worst quiz will be discarded.

Design and Analysis of Algorithms

- ***Design:*** Design algorithms which minimize the cost.
- ***Analysis:*** Predict the cost of an algorithm in terms of resources and performance (computation time).
  - In this course, emphasis is on performance.
  - What may be more important than performance?
    - correctness
    - simplicity
    - maintainability (programmer time)
    - robustness
    - functionality (providing more features)
    - security

Why study performance of algorithms?

- Algorithms help us to understand *scalability*.
- Performance often draws the line between what is feasible and what is impossible.
- Algorithmic mathematics provides a *language* for talking about program behavior.
- The lessons of program performance generalize to other computing resources like memory, communication etc.
- Useful for daily life!

The problem of sorting

Input: sequence $\langle a_1, a_2, \ldots, a_n \rangle$ of numbers.

Output: rearrangement $\langle a'_1, a'_2, \ldots, a'_n \rangle$ such that $a'_1 \leq a'_2 \leq \cdots \leq a'_n$.

Example:

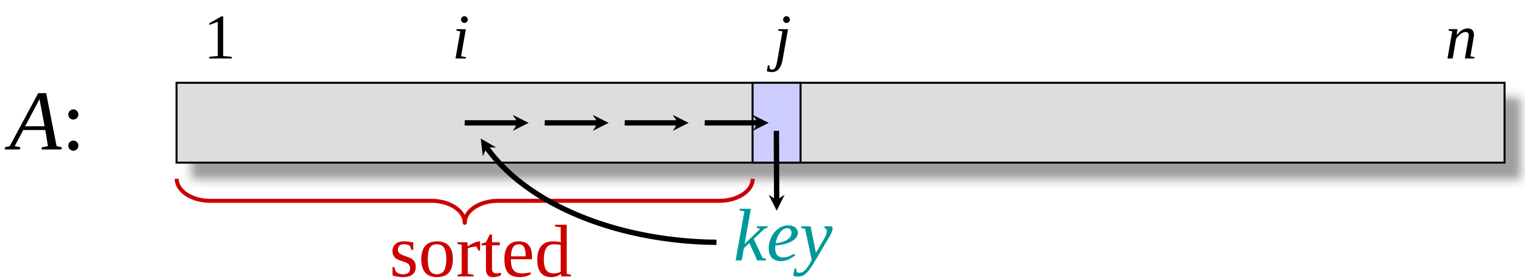
Input: 8 2 4 9 3 6

Output: 2 3 4 6 8 9

Insertion sort

"pseudocode"

```
INSERTION-SORT (A, n)    ⊳ A[1 . . n]
for j ← 2 to n
begin
    key ← A[ j]
    i ← j – 1
    while i > 0 and A[i] > key
    begin
        A[i+1] ← A[i]
        i ← i – 1
    end
    A[i+1] = key
end
```



Example of insertion sort

8	2	4	9	3	6	
2	8	4	9	3	6	
2	4	8	9	3	6	
2	4	8	9	3	6	
2	3	4	8	9	6	
2	3	4	6	8	9	*done*

Running time

- We parameterize the running time by the size of the input, since short sequences are easier to sort than long ones.

  $T_A(n)$ = time of algorithm A on length n inputs
- We generally seek upper bounds on the running time, to have a guarantee of performance.
- Running time also depends on the input itself: an already sorted sequence is easier to sort. (best case.. worst case.. next slide)

Kinds of analyses

Worst-case: (usually)

- $T(n)$ = maximum time of algorithm on any input of size n.

Average-case: (sometimes)

- $T(n)$ = expected time of algorithm over all inputs of size n.
- Need assumption of statistical distribution of inputs. Examples?

Best-case: (NEVER)

- A slow algorithm may work fast on *some* input.

Machine-independent time

Q. *What is insertion sort's worst-case time?*

A. It depends on the speed of our computer:

- relative speed (on the same machine),
- absolute speed (on different machines).

BIG IDEA: "Asymptotic Analysis"

- Ignore machine dependent constants, otherwise impossible to compare algorithms.
- Look at ***growth*** of $T(n)$ as $n \to \infty$.

Our machine model

Random Access Machine (RAM)

- Executes operations sequentially, with no concurrent operations.
- Uses a set of primitive operations:
  - Arithmetic, Logical, Comparisons, etc.
- Simplifying assumption: All operations cost 1 unit.

Θ-notation (Asymptotic Analysis)

Engineering way of thinking:

- Drop low-order terms. Why?
- Ignore leading constants. Why?
- Example: $T(n) = 3n^3 + 90n^2 - 5n + 6046 = \Theta(n^3)$

Mathematical definition:
(*will see this in Lecture* 2)

$f(n) = \Theta(g(n))$: There exist positive constants c_1, c_2, and n_0 such that $c_1 g(n) \le f(n) \le c_2 g(n)$ for all $n \ge n_0$

Insertion sort analysis

INSERTION-SORT (A, n)		
for $j \leftarrow 2$ **to** n	c_1	n-1 times
begin		
$key \leftarrow A[\,j]$	c_2	n-1 times
$i \leftarrow j - 1$	c_3	n-1 times
while $i > 0$ and $A[i] > key$	c_4	
begin		
$A[i+1] \leftarrow A[i]$	c_5	$\sum_{j=2}^{n} j$ times
$i \leftarrow i - 1$	c_6	
end		
$A[i+1] = key$	c_7	n-1 times
end		

Insertion sort analysis

$$T(n) = (c_1 + c_2 + c_3)(n-1) + (c_4 + c_5 + c_6)\sum_{j=2}^{n} j + c_7(n-1)$$

Worst case:
Input is in reverse order.

$$T(n) = \Theta(n) + \Theta(n^2) + \Theta(n) = \Theta(n^2)$$

$$\sum_{j=2}^{n} j = \frac{n(n+1)}{2} - 1 = \Theta(n^2)$$

[arithmetic series]

Insertion sort analysis

$$T(n) = (c_1 + c_2 + c_3)(n-1) + c_4(n-1) + c_7(n-1)$$

Best case:

Input is already in right order. While loop does not turn at all.

(But the logical comparison at the begining of the while loop (c_4) runs).

$$T(n) = \Theta(n) + \Theta(n) + \Theta(n) = \Theta(n)$$

Insertion sort analysis

Average case: All permutations are equally likely.

For each j, the subarray $A[1... j\text{-}1]$ is checked.

On average, half of the elements ($j/2$) are checked.

$$T(n) = \sum_{j=2}^{n} \frac{j}{2} = \Theta(n^2)$$

Is insertion sort a fast sorting algorithm?

- Moderately so, for small n.
- Not at all, for large n.

Example 2: Searching

Problem of *searching an ordered list*.

- Given a sorted list of n elements.
- And given a particular element x,
- Determine whether x appears in the list,
- and if so, return its index (position) in the list.

Linear Search

LINEARSEARCH (x: integer, $A[1\,..\,n]$: sorted list)
$i \leftarrow 1$
while ($i \leq n$ and $x \neq A[i]$)
  $i \leftarrow i + 1$
if $i \leq n$ **then** *location* $\leftarrow i$
else *location* $\leftarrow 0$
{*location* is the index of the term equal to x
or 0 if x is not found in the list}

Linear Search Analysis

LINEARSEARCH (x: integer, $A[1 \,.\,.\, n]$: sorted list)

$i \leftarrow 1$	c_1
while ($i \leq n$ and $x \neq A[i]$)	c_2
$i \leftarrow i + 1$	c_3
if $i \leq n$ **then** $location \leftarrow i$	c_4
else $location \leftarrow 0$	c_5

What are the *worst-case*, *best-case and average-case* analyses?

Linear Search Analysis

Worst case:

$$T(n) = c_1 + \left(\sum_{i=1}^{n} (c_2 + c_3) \right) + c_4 + c_5 \text{ is } \Theta(?)$$

$$T(n) \text{ is } \Theta(1) + \Theta(n) + \Theta(1) + \Theta(1) = \Theta(n)$$

Best case: The searched element in the first one.

$$T(n) = c_1 + c_2 + c_4 \text{ is } \Theta(1)$$

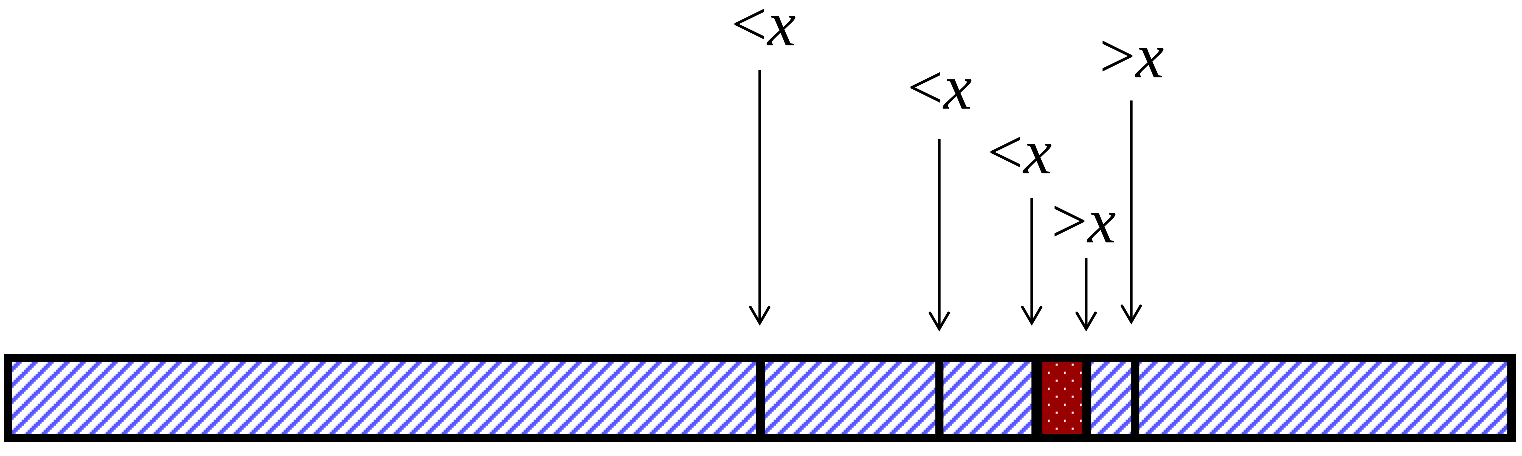
Average case: Finding the element in the middle.

$$T(n) = c_1 + \left(\sum_{i=1}^{n/2} (c_2 + c_3) \right) + c_4 + c_5 \text{ is } \Theta(n)$$

Algorithmic Thinking

Searching problem: Linear search starts from left and checks one by one. Time complexity is $\Theta(n)$.

Better approach (*Binary Search*): At each step, look at the *middle term* of the remaining list to eliminate half of it.



Binary Search

BINARYSEARCH (x: integer, $A[1\,..\,n]$: sorted list)

$\left.\begin{array}{l} i \leftarrow 1 \\ j \leftarrow n \end{array}\right\} \Theta(1)$

Key question:

How many loop iterations?

while $i<j$ **begin**

$\left.\begin{array}{l} m \leftarrow \lfloor (i+j)/2 \rfloor \\ \textbf{if } x>A[m] \textbf{ then } i \leftarrow m+1 \textbf{ else } j \leftarrow m \end{array}\right\} \Theta(1)$

end

$\left.\begin{array}{l} \textbf{if } x = A[i] \textbf{ then } location \leftarrow i \\ \quad \textbf{else } location \leftarrow 0 \end{array}\right\} \Theta(1)$

Binary Search Analysis

- Suppose $n=2^k$.
- Original list from $i=1$ to $j=n$ contains n elements.
- Each iteration: Size $j-i+1$ of range is cut in half.
- Loop terminates when size of range is $1=2^0$ ($i=j$).
- Therefore, number of iterations is $= k = \log_2 n$.
- Complexity ≡ # of iterations: $O(\log_2 n) = O(\log n)$.
- Even for $n \neq 2^k$ (not an integral power of 2),
  time complexity is still $O(\log_2 n) = O(\log n)$.

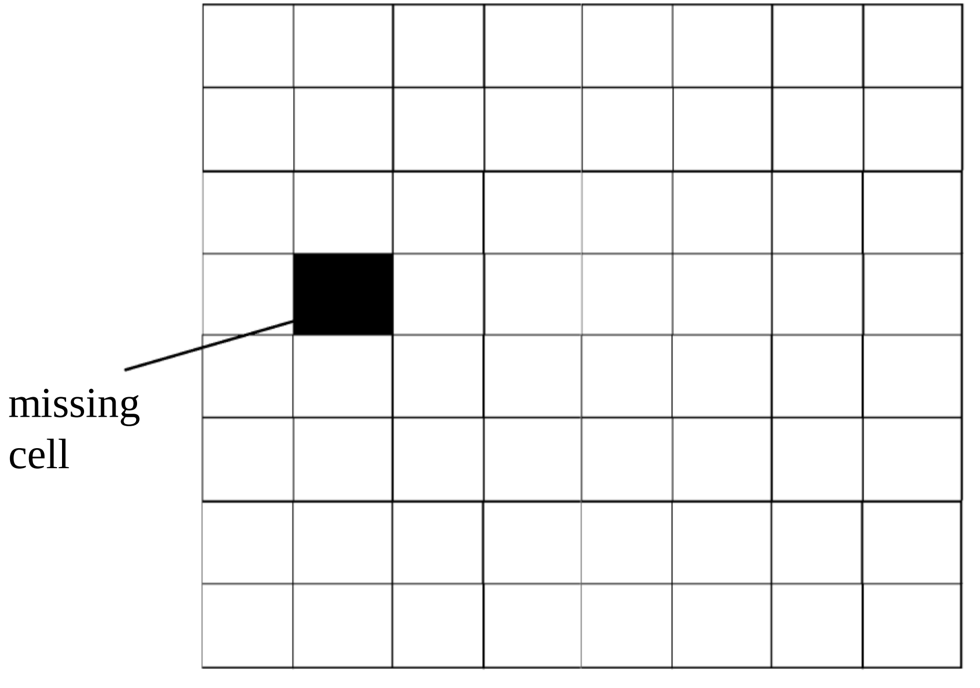
Algorithmic Thinking

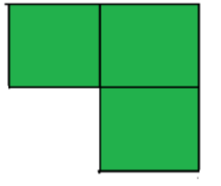
Throwing eggs from a building.

Suppose that you have an N-storey building and plenty of eggs. Suppose also that an egg is broken if it is thrown off floor F or higher, and unbroken otherwise. How many steps does it take to find F?

Algorithmic Thinking

L shaped tiling: Given a nxn board where n is 2^k .
The board has one missing cell (of size 1x1).
Fill the board using L shaped tiles. An L shaped tile is a 2x2 square with one 1x1 cell is missing.





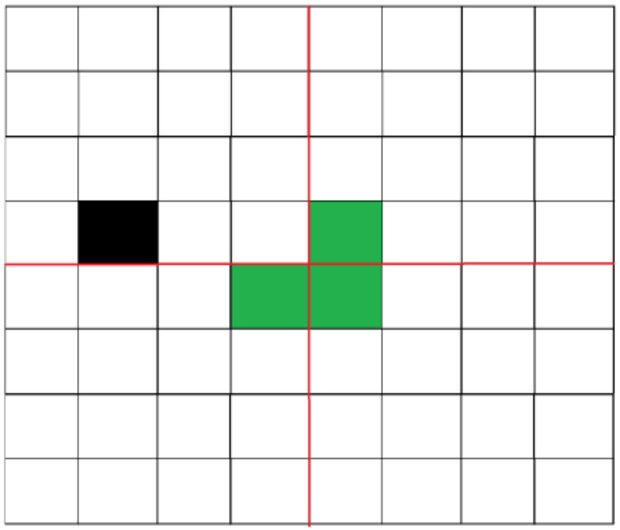
L shaped tile

Algorithmic Thinking

L shaped tiling: Algorithm steps:

1) Base case: *n* = 2, A 2x2 square with one cell missing is nothing but a tile and can be filled with a single tile.

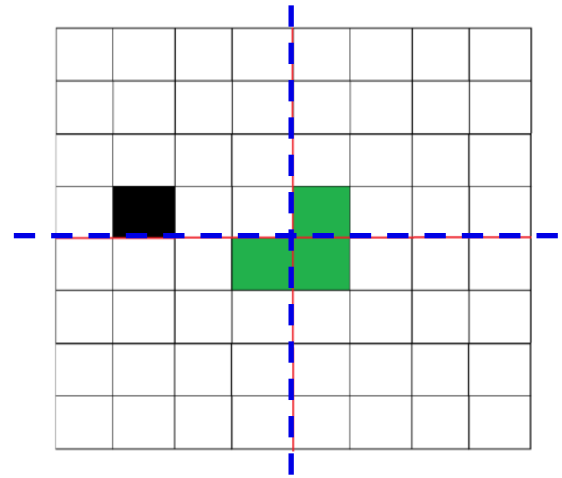
2) Place a L shaped tile at the center such that it does not cover the *n*/2 x *n*/2 subsquare that has a missing square.
Now all four subsquares of size *n*/2 x *n*/2 have a missing cell.

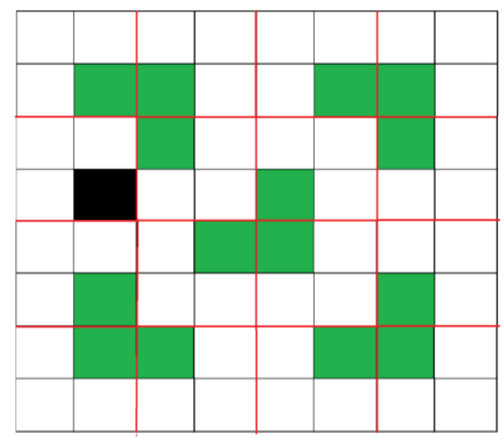


Algorithmic Thinking

L shaped tiling: Algorithm steps:

3) Solve the problem recursively for following four $n/2$ x $n/2$ squares.





Divide & Conquer

Algorithmic thinking examples so far belong to a strategic approach called 'Divide and Conquer'.

Divide the problem into several subproblems.

Conquer the subproblems, solve them recursively.

Combine the solutions of subproblems.

- Binary search: Divide into one subproblem.
- Throwing eggs: Divide into one subproblem.
- L shaped tiling: Divide into four subproblems.