SLIIT

Discover Your Future

IT2070 – Data Structures and Algorithms

Lecture 06

Introduction to Algorithms

U. U. Samantha Rajapaksha

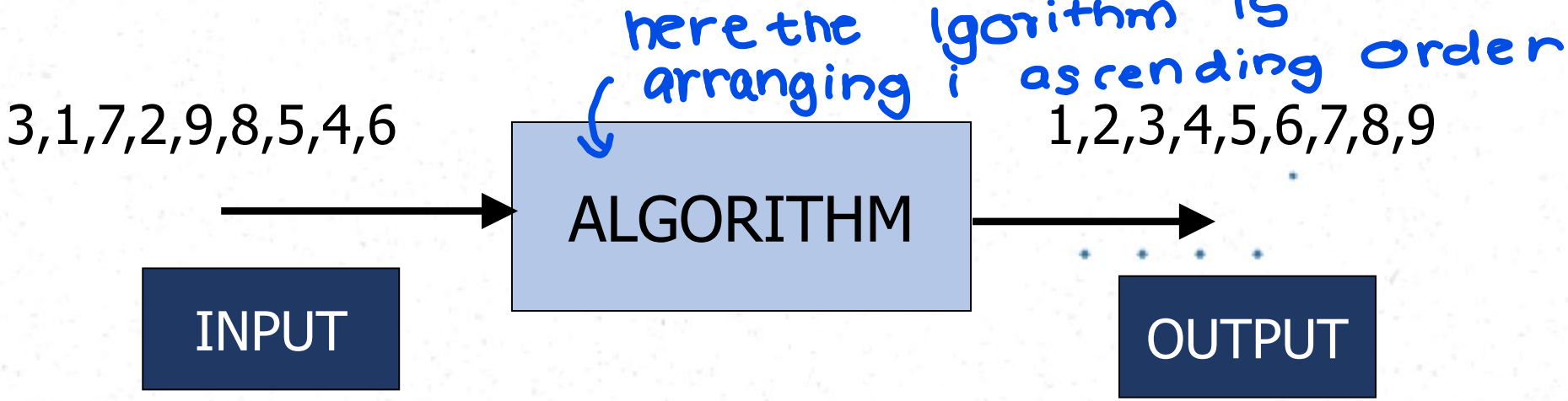
M.Sc.in IT, B.Sc.(Engineering) University of Moratuwa

Senior Lecturer SLIIT

Samantha.r@slit.lk

ALGORITHMS

- 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.



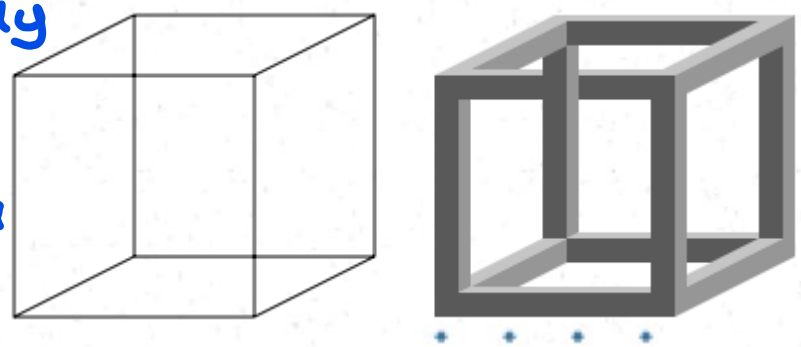
ALGORITHM (Contd.)

Types of operations that can be considered as algorithms

1. Get the smallest value from the input.
2. Remove it and output.
3. Repeat above 1,2 for remaining input until there is no item in the input.

Properties of an Algorithm.

- Be correct.
- Be unambiguous. → it can't have multiple meaning straight to the point
- Give the correct solution for all cases.
- Be simple. → no unnecessarily complicated
- It must terminate. → should have a end



Necker_cube_and_impossible_cube

Source:http://en.wikipedia.org/wiki/Ambiguity#Mathematical_interpretation_of_ambiguity

Applications of Algorithms

- Data retrieval
- Network routing → to find the suitable path inside a network
- Sorting → arranging the numbers in array in a order
- Searching
- Shortest paths in a graph

Pseudocode

- Method of writing down a algorithm.
- Easy to read and understand.
- Just like other programming language.

not a pro ramming language
generic flow of our algorithms

- More expressive method.
- Does not concern with the technique of software engineering.

Pseudocode Conventions.

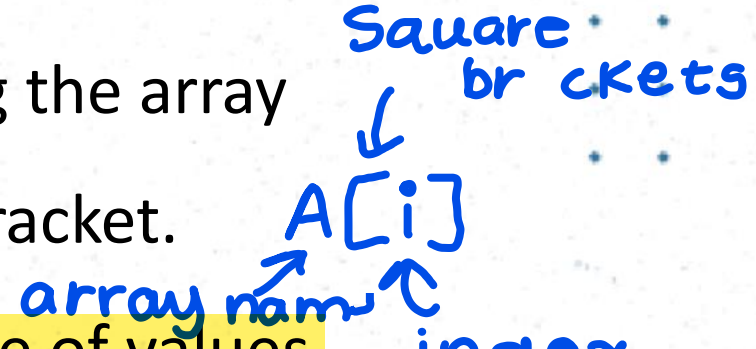
- English.
- Indentation.
- Separate line for each instruction.
- Looping constructs and conditional constructs.
- // indicate a comment line.
- = indicate the assignment.

Pseudocode Conventions.

- Array elements are accessed by specifying the array name followed by the index in the square bracket.
- The notation ".." is used to indicate a range of values within the array.

Ex:

A[1..i] indicates the sub array of A consisting of elements A[1] , A[2] , .. , A[i].



Analysis of Algorithms

we compare algorithms using this

Idea is to predict the resource usage.

*for same problem there might be multiple solutions / algorithms

when resource usage is low algorithm is good

*we should choose the best for our task

- Memory
- Logic Gates
- **Computational Time**

we are more focused to this resource

Why do we need an analysis?

- To compare → between algorithms
- Predict the growth of run time

Worst, Best and Average case.

arrange in ascending order
[10, 3, 5, 2, 9, 1, 3, 8]

best case
input int numbers are already in the ascending order

worst case
input int numbers are in desecending order

Average case
Averagely in ascending and desenaing order

Running time will depend on the chosen instance characteristics.

- **Best case:**
  Minimum number of steps taken on any instance of size n.

example: assume you are asked to earch key value a in this arr
[10, 3, 5, 2, 9, 1, 3, 8]
in the best case the number of steps taken to search this number 9 should have the minimum number of steps of all the possible casses that can be taken to search a

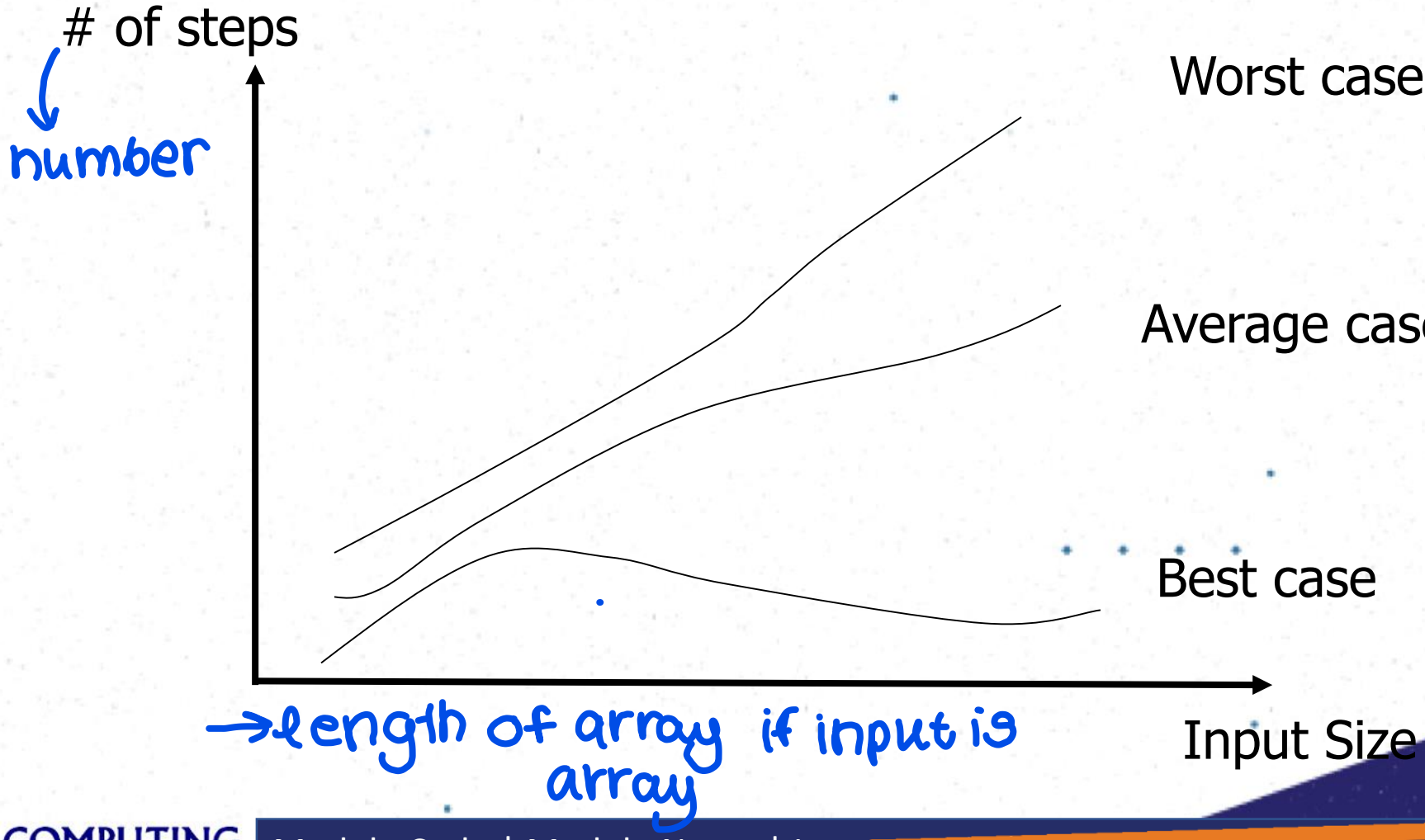
- **Worst case:**
  Maximum number of steps taken on any instance of size n.

worst case is the case which maximum number of steps are taken to search a

- **Average case:**
  An average number of steps taken on any instance of size n.

average number of cases are taken

Worst,Best and Average case(Contd.)



Analysis Methods

inorder to analyze running time methods

1) Operation Count Methods
2) Step Count Method(RAM Model)
3) Exact Analysis
4) Asymptotic Notations

1) Operation count

- Methods for time complexity analysis.
- Select one or more operations such as add, multiply and compare.
- Operation count considers the **time spent on chosen operations** but not all.

but since the times allocated for these operations are different from even computer to computer this method to cal. running time ∴ not good

2) Step Count (RAM Model)

- Assume a generic one processor.
- Instructions are executed one after another, with no concurrent operations. → since one processors
- +, - , =, it takes exactly one step.
- Each memory access takes exactly 1 step. → assumption here
- **Running Time = Sum of the steps.**

RAM Model Analysis.

Example1:

n = 100	1step
n = n + 100	2steps
Print n	1step

addition n + 100

assignment n = n + 100

Steps = 4

Example2:

$sum = 0$ ..... 1 assignment

$\text{for } i = 1 \text{ to } n$ .....
- n+1 assignments
- n+1 comparisons
- n additions

$sum = sum + A[i]$ .....
- n assignments
- n additions
- n memory accesses

Steps = 6n+3

Question 01

- Using RAM model analysis, find out the no of steps needed to display the numbers from 1 to 10.

i = 1 → 1 step

While i <=10 → 11 steps

print i → 10 steps

i = i + 1 → 10 + 10 = 20 steps

Steps = 42

10 steps until number 10

1 steps after 10

addition → adding till 11(10)

intialization (10)

Question 02

- Using RAM model analysis, find out the no of steps needed to display the numbers from 10 to 20.

i = 10 → 1 step

While i <= 20 → 12 steps (Hint :20 – 10 + 2 = 12)

print i → 11 steps

i = i + 1 → 11 + 11 = 22 steps

Steps = 46

because here the printing stares from 10 (in the previous case if it was printed from 0 it would be the same case 2 steps after 20

Question 03

- Using RAM model analysis, find out the no of steps needed to display the even numbers from 10 to 20.

for i = 10 to 20 → (12+ 12 + 11) steps = 35 steps

for(i=10; i<=20; i++)

assigning 12 — compare 12 — addition 11

if i % 2 == 0 → 2 * 11 = 22 steps

remainder 11 → 22

compare 11

print i → 6 steps

10, 12, 14, 16, 18, 20

Steps = 63

Problems with RAM Model

- Differ number of steps with different architecture.

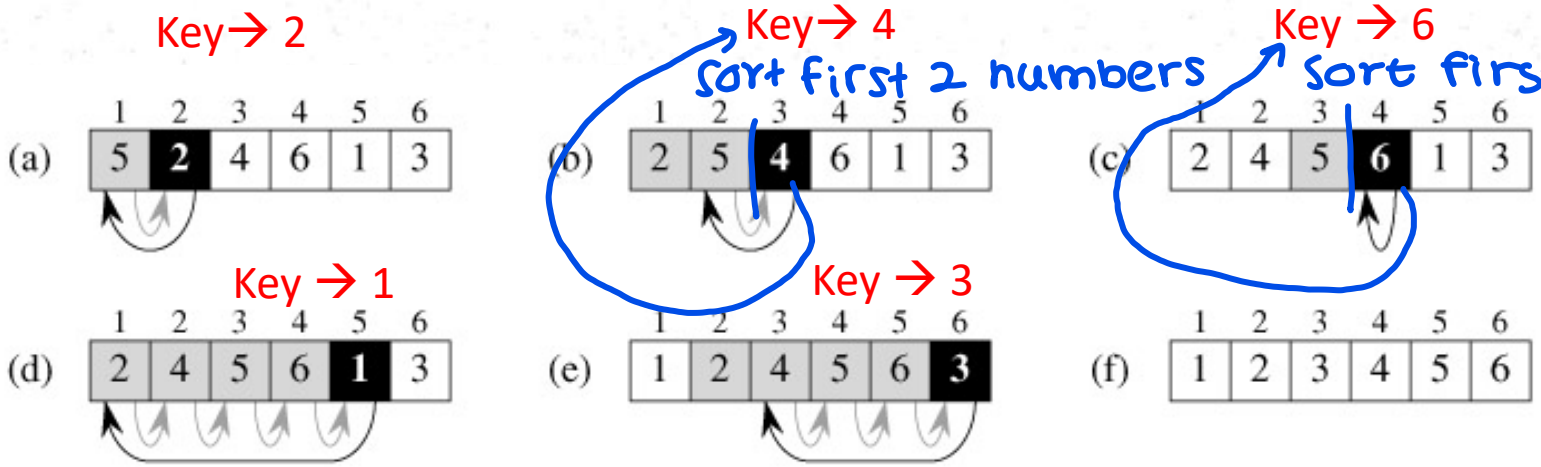
  eg: sum = sum + A[i] is a one step in the CISC processor.

- It is difficult to count the exact number of steps in the algorithm.

  eg: See the insertion sort , efficient algorithm for sorting small number of elements.

below they have shown how a basic INSERTION_SORT algorithm would be analysed using the RAM model

Insertion sort



Pseudocode for insertion sort.

```
INSERTION-SORT(A)
1 for j = 2 to A.length
2     key = A[j]
3     // Insert A[j] into the sorted sequence A[1..j-1]
4     i = j - 1
5     While i > 0 and A[i] > key
6         A[i+1] = A[i]
7         i = i-1
8     A[i+1] = key
```

for loop upto the length of array

key is the 2nd element in array

This will tell you how many elements we have infront to compare

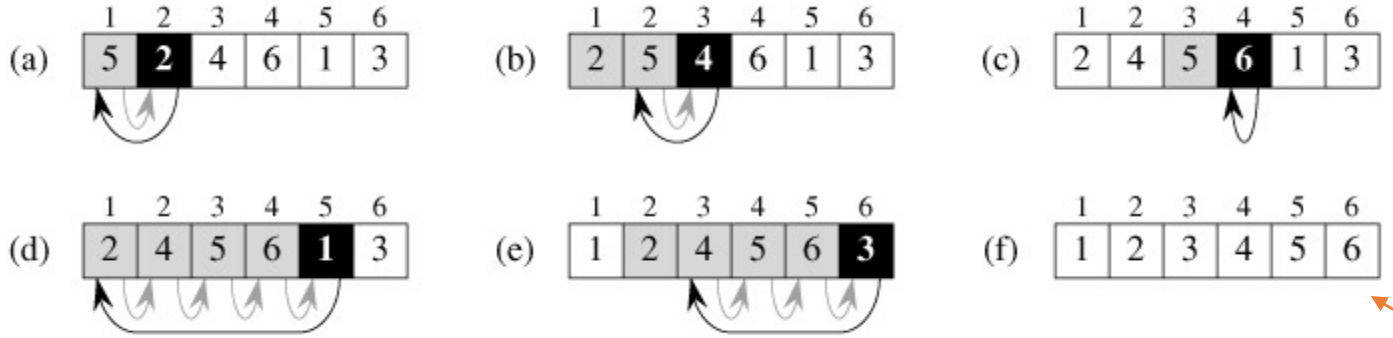
2 2

no change if numbers are equal

This part will shift the large numbers to right

key / marker shifts

Insertion sort - Example



sorted array

- *(a)-(e)* The iterations of the *for* loop → lines 1-8.
- In each iteration, the black rectangle holds the key taken from *A*[*j*],
- Key is compared with the values in shaded rectangles to its left → line 5.
- Shaded arrows show array values moved one position to the right → line 6,
- Black arrows indicate where the key is moved to → line 8.

this get quite complicated we then use the Exact Analysis to find the running time

we learn the insertion sort algori. to do the exact analysis

3) Exact analysis of Insertion sort

- Time taken for the algorithm will depend on the input size (number of elements of the array)

Running Time (Time complexity):

This is the number of primitive operations or steps executed through an algorithm given a particular input.

Running Time : T(n)

assuming the length = n so if n length if J starts from = n

time it takes to run the code

	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**

how many times the line will run

body of loop runs (n-1) times

body of while loop always runs 1 step less

the number of times this will run will be different based on array

decide if numbers should be shifted

i^{th} line takes time c_i where c_i is a constant.

For each j=2,3,…,n , t_j be the number of times the while loop is executed for that value of j

Running Time(contd.)

Using grid

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

t_j (variable) — this always changes

$\sum_{j=2}^{n} 1$

$= 0$

- Best Case (Array is in sorted order)
  - $T(n) \rightarrow an+b$
- Worst Case (Array is in reverse sorted order)
  - $T(n) \rightarrow cn^2 + dn + e$

after assing to 4 equations

equation changed

number of elements are in descending order

every step is 1 just to compare no change in place happens

1	2	3	4	5	6

∴ to just compare all the numbers in array

∴ $(t_j - 1)$ cancels $t_j = 1$ when putting to general equation

Worst Case T(n) $\rightarrow cn^2 + dn + e$

Worst case: The array is in reverse sorted order.

- Always find that $A[i] > key$ in while loop test.
- Have to compare *key* with all elements to the left of the jth position $\Rightarrow$ compare with $j - 1$ elements.
- Since the while loop exits because i reaches 0, there's one additional test after the $j - 1$ tests $\Rightarrow t_j = j$.
- $\sum_{j=2}^{n} t_j = \sum_{j=2}^{n} j$ and $\sum_{j=2}^{n} (t_j - 1) = \sum_{j=2}^{n} (j - 1)$.
- $\sum_{j=1}^{n} j$ is known as an ***arithmetic series***, and equation (A.1) shows that it equals $\frac{n(n+1)}{2}$.

$$\sum_{j=1}^{n} = 1 + 2 + 3 + \ldots + n = \frac{n(n+1)}{2}$$

Worst Case T(n) $\rightarrow cn^2 + dn + e$

equal to

- Since $\sum_{j=2}^{n} j = \left(\sum_{j=1}^{n} j\right) - 1$, it equals $\frac{n(n+1)}{2} - 1$.

  [The parentheses around the summation are not strictly necessary. They are there for clarity, but it might be a good idea to remind the students that the meaning of the expression would be the same even without the parentheses.]

- Letting $k = j - 1$, we see that $\sum_{j=2}^{n}(j-1) = \sum_{k=1}^{n-1} k = \frac{n(n-1)}{2}$.

- Running time is

$$
\begin{aligned}
T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5\left(\frac{n(n+1)}{2} - 1\right) \\
&\quad + c_6\left(\frac{n(n-1)}{2}\right) + c_7\left(\frac{n(n-1)}{2}\right) + c_8(n-1) \\
&= \left(\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2}\right) n^2 + \left(c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8\right) n \\
&\quad - (c_2 + c_4 + c_5 + c_8)
\end{aligned}
$$

- Can express $T(n)$ as $an^2 + bn + c$ for constants a, b, c (that again depend on statement costs) $\Rightarrow T(n)$ is a *quadratic function* of n.

our equation should finally be in this form

exact analysis is hard and complex

Asymptotic Notations

therefore more simple

when our array is very large we go for this approach

- RAM Model has some problems.
- Exact analysis is very complicated.

Therefore we move to **asymptotic notation**

- Here we focus on determining the biggest term in the complexity function.
- Sufficiently large size of n.

Asymptotic Notations(Contd.)

- There are three notations.

1) **O - Notation**
2) **Θ - Notation**
3) **Ω - Notation**

Big O - Notation

Introduced by Paul Bechman in 1892.

- We use Big O-notation to give an upper bound on a function.

Definition:

$O(g(n)) = \{ f(n)$: there exist positive constants c and n_o such that

$0 \le f(n) \le cg(n)$ for all $n \ge n_o \}$.

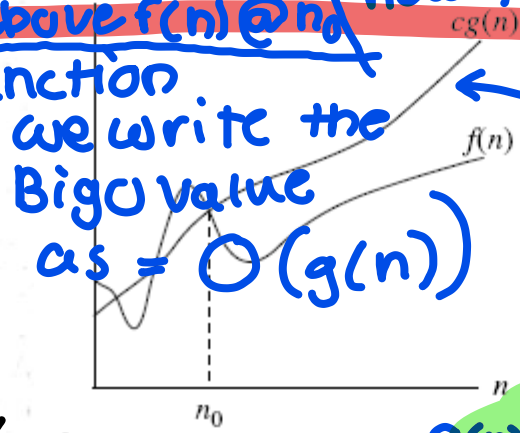
Eg: What is the big O value of f(n)=2n + 6 ?

c = 4

$n_o = 3$

g(n)=n therefore

f(n) = O(n)



$g(n)$ is an *asymptotic upper bound* for $f(n)$.

If $f(n) \in O(g(n))$, we write $f(n) = O(g(n))$

Back to the example

when algorithms are given instead of just f(n)

- Alternative calculation: now we need to find our function f(n)

	cost	times
sum = 0	c_1	1
for i = 1 to n	c_2	n+1
sum = sum + A[i]	c_3	n

$$T(n) = c_1 + c_2\,(n+1) + c_3\,n$$

$$= (c_1 + c_2) + (c_2 + c_3)\,n$$

$$= c_4 + c_5\,n \rightarrow O(n)$$

as n

Proof: $c_4 + c_5\,n \le c\,n$ → TRUE for $n \ge 1$ and $c \ge c_4 + c_5$

Big O – Notation(Contd.)

Assignment (s = 1)

Addition (s+1)

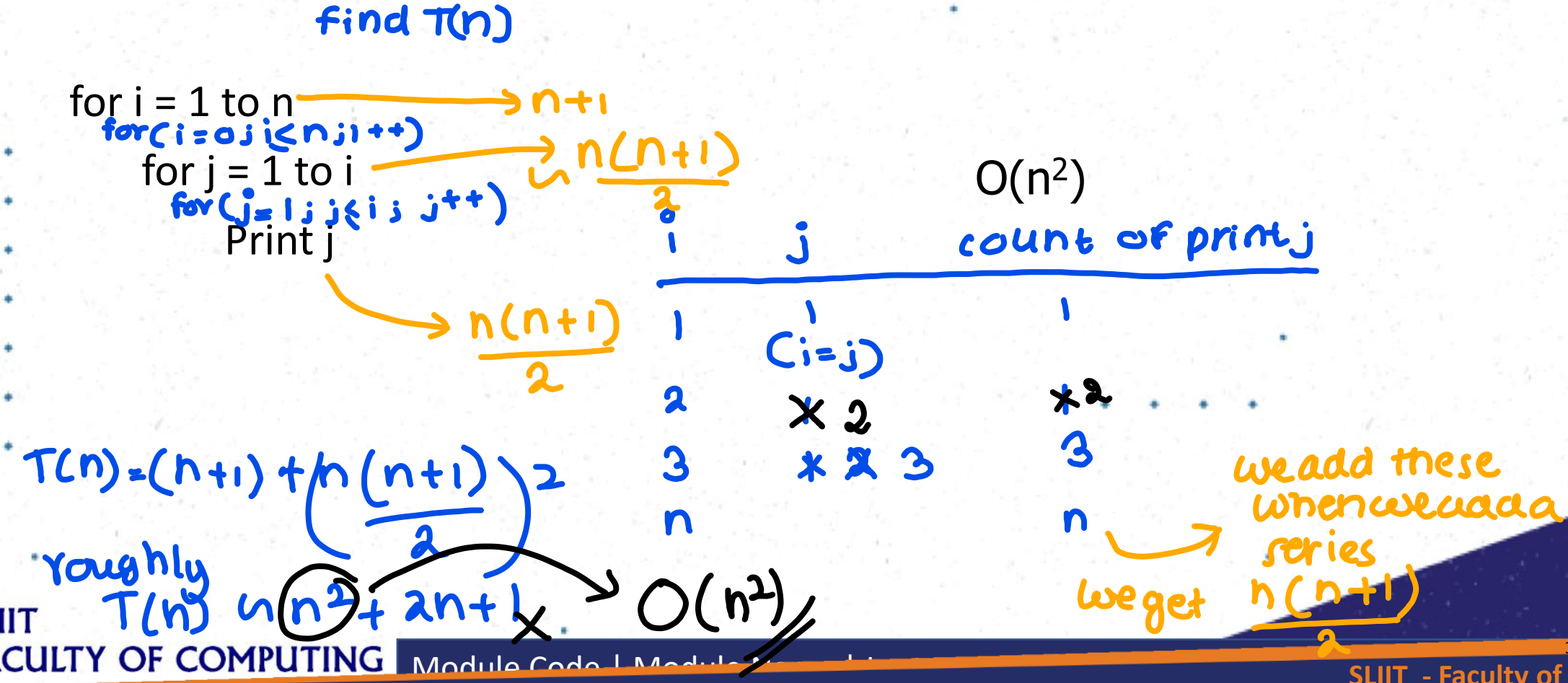
Multiplication (s*2)

Comparison (S<10)

O(1)

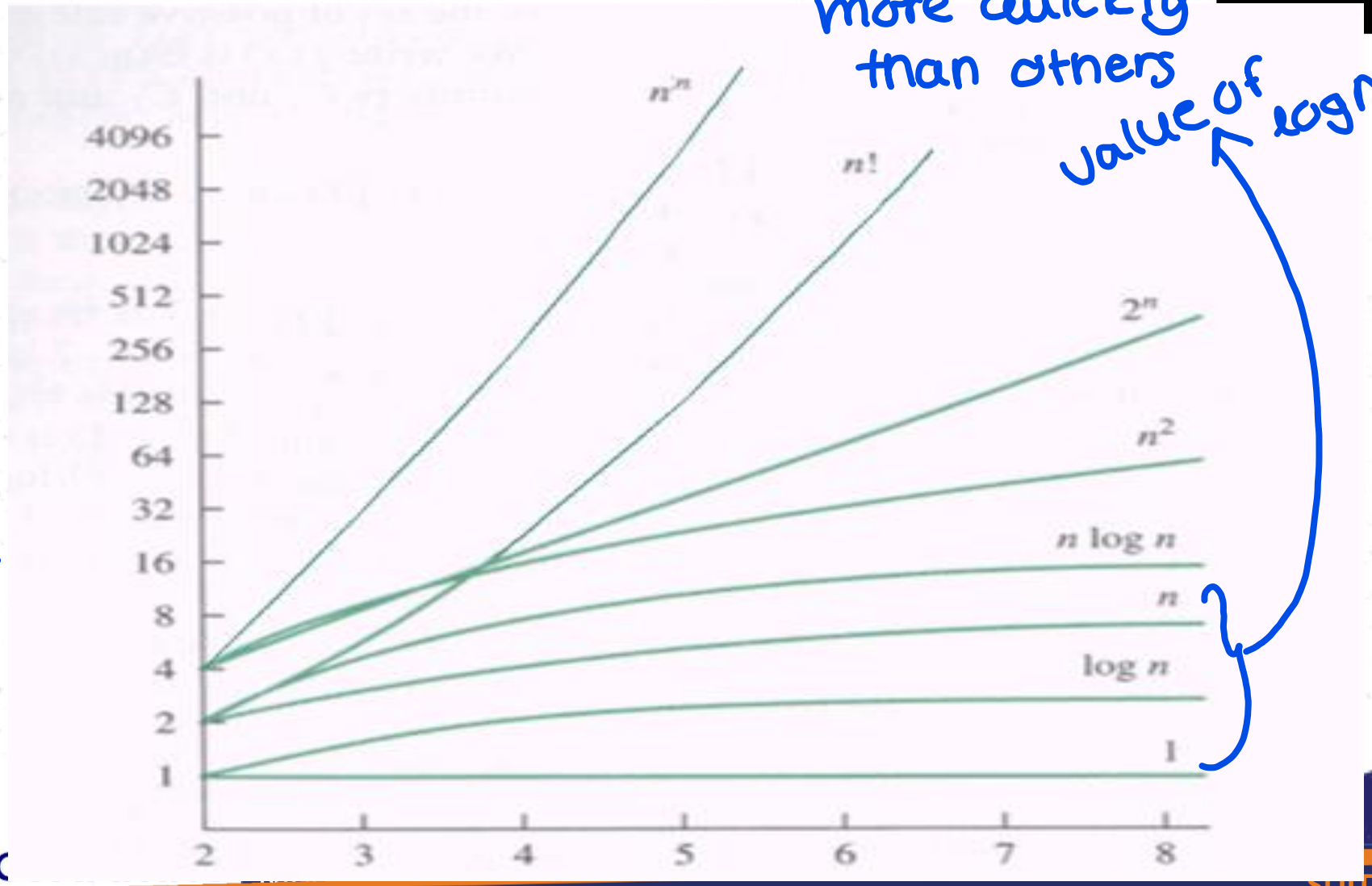
Question

- Find the Big O value for following fragment of code.



when the n type inside the big O value is smaller we can get the answers more quickly than others

Graphs of functions



Time it takes to execute ↑

value of log n < n

n	$\log n$	n	$n\log n$	n^2	n^3	2^n
4	2	4	8	16	64	16
8	3	8	24	64	512	256
16	4	16	64	256	4,096	65,536
32	5	32	160	1,024	32,768	4,294,967,296
64	6	64	384	4,094	262,144	$1.84 * 10^{19}$
128	7	128	896	16,384	2,097,152	$3.40 * 10^{38}$
256	8	256	2,048	65,536	16,777,216	$1.15 * 10^{77}$
512	9	512	4,608	262,144	134,217,728	$1.34 * 10^{154}$
1024	10	1,024	10,240	1,048,576	1,073,741,824	$1.79 * 10^{308}$

Big O – Notation(Contd.)

- Find the Big O value for the following functions.

  (i) $T(n)= 3 +5n + 3n^2$ → $O(n^2)$

  (ii) $f(n)= 2^n + n^2 +8n +7$ → $O(2^n)$

  (iii) $T(n)= n + \log n +6$ → $O(n)$

Answers:

(i) $O(n^2)$

(ii) $O(2^n)$

(iii) $O(n)$

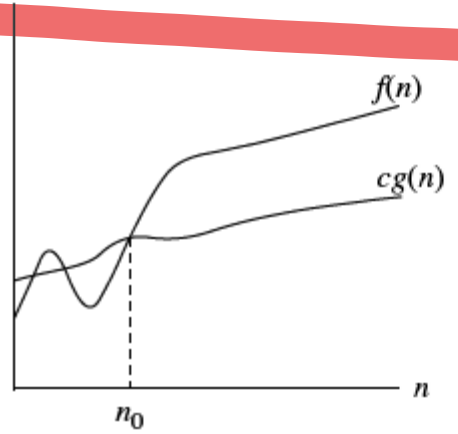
Ω - Notation

- Provides the lower bound of the function.

Definition:

this is the opposite of big O here the g(n) function is smaller than f(n) @ n

$\Omega(g(n))$ = { f(n) : there exist positive constants c and n_0 such that $0 \leq cg(n) \leq f(n)$ for all $n \geq n_o$}



$g(n)$ is an ***asymptotic lower bound*** for $f(n)$.

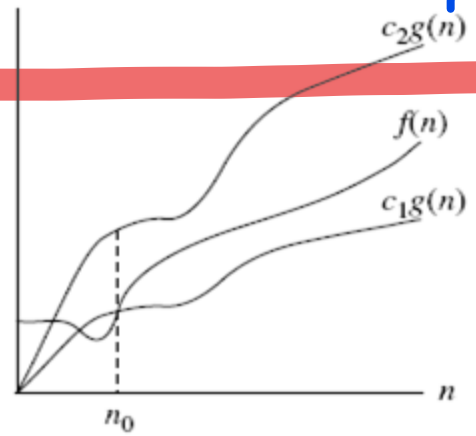
Θ - Notation

- This is used when the function f can be bounded both from above and below by the same function g.

Definition:

$\Theta(g(n))$ ={ f(n): there exist positive constant c_1, c_2, and n_0 such that $0 \leq c_1\, g(n) \leq f(n) \leq c_2 g(n)$ for all $n \geq n_0$ }

→ the function g(n) when multiplied by c_2 goes above f(n) when multiplied by c_1 goes below f(n)



$g(n)$ is an *asymptotically tight bound* for $f(n)$.

Summary

- What is an algorithm?
- Properties of an algorithm.
- Design methods.
- Pseudocode.
- Analysis(Operation count & Step count, RAM model).
- Insertion Sort.
- Asymptotic Notation

References

- T.H. Cormen, C.E. Leiserson, R.L. Rivest, Clifford Stein Introduction to Algorithms,3rd Edition, MIT Press, 2009.