Note to the Students:

This lecture note is prepared by refering several textbooks and online resources. I suggest you to use this note only for the reference for the topics covered in the class. If you find any mistakes Please let me know (Put it on discussion form ; canvas)

I also suggest you to read atleast one standard textbook on this course for better understanding of the subject.

Introduction

Think about the following two questions.

Q Why study about Algorithms ?

Q What do we learn in this course ?

Q Why Study Algorithms ?

- Lot of real/practical applications
- Increases Problem solving and logical skills (Algorithmic thinking)
- improves Programming Skills.
- IMP In many interviews you will likely be questioned on algorithms.

Q What do we learn in this course ?

"Design and Analysis of algorithms".

→ Mathematical analysis

Design of fast algorithms

↓ Study some design techniques

An Algorithm is any well-defined Computational Procedure that takes some value or set of values as input and produce some value or set of values as output.

- Fast
- Correct solution
- On all inputs.

} defined later

High level overview of the course

We want to design efficient (fast) algorithms for Computational Problems.

→ speed: how long an algorithm takes to produce its result.

There may be many ways to solve the same problem.

We want to find the one that is "best".

At some point we have to give up !!!

Next we look at some problems and see what kind of questions we can ask.

Sorting Problem

Input: A sequence of n elements $a_1, a_2, \ldots a_n$

output: A permutation $a_1', a_2' \ldots a_n'$ of input sequence such that $a_1' \leq a_2' \leq \ldots \leq a_n'$.

Think about your favourite sorting algorithm ?

Popular Sorting algorithms

Bubble Sort

Insertion Sort

Selection Sort

Merge Sort

Quick Sort

- - - - - - So on.

Q: Which one is "good" or "better" ?

↓

defined later.

Ans: depends on several factors like,

① Number of elements to be sorted

② Possible restrictions on the elements values

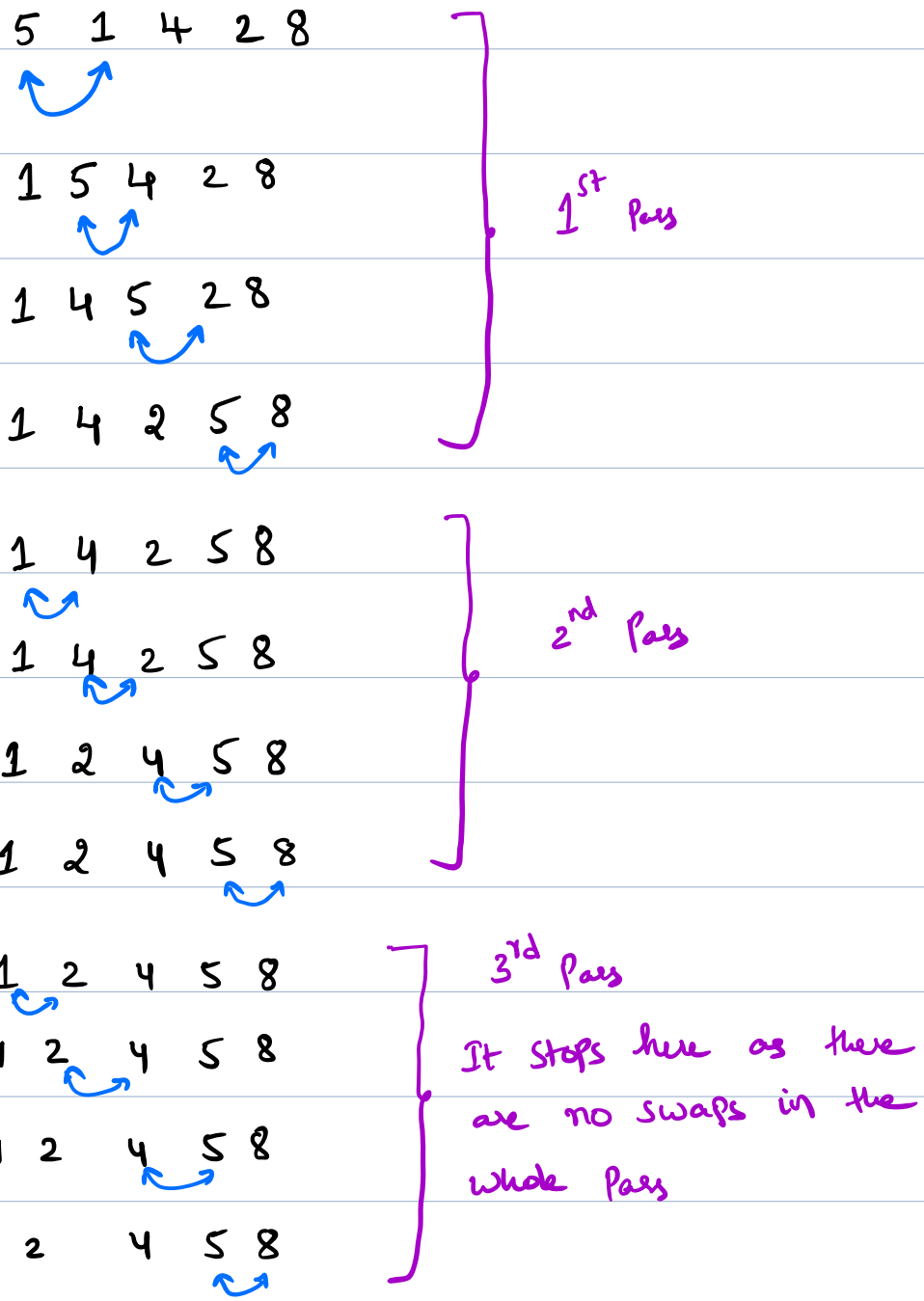
③ the extent to which elements are already somewhat sorted.

So on.

Bubble Sort

It works by repeatedly swapping the adjacent elements if they are in wrong order.

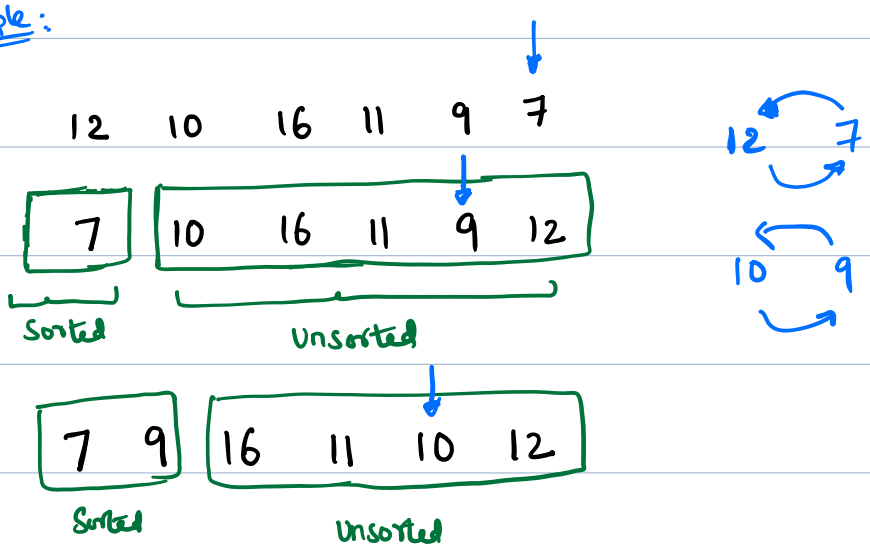
Eg:



Selection Sort:

The selection sort algorithm sorts an array by repeatedly finding the minimum element from unsorted part and swaps it with the value in the first position.

Example:



By repeating the above steps, finally we get

Few key Questions

- Which of the above two algorithms is better/good ?
- Do they terminate ?
- Does they produce sorted list on any input list

Q Given any input array, does the above algorithm always gives the Sorted array as the output ?

"Correctness of the algorithm" ↓ Proved later

Formally

An algorithm is said to be Correct if for every input instance, it stops & gives the Correct output.

Pseudocode

It is an informal way of description of steps in an algorithm. It is intended for human reading rather than machine reading.

GCD of two numbers

I/p: $a, b \in \mathbb{Z}^+$, WLOG $a > b$

O/p: $gcd(a, b)$ (largest integer that divides both a & b)

Algorithm 1 (School level) [Laborious algorithm]

1. Find all Prime factors of a : ie, $a = a_1^{k_1} a_2^{k_2} \cdots a_i^{k_i}$
2. Find all Prime " " b : ie, $b = b_1^{l_1} b_2^{l_2} \cdots b_j^{l_j}$
3. gcd = Product of the common factors (including repetitions)

Example: $b = 48$, $a = 78$

$48 = 2^4 \cdot 3$

$78 = 2 \times 3 \times 13$

$gcd(48, 78) = 2 \times 3 = 6$

Algorithm 2: Euclid - Pseudo Code

```
Euclid-Gcd (a, b)
    While b ≠ 0
        r = a mod b
        a = b
        b = r
    end
    return a
```

It is an informal way of description of steps in an algorithm. It is intended for human reading rather than machine reading.

Example: b = 48, a = 78

$48 \overline{)78} (1$ (48, 30)

$30 \overline{)48} (1$ (30, 18)

$18 \overline{)30} (1$ (18, 12)

$12 \overline{)30} (2$ (24, 6)

$6 \overline{)12} (2$ (12, 0)

Example: $b = 48, \quad a = 78$

$48 = 2^4 \cdot 3$ ($\geq$ 8 divisions)

$78 = 2 \times 3 \times 13$ (many more)

$\gcd(48, 78) = 2 \times 3 = 6$

We check whether there is any divisor b/w 3 & 13.

Example: $b = 48, \quad a = 78$

$48 \overline{)78} (1$ (48, 30)

$30 \overline{)48} (1$ (30, 18)

$18 \overline{)30} (1$ (18, 12)

$12 \overline{)30} (2$ (24, 6)

$6 \overline{)12} (2$ (12, 0)

(Five divisions)

Correctness of Euclid's algorithm

Lemma: Let $a = bq + r$, where a, b, q and r are integers. Then $\gcd(a,b) = \gcd(b,r)$

(r: $a \bmod b$)

Proof: We show that common divisors of a & b are same as the common divisors of b & r, that shows $\gcd(a,b) = \gcd(b,r)$

Suppose $d|a$ & $d|b$ then $d|a-bq = r$

so d is a common divisor of b & r

Suppose, $d|b$ & $d|r$ then $d|bq+r = a$

so d is a common divisor of a & b

$$\Rightarrow \gcd(a,b) = \gcd(b,r)$$

∴ As the loop executes a & b might change but their gcd is preserved.

Q: Does Euclid algorithm terminate?

b	a
a mod b	b

$a \bmod b < b$

decreases by at least one in each iteration

Next: we want to give an upper bound on # of iterations.

Theorem: If Euclid is called with P and q, ie, Euclid-Gcd(P,q), then if $q < P$ then in each iteration the sum of the values of variables a and b will decrease at least by a factor 3/2.

$$\left[\#\text{ of iterations} \leq \log_{3/2}^{P+q} \right]$$

Proof:- Begining of the iteration, $a = P$, $b = q$

After iteration - 1 : $a = p'$, $b = q'$

$p' \downarrow q$ $\quad$ $q' \downarrow P \bmod q < q$ —②

Goal:-

Estimate $\dfrac{P+q}{p'+q'}$

$p' + q'$

$p' \downarrow$ divisor $\quad$ $q' \downarrow$ reminder when P is divided by q

reminder

—①

divisor + reminder $\leq$ dividend $= P$

$p' + q' \leq p$ (from ①) $\qquad p' + q' \leq 2q$ (from ②)

$$3(p'+q') = 2(p'+q') + (p'+q')$$

$$\leq 2p + 2q$$

$$3(p'+q') \leq 2(p+q), \quad p'+q' \leq \frac{2}{3}(p+q)$$

$$\frac{p+q}{p'+q'} \geq \frac{3}{2}$$

∴ Algorithm terminates after at most $\log_{\frac{3}{2}}^{p+q}$ iterations.

Activity selection Problem

Given a set S of n activities $a_1 \ldots a_n$

s_i = Start time of activity a_i

f_i = finish time of activity a_i

ie., Activity a_i takes Place during the interval $[s_i, f_i)$

Two activities a_i and a_j are Compatible if the intervals $[s_i, f_i)$ and $[s_j, f_j)$ do not overlap.

Find max-size subset A of compatible activities.

[WLOG $f_1 \leq f_2 \ldots \leq f_n$]

Example:

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16

$\{a_1, a_4\}$ are Compatible

$\{a_1, a_3\}$ are Not Compatible.

Some possible Ideas

1. Brute Force
2. Select the activity which takes least time
3. " " " " Starts first
4. etc.

@ What about choosing the activity with earlier finish time?

Does this work?

Again, we need the Proof.

Coin-Change Problem:

Given a value N, we want to make change for N Rs, using the infinite supply of indian currency $\{1, 2, 5, 10, 20, 50, 100, 200, 500, 1000\}$.

What is the minimum # of coins and/or notes needed to make the change?

Example: $N = 121$

$= 100 + 20 + 1$ } so answer is 3.

(Q) Can you think of an algorithm?

(Q) What happens if denominations equals to $\{1, 75, 100\}$ and $N = 150$?

Summary Sofar

Proof of Correctness is important

① Helps us in **design of algorithms**

② we can find error that would be difficult to find with **testing alone**

③ It also enhances **programming abilities as well.**

Analyzing Algorithms

Same Problem can be solved in different ways

For example Sorting, gcd etc.

We are interested in analyzing the Performance of algorithms.

The two obvious Performance measures of an algorithm are

1. Running time (Formal defn later)
2. Memory / Space

Size of an input instance:

Formal: The number of bits needed to represent the input instance.

Eg 1 For GCD Problem input consists of two integers.

36, 48

↓ ↓

6 bits 6 bits = 12 bits

Informal: Any Parameter which grows, roughly with the formal notion of Size.

For the above example, we could say the Size is simply the Sum of numbers.

36 + 48 = 84

↑

Running time - Motivation

The running time of an algorithm depends

- Speed of the Processor
- Amount of memory available
- The OS on the PC
- Programming language / implementation
- The size of the input
- The structure of the input etc

We want to say something about the running time regardless of the impact of the above factors.

① Analyze without implementing (RAM - model)

② Worst case input (Worst case running time)

③ Ignore unnecessary details (Big-O notation)

The RAM Model of Computation

Machine-independent algorithm design depends upon a hypothetical Computer called Random access machine.

In this model of Computation, we deal with a Computer where

- Each Simple Operation such as +, −, *, if, assigning value to a variable etc takes exactly one time step.
- Loops and Subroutines are not considered Simple operations. They are Composition of many Simple step operations.
- Each memory access takes one time step. Further, we have as much memory as we need.

Under RAM model, we measure run time by counting up the number of steps an algorithm takes on a given problem instance.

The RAM model sounds too simple of how computer perform, and yet it provides an excellent model for understanding how an algorithm will perform on a real computer.

The running time of an algorithm on a particular input is the number of operations or "steps" executed.

(machine-independent)

Assume that a constant amount of time is required to execute each line of our algorithm.

One line may take a different amount of time than another line.

We assume that i^{th} each execution of i^{th} line takes time c_i, where c_i is constant.

Examples:

		Cost	times
	Sum-array (A, n)		
1	Sum = 0,	c_1	1
2	i = 1	c_2	1
3	While i <= n	c_3	n+1
4	Sum = Sum + A[i]	c_4	n
5	i = i+1	c_5	n
6	Return Sum	c_6	1

Total cost or running time

$$= c_1 + c_2 + c_3(n+1) + c_4 n + c_5 n + c_6$$

$$= (c_1 + c_2 + c_6 + c_3) + n(c_3 + c_4 + c_5)$$

Notation: $T(n)$: The running time of an algorithm

in the **worst case** on input of size n.

↳ longest running time for any input of size n.

Search (A, n, k)

1	$i = 1$	c_1	1
2	While $i <= n$	c_2	?
3	If $A[i] == k$	c_3	?
4	return Found	c_4	1
5	$i = i + 1$	c_5	1
6	return Not found	c_6	1

Best and Worst Case Complexity

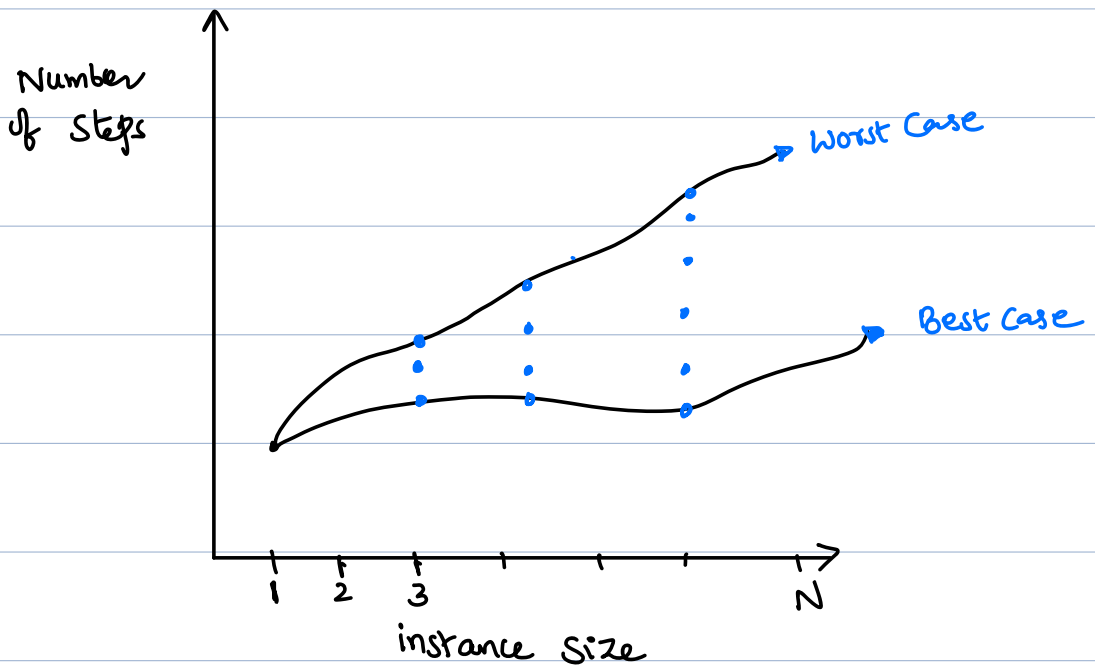
Using RAM model of Computation, we can Count how many steps our algorithm takes on any given input instance by executing it.

However, to understand how good or bad an algorithm is in general, we must know how it works over all instances.

Worst-Case Complexity of the algorithm is the function defined by the MAXIMUM number of steps taken in any instance of size n.

Best-Case Complexity: of the algorithm is the function defined by the MINIMUM number of steps taken in any instance of size n.

[see the figure in next Page]


Number of Steps
Worst Case
Best Case
1
2
3
N
instance Size

Describing Algorithms

Usually a Complete description of any algorithm has four Components.

- **What:** A precise specification of the problem that the algorithm solves.
- **How:** A precise description of the algorithm itself. (Pseudocode & Plain English)
- **Why:** A proof that the algorithm solves the problem it is supposed to solve.
- **How fast:** An analysis of the running time of the algorithm.

Running time

Correctness

Summary so far

1. Running time depends on the input size
2. Measure time efficiency as a function of input size.
3. Different inputs of same size can take a different time.

In Next Class,

We will do analysis of algorithms more formally

Prerequisite

I assume some familiarity with basic Proof techniques like induction, Contradiction.

Familiar with any Programming language

Books:

1. Introduction to Algorithms: Coremen et al.
2. Algorithm Design: Kleinberg and Tardos
3. Algorithms: S. Dasgupta, Papadimitriou, U.V. Vazirani

Various free online resources

1. MIT Course on algorithms: Youtube
2. Coursera: Tim Roughgarden
3. NPTEL

Evaluation method:

Home works – 20% [Theory + Programming]

Tierce exams – 80% [2 Exams, 40% each]

100%