ILLINOIS INSTITUTE OF TECHNOLOGY

Transforming Lives. Inventing the Future. www.iit.edu

CS430-02/03/04

Spring 2023

Introduction to Algorithms

Lec 1

Instructor: Dr. Lan Yao

Agenda

Course Overview

- Review on Syllabus, BB System
- Design Techniques/Approaches
- Analysis – runtime, memory

People

Instructor: Lan Yao

TAs:

Kayenat Patil kpatil7@hawk.iit.edu

Sharandeep Singh ssingh136@hawk.iit.edu

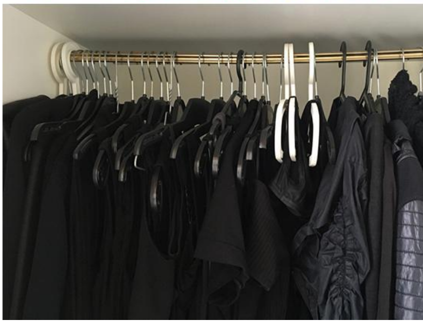
SI:

Shashank Parameswaran sparameswaran@hawk.iit.edu

Sections

- **Live Lecture: CS430-02/03/04**
- **Recitation Sections:**
  5:00-5:50/6:25-7:15 pm F

Design Techniques/Approaches


- An algorithm is any well-defined computational procedure that takes some value or set of values as input and produces some value or set of values as output. It is a sequence of computational steps that transform the **input** into the **output**.

Try This!

Design an algorithm to return the maximum element of an unsorted array.

Design an algorithm to return the maximum and minimum elements of an unsorted array.

Sorting*

General Problem Definition:

input: sequence of items <a1, a2, a3, …, an>

output: permutation <a1', a2', a3', . . . , an'>
such that a1' <= a2' <= a3' <= . . . <= an'

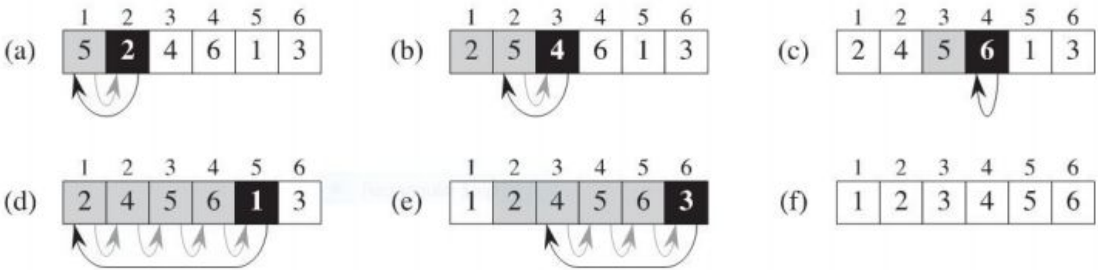
Permutations

For example 5! = 120

5 slots for numbers, 5 choices for 1st slot,

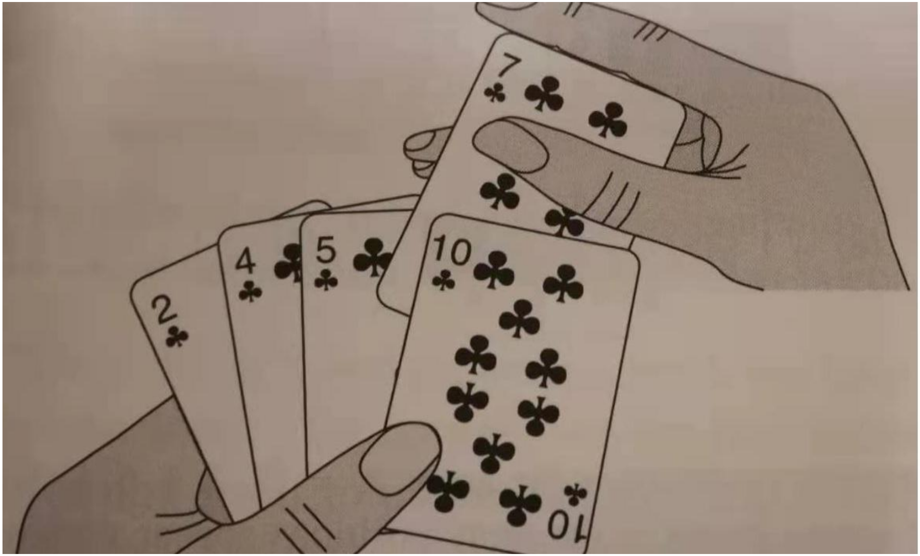
4 choices left for 2nd slot, etc.

Insertion Sort



iterratively insert keys in sorted arrays

An Instance of Insertion



Insertion Sort Pseudo-code

```
INSERTION-SORT(A)
for j ← 2 to length[A]
    key ← A[j]
    //Insert A[j] into sorted sequence A[1 ..j - 1]
    i ← j - 1
    while i > 0 and A[i] > key
        A[i + 1] ← A[i]
        i ← i - 1
    A[i + 1] ← key
```

Analysis of Algorithms – summarize behavior*

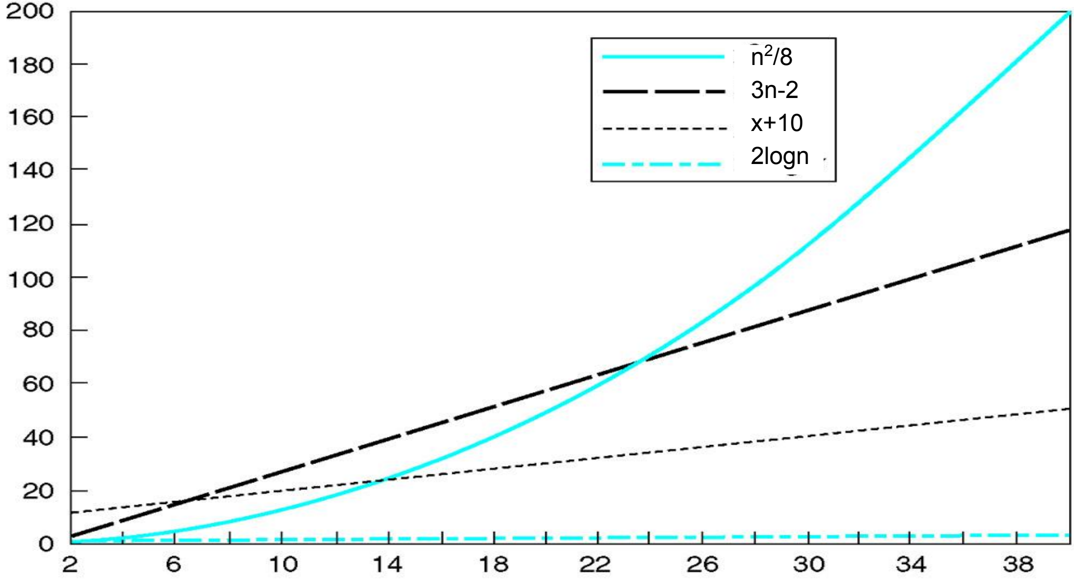
Run-Time Analysis for Sorting –
Depends on input size & input itself

Kinds of Analysis

- worst case (usually used)
- average case (sometimes used)
- best case (hardly ever used)

Analysis – runtime, memory

time or memory*



size of problem "n"

Individual points on the graph are irrelevant, only the growth of the function matters

Run-time Analysis*

Functional Analysis

- ignore machine dependent constants
- look at the growth of $T(n)$ as n-> infinity
- as you double n, what does $T(n)$ do?? double?? square??

Runtime Analysis Approaches*

For iterative algorithms

- count the number of times each statement is executed
- define constants for the execution time of various types of statements
- develop a function describing the runtime as a function of the problem size.

For recursive algorithms, develop and solve a recurrence relation--later

Insertion Sort Runtime Analysis

INSERTION-SORT(A)	*cost*
for $j = 2$ **to** $A.length$	c_1
$key = A[j]$	c_2
// Insert $A[j]$ into the sorted sequence $A[1..j-1]$.	0
$i = j - 1$	c_4
while $i > 0$ and $A[i] > key$	c_5
$A[i+1] = A[i]$	c_6
$i = i - 1$	c_7
$A[i+1] = key$	c_8

	INSERTION-SORT(A)	*cost*	*times*
1	**for** $j = 2$ **to** $A.length$	c_1	n
2	$key = A[j]$	c_2	$n-1$
3	// Insert $A[j]$ into the sorted sequence $A[1 \,..\, j-1]$.	0	$n-1$
4	$i = j - 1$	c_4	$n-1$
5	**while** $i > 0$ and $A[i] > key$	c_5	$\sum_{j=2}^{n} t_j$
6	$A[i+1] = A[i]$	c_6	$\sum_{j=2}^{n} (t_j - 1)$
7	$i = i - 1$	c_7	$\sum_{j=2}^{n} (t_j - 1)$
8	$A[i+1] = key$	c_8	$n-1$

let t_j be the number of times the **while** loop test in line 5 is executed for that value of j

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^{n} t_j + c_6 \sum_{j=2}^{n} (t_j - 1) + c_7 \sum_{j=2}^{n} (t_j - 1) + c_8(n-1)$$

Best Case: the array has been sorted, then

$t_j = 1$ for all j

$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5(n-1) + c_8(n-1)$

$= (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8)$

$= an + b$ -- Linear Growth Function