Algorithm etymology

- Algorism = process of doing arithmetic using Arabic numerals.
- A misperception:
- Algiros [painful] + Arithmos [number]
- Origin: Abu’ Abd Allah Muhammad ibn Musa al-Khwarizmi was a famous 9th century Persian textbook author who wrote Kitab al-jabr wa’l-muqabala, which evolved into today’s high school algebra text.

Algorithm definition:

- “A procedure for solving a mathematical problem (as of finding the greatest common divisor) in a finite number of steps that frequently involves repetition of an operation”

- An algorithm is a finite, definite, effective procedure, with some input and some output”

Why study algorithm because they work in:

- **Internet**: web search, packet routing, distributed file sharing
- **Biology**: human genome project, protein folding,…
- **Computers**: circuit layout, databases, caching, networking, compilers..
- **Computer graphics**: movies, video games, virtual reality
- **Security:** cell phones, e-commerce
- **Multimedia**: MP3, JPG, DivX, HDTV, face recognition,…
- **Social networks:** recommendations, news feeds, advertisements,…
- **Physics:** N-body simulation, particle collision simulation,…

We emphasize algorithms and techniques that are useful in practice.

Algorithms (for computers)

- Algorithms:
- For a computation problem a series of calculations that convert an input to an output, is called algorithm.
- For example, given you a bunch of numbers <31,41,59,26,41,58>, a sorting algorithm can put these numbers from small to large output into <26,31,41,41,58,59>

Algorithmics

- It is the science that lets designers study and evaluate the effect of algorithms based on various factors so that the best algorithmics selected to meet a particular task in given circumstances.

- It is also the science that tells how to design a new algorithm for a particular job.

Is there any problem with the algorithm?

- Given n matrices <A1,A2,,….An>, ask for their product A1A2….An. Since the matrix multiplication has the characteristics of the union law, there are various legal multiplication sequences. For example, when n=4, the following order is valid (A1(A2A3)), ((A1A2)A3), A4(A1A2)A3), ((A1A2)(A3A4)), ((A1(A2A3)A4), or (((A1A2)A3)A4).

- If these matrices are square, then the order of multiplication does not affect the time to calculate the answer, otherwise, the order of multiplication for time will be very large. It would take a long time to try all that would be followed by an efficient algorithm to solve the problem.

Matrix multiplication

- This is a very classic topic or problem of Dynamic Programming.
- Assume: A1 matrix size is 2*3, A2 matrix size is 3*1, A3 matrix size is 1*2
- A1(A2A3)) the number of multiplication required is 6+12 = 18
- ((A1A2)A3) The number of multiplication required is only 6+4=10, which is better than the former.

- **(A1(A2A3))**
- (2*3((3*1)(1*2)
- (2*3(3*1*2))
- ((2*3)(3*2))+6
- (2*3*2)+6
- 12+6 = 18

((A1A2)A3)

((2*3)(3*1))1*2)

(2*3*1)(1*2)

(6)+((2*1)(1*2))

(6) + (2*1*2)

6+4 = 10

Algorithms as a Technology

Efficiency: different algorithms may have considerable differences in efficiency when solving the same problem. This difference may be far greater than the impact of hardware or software.

For example, in the second chapter we will introduce the insertion sort to sort n numbers take the time is about $c1n^2$ where c1 is a constant. Merge sort takes about c2 nlogn (c2 is constant). In general c1 will be less than c2, but the impact of constants will be much smaller than n.

In fact, optimized code can speed up at a limited rate, but improving algorithms can greatly improve performance.

Algorithms as a Technology (cont.)

- For example: c1=2, c2=50,

  Faster computer A can run 10^9 instructions/second,

  Slower computer B can run 10^7 instructions/second,

  Assume to sort one million instructions: $n=10^6$

  For A, Sort time = $2(10^6)^2$ instructions in insertion sort 10^9 instructions/sec

  The computer B running merge sort time = $50 \cdot 10^6 \log 10^6$

  Instructions running at 10^7 instructions/sec

Although computer B is slower, the merge sort used is better than the insertion sort used on computer A. therefor, the time that computer B ran out was much shorter than that of computer A.

- Note: When sorting the number of digits reaches 10^7 , it takes about 2.3 days to use insertion sort, but merge sort takes only 20 minutes. So the greater the size of the input, the more obvious the advantages of merge sort.
- This is also the spirit of the algorithm design. We hope that the larger the size of the input, the more obvious the advantage is has.

$$\frac{2 \cdot (10^6)^2 \text{ instructions}}{10^9 \text{ instructions/second}} = 2000 \text{ seconds} ,$$

while computer B takes

$$\frac{50 \cdot 10^6 \lg 10^6 \text{ instructions}}{10^7 \text{ instructions/second}} \approx 100 \text{ seconds} .$$

Algorithm characteristics:

- **Deterministic**: such algorithm does the same thing on the same input presented next time.
- **Correct:** if an algorithm halts with correct output upon every instance of input. Then we say that correct algorithm solves the given computational problem.
- **Incorrect:** such algorithm might not halt at all on some input instances. Or it might halt with an answer other than desired one. Incorrect algorithms can be sometime useful if their error rate can be controlled. Robin-Miler returns definite for composite but return almost sure certain prime for primality test.
- **Description:** it can be flow chart, program code, pseudo code, hardware or English description even their mixture.

Size of Input and Time:

- Input size is also termed as problem size. Mostly it is considered as number of numbers or objects in the input. But those object or data may have to extensively read and write in the memory (memory movement). In such case it might be more relevant to consider number bits instead .
- This is not considered in usual time units like seconds, minutes etc., but number of clocks or flops or number of primitive operations performed to solve the problem. Some dominant count of a primitive might work well.

Bubble Sort

Sorting problem:

- Sorting is frequently posed as a permutation problem.
- INPUT: $<a_1,a_2,a_3,.....a_n>$
- OUTPUT: $< a_{i1}, a_{i2}, a_{i3},.....a_{in}>$
- For all $i<=k<n$ $a_{ik}<=a_{ik+1}$

- Sorting is a very important fundamental operation that is frequently involved in various computational problem. It is profoundly analyzed problem and used in explaining algorithm analysis and design. Many algorithms have been designed for sorting.

BUBBLE SORT ALGORITHM

- **Input: unsorted array of double-type of size nA**
- **Output: sorted array of double-type of size n A**
- ```
  for (i= n-1; i>0;--i)
      for(j=0;j<i;++j)
          if(A[j]>A[j+1]) {
              temp=A[j];
              A[j]=A[j+1];
              A[j+1]=temp;
          }
  ```
  ```

Primitive Operations in Bubble Sort

Following are the primitive operation involved in the above algorithm.

- 1. Addition ADD4: n-1 , --i, ++j, j+1
- 2. comparison CMP4 : i>0 , j<i,
- 3. comparison CMP8 : A[j]>A[j+1]
- 4. memory move MMV4: i=n, j=0
- 5. memory move MMV8: temp=A[j]; A[j]=A[j+1]; A[j+1]=temp;

From program point of view we do regard operations on 4 byte value different from operations on double values of 8bytes, that why we have CMP8 and MMV8 which might take twice as much time as CMP4 and MMV4. However, we assume that each 4byte operations take same time and similar is also for 8byte operations.

Bubble Sort Algorithm for easy analysis

```
i= n-1;                          1 time
for (; i>0; --i){j=0;            i>0: n  others n-1 times
    for(;j<i;++j)                j<i: n others n-1 times
        if(A[j]>A[j+1]) {        firstly & second lastly
            temp=A[j];           j<i: 2 others 1 time
            A[j]=A[j+1];         & lastly
            A[j+1]=temp;         j<i: 1 others 0 time
        }
```

Input is in reverse order: Worst Case analysis all operations occur in every iteration in Bubble sort

	ADD4	CMP3	CMP4	MMV8	MMV4
Outer-loop	n		n		n
Inner-loop	$2n^2-2n$	$0.5n^2-0.5n$	$0.5n^2+0.5n$	$1.5n^2-1.5n$	
Sum	$2n^2-n$	$0.5n^2-0.5n$	$0.5n^2+1.5n$	$1.5n^2-1.5n$	n
SUM as 4Byte OP	$2n^2-n$		$1.5n^2+0.5n$		$3n^2-2n$
G-Total	$6.5n^2-2.5n$				

- Worst case $w(n) = 6.5\,n^2 - 2.5n$

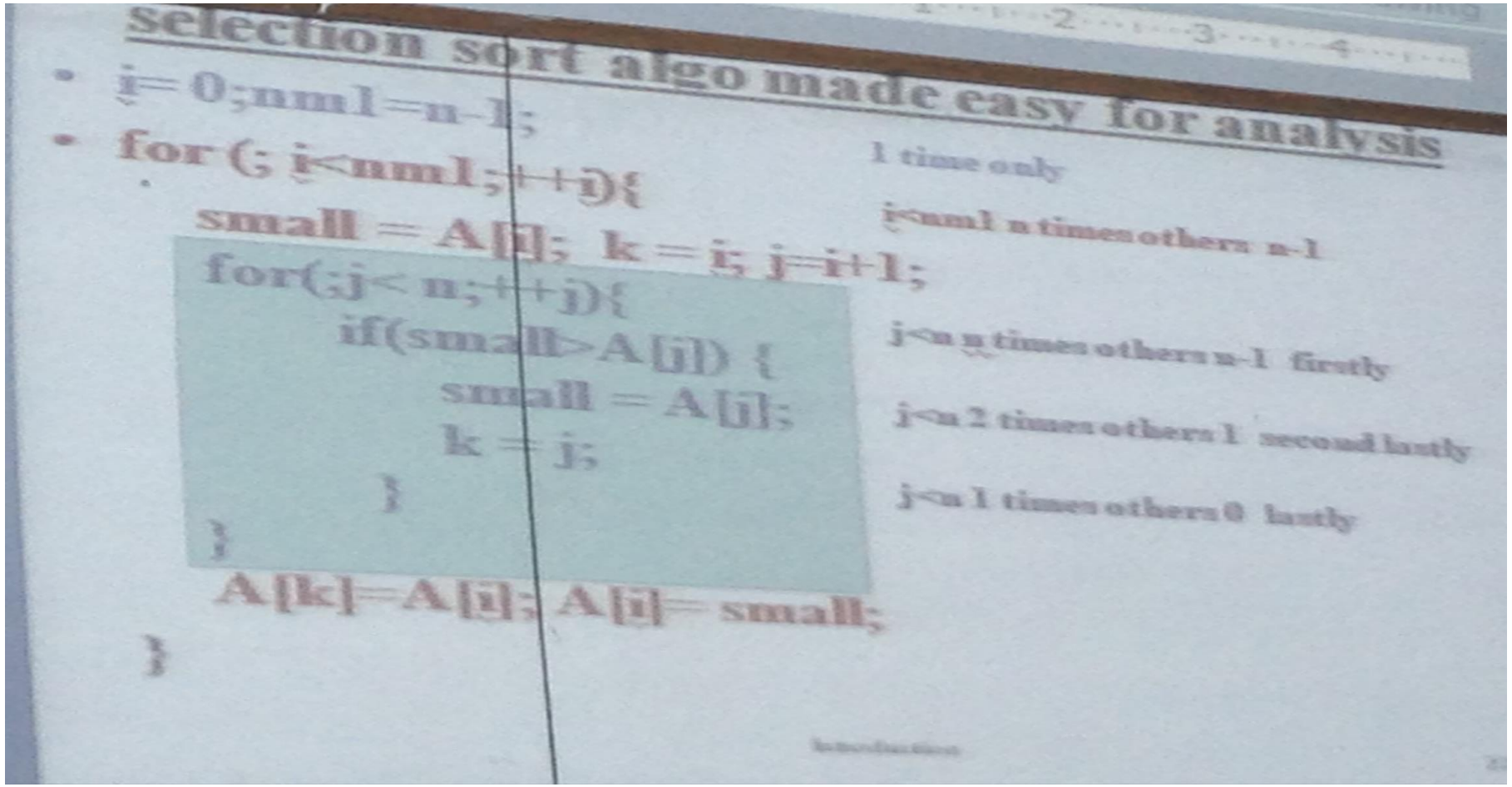
Selection Sort

SELECTION SORT ALGORITHM

- Input: unsorted array of double of size n
- Output: sorted array of double of size n
- for(i=0; i<n-1; ++i){

```
    Small=A[i]; k=i;
    for(j=i+1;j<n;++j){
        If(small>A[j]){
            Small=A[j];
                K=j;
        }
}
A[k]=A[i];
A[i]=small;
```

- **Selection sort algo made easy for analysis**



Primitive Operations in Selection Sort

Input is in reverse order: Worst Case analysis all operations occur in every iteration of Selection sort

	ADD4	CMPS	CMP4	MMV8	MMV4
Outer-loop	2n-1		n	3n-3	2n
Inner-loop	$0.5n^2-0.5n$	$0.5n^2-0.5n$	$0.5n^2+0.5n$	$0.5n^2-0.5n$	$0.5n^2-0.5n$
Sum	$0.5n^2+1.5n-1$	$0.5n^2-0.5n$	$0.5n^2+1.5n$	$0.5n^2+2.5n-3$	$0.5n^2+1.5n$
SUM as 4Byte OP	$0.5n^2+1.5n-1$		$1.5n^2+0.5n$		$1.5n^2+6.5n-6$
G-Total	$3.5n^2+8.5n-7$				16

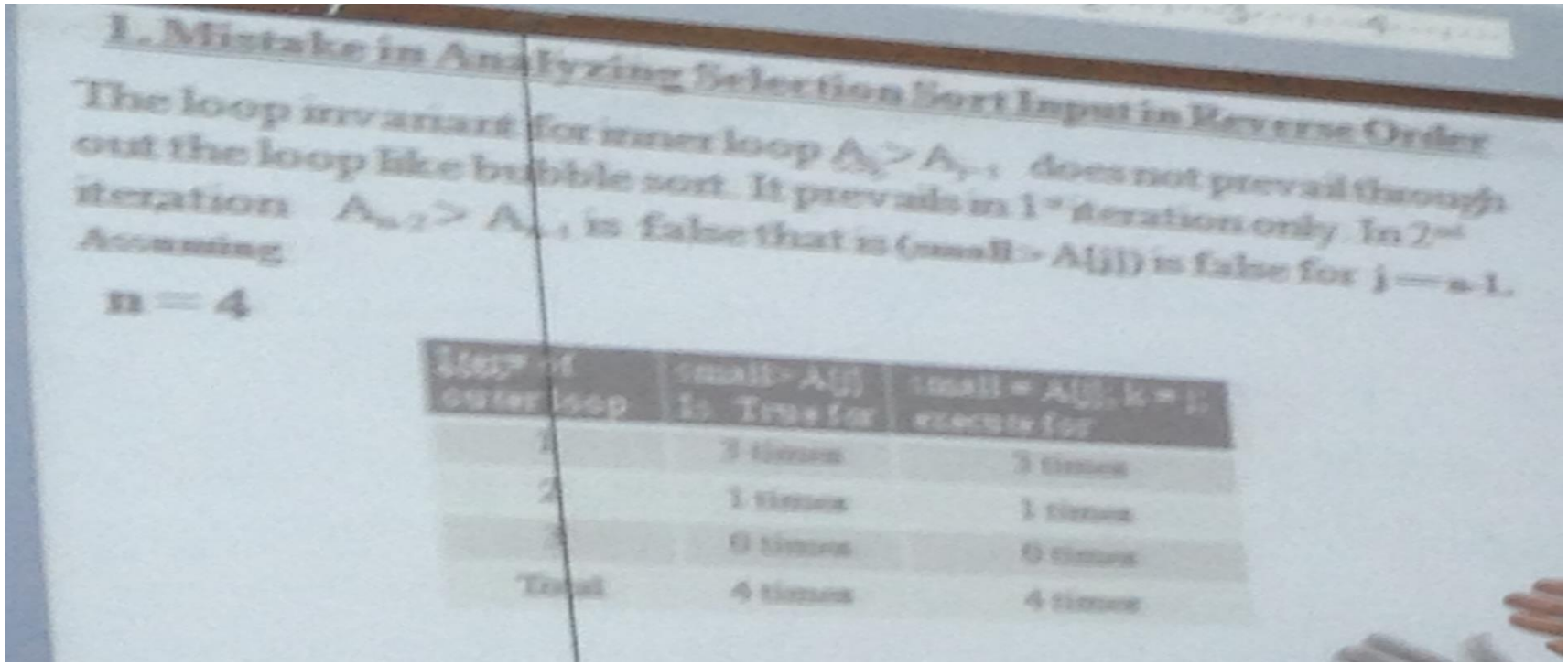
- Worst case $w(n) = 3.5n^2 + 8.5n - 7$
- **Input is in desired order: Best Case analysis of inner loop will never occur**

Best case $b(n) = 2n^2 + 10n - 7$, $g(n) = 2.75n^2 + 9.25n - 7$

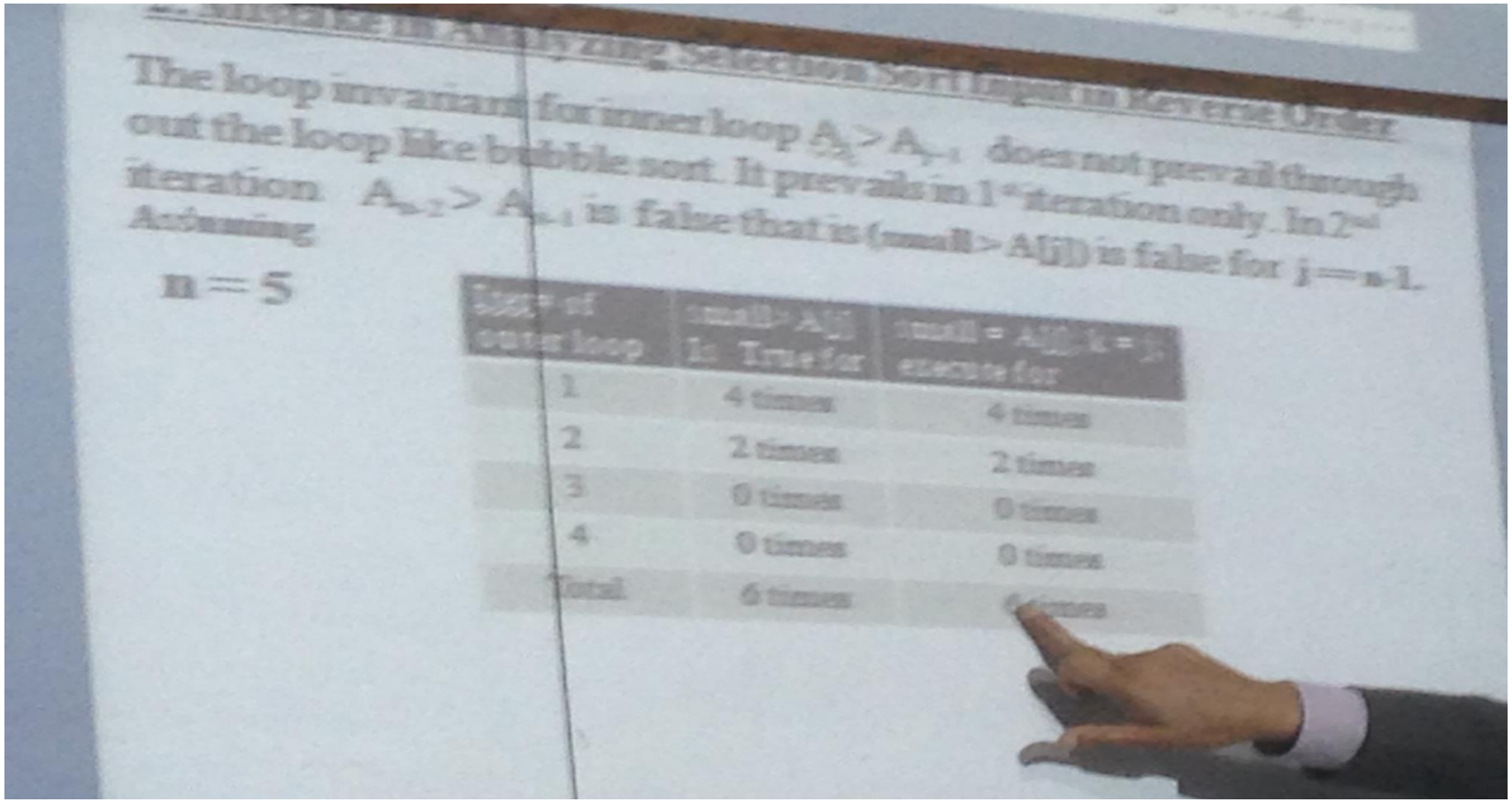
1. Mistake in analyzing selection sort Input in Reverse Order

the loop invariant for inner loop $A_i > A_{j+1}$ does not prevail through out the loop like bubble sort. It prevails in 1st iteration only. In 2nd iteration $A_{n-2} > A_{n-1}$ is false that is (small>A[j]) is false for j=n-1.

Assuming: n=4



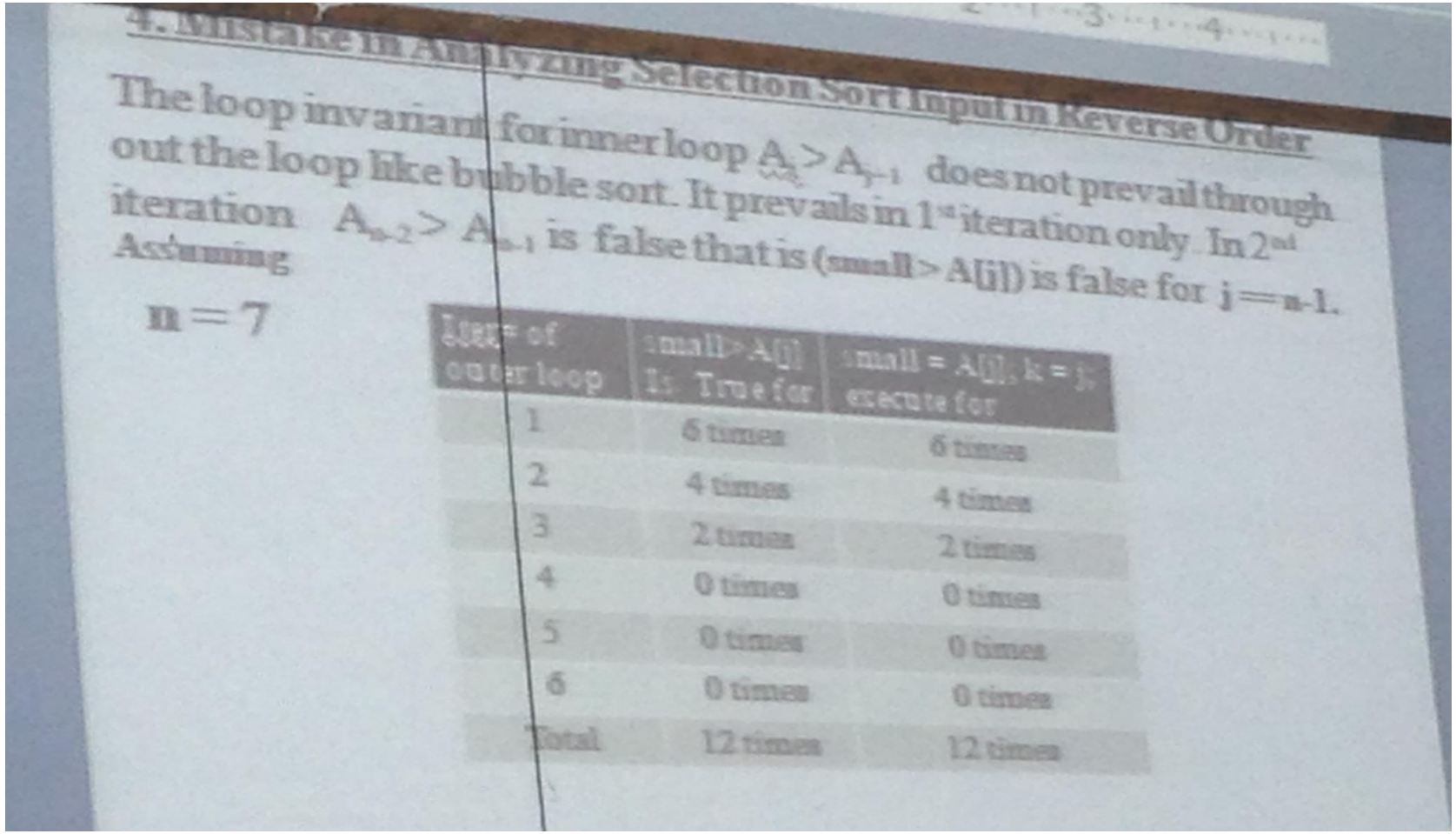
n=5


The loop invariant for inner loop
out the loop like bubble sort. It prevails in 1st iteration only. In 2nd
iteration
is false that is
Assuming
n = 5
1
4 times
4 times
2
2 times
2 times
3
0 times
0 times
4
0 times
0 times
Total
6 times

n=6


3. Mistake in Analyzing
Input in Reverse Order
The loop invariant for inner
does not prevail through
out the loop like bubble sort. It prevails in 1st iteration only. In 2nd
iteration
is false that is
Assuming
n = 6

n=7



4. Mistake in Analyzing Selection Sort Input in Reverse Order

The loop invariant for inner loop $A_j > A_{j-1}$ does not prevail through out the loop like bubble sort. It prevails in 1st iteration only. In 2nd iteration $A_{n-2} > A_{n-1}$ is false that is (small > A[j]) is false for j = n-1.

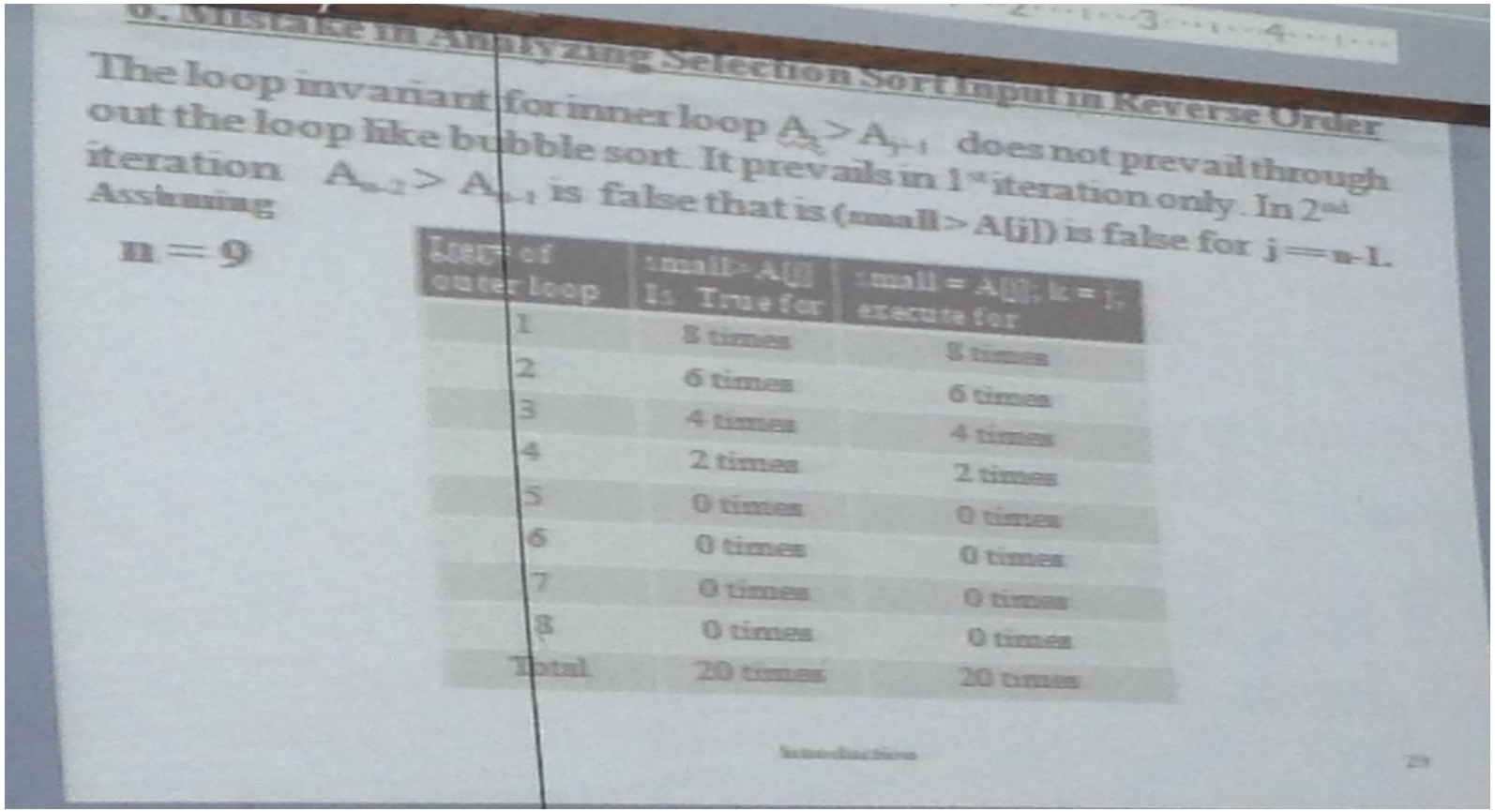
Assuming

n = 7

Count of outer loop	small > A[j] Is True for	small = A[j]; k = j; execute for
1	6 times	6 times
2	4 times	4 times
3	2 times	2 times
4	0 times	0 times
5	0 times	0 times
6	0 times	0 times
Total	12 times	12 times

n=8

n=9



6. Mistake in Analyzing Selection Sort Input in Reverse Order

The loop invariant for inner loop $A_j > A_{j+1}$ does not prevail through out the loop like bubble sort. It prevails in 1st iteration only. In 2nd iteration $A_{n-2} > A_{n-1}$ is false that is (small > A[j]) is false for j == n-1. Assuming

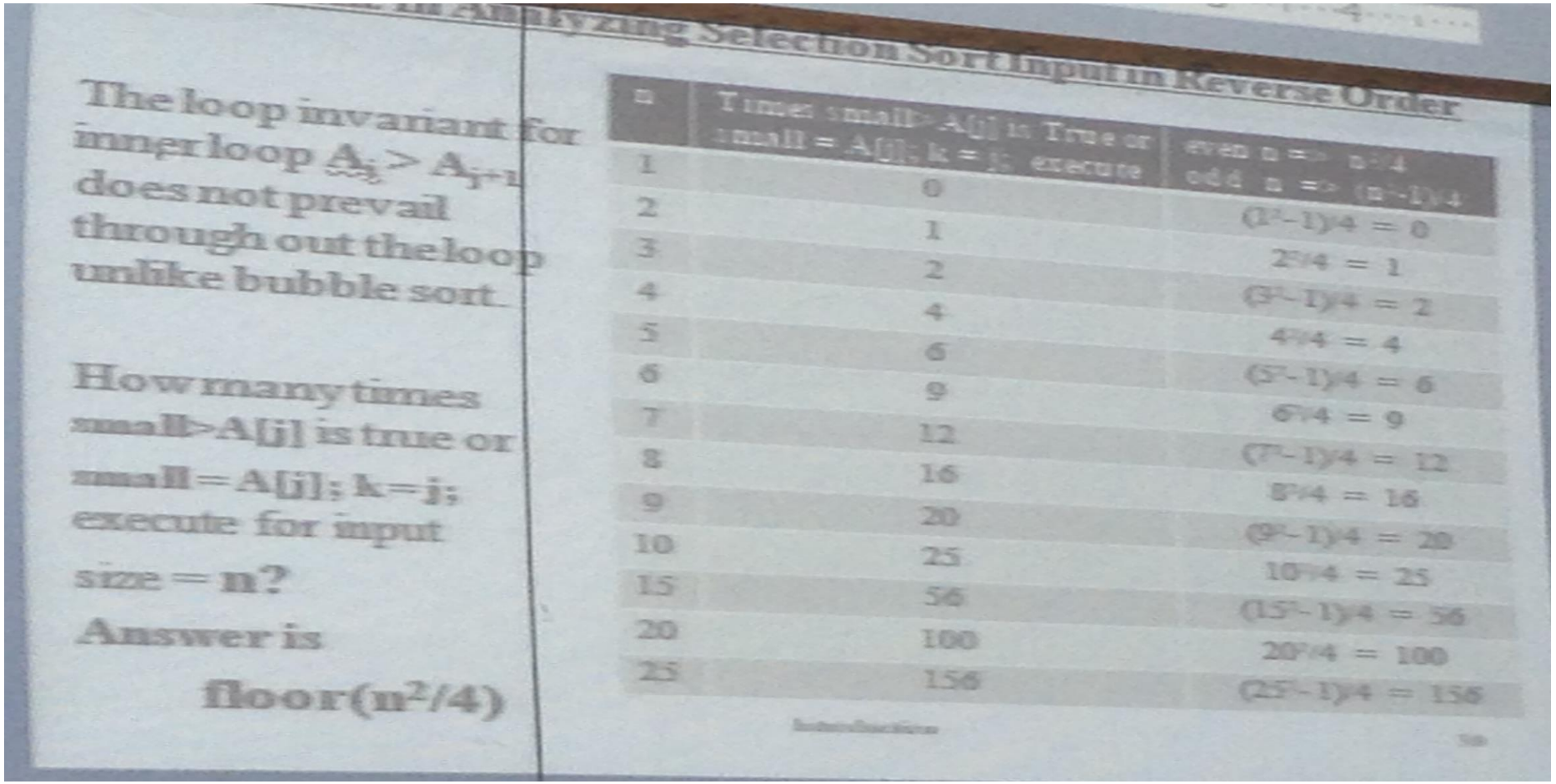
$n = 9$

Index of outer loop	small > A[j] is True for	small = A[j]; k = j; execute for
1	8 times	8 times
2	6 times	6 times
3	4 times	4 times
4	2 times	2 times
5	0 times	0 times
6	0 times	0 times
7	0 times	0 times
8	0 times	0 times
Total	20 times	20 times

Introduction

29

Mistake un analyzing Selection Sort Input in Reverse Order



The loop invariant for inner loop $A_{j} > A_{j+1}$ does not prevail through out the loop unlike bubble sort.

How many times small>A[j] is true or small = A[j]; k = j; execute for input size = n?

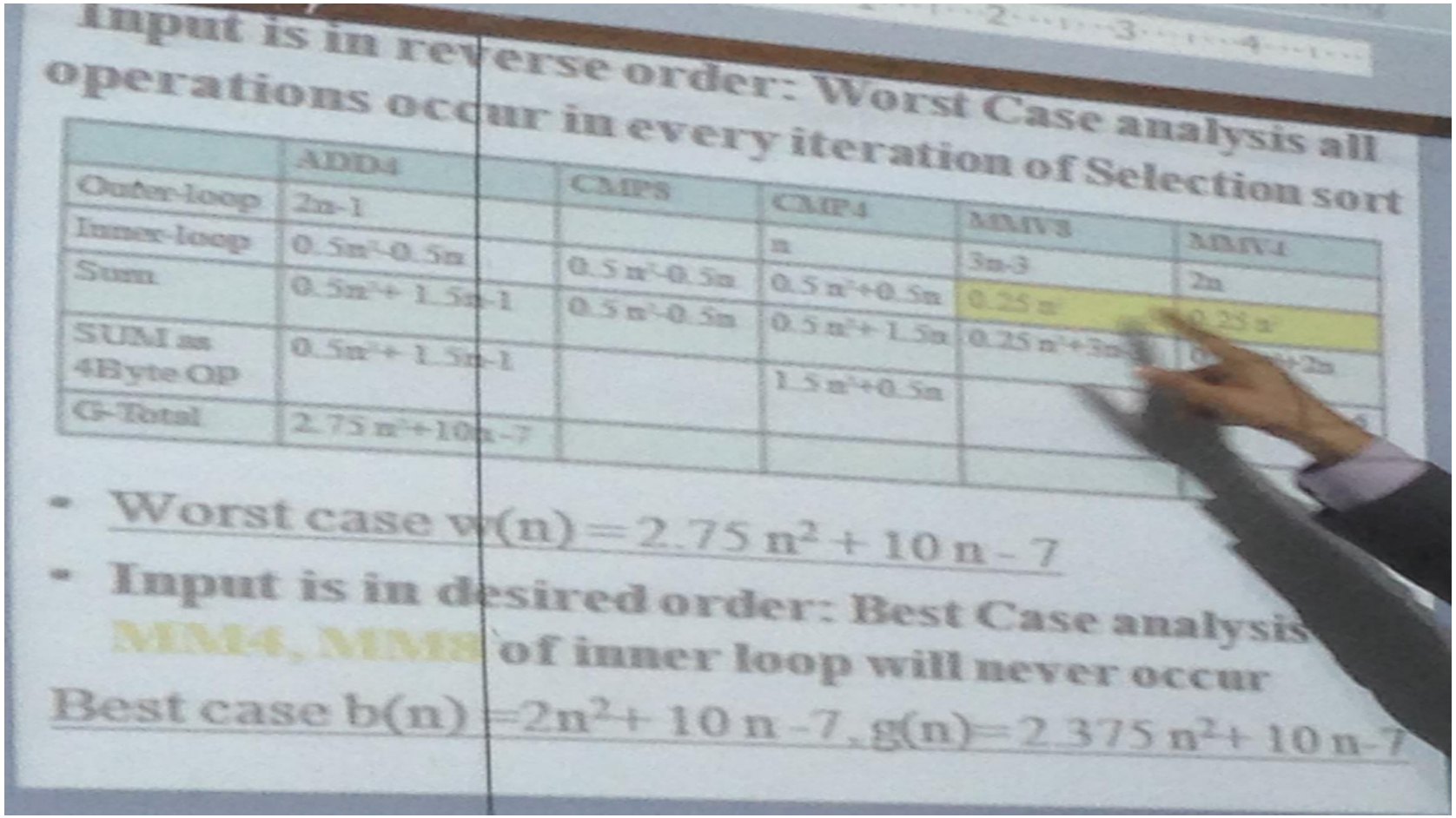
Answer is

floor($n^2/4$)

Analyzing Selection Sort Input in Reverse Order

n	Times small>A[j] is True or small = A[j]; k = j; execute	even n => n^2/4 odd n => (n^2-1)/4
1	0	
2	1	$(1^2-1)/4 = 0$
3	2	$2^2/4 = 1$
4	4	$(3^2-1)/4 = 2$
5	6	$4^2/4 = 4$
6	9	$(5^2-1)/4 = 6$
7	12	$6^2/4 = 9$
8	16	$(7^2-1)/4 = 12$
9	20	$8^2/4 = 16$
10	25	$(9^2-1)/4 = 20$
15	56	$10^2/4 = 25$
20	100	$(15^2-1)/4 = 56$
25	156	$20^2/4 = 100$
		$(25^2-1)/4 = 156$

- **Worst Case analysis**



Insertion Sort

INSERTION SORT ALGORITHM

- Input: unsorted array of double of size n
- Output: sorted array of double of size n
- for (i= 1; i<n;++i){

```
item = A[i]; j = i-1;
while (A[j] > item) {
        A[j+1]= A[j];
        --j; if (j<0) break;
}
A[j+1]= item;
}
```

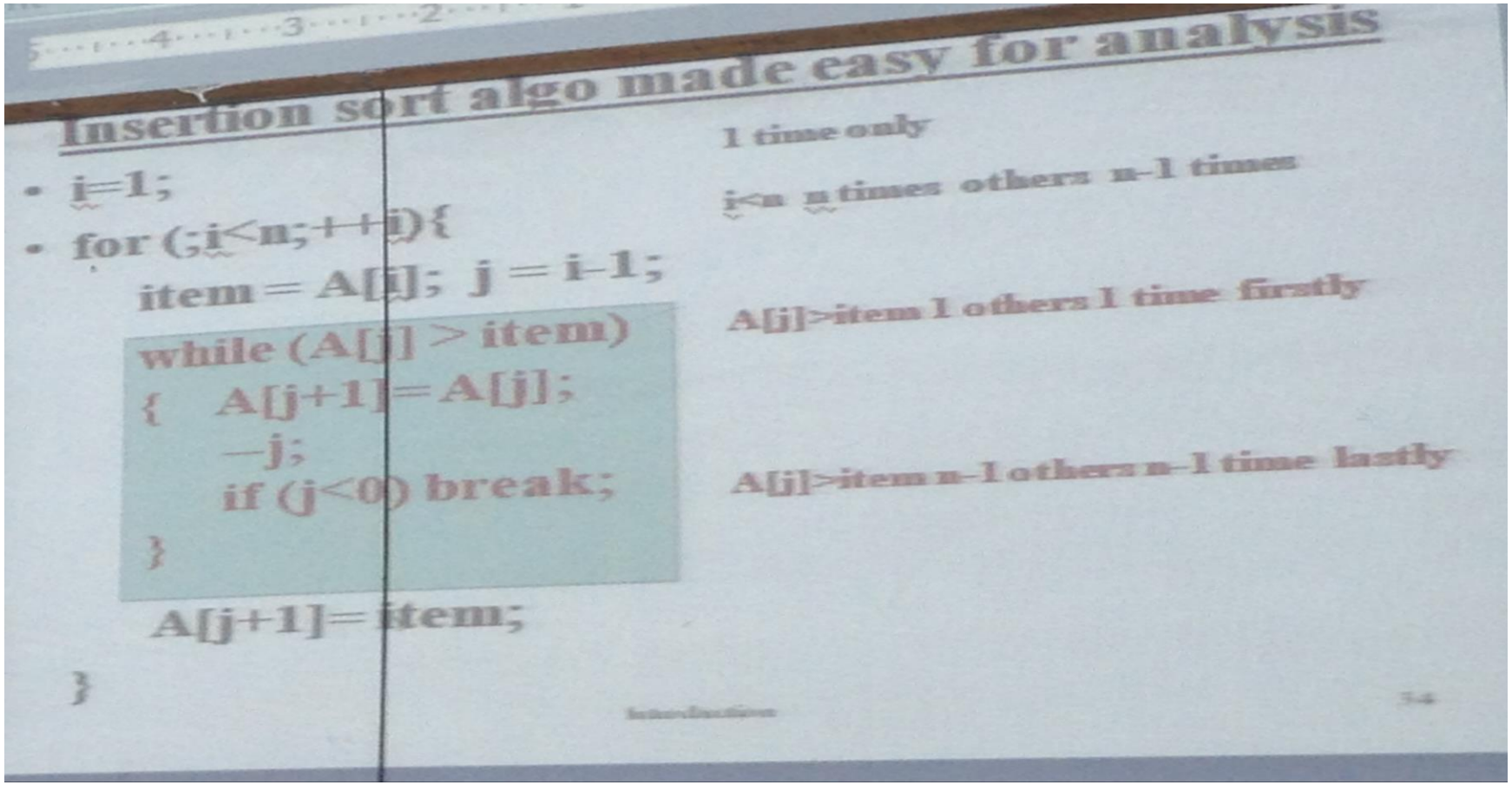
Primitive Operations in Insertion Sort

Following are the primitive operation involved in the above algorithm.

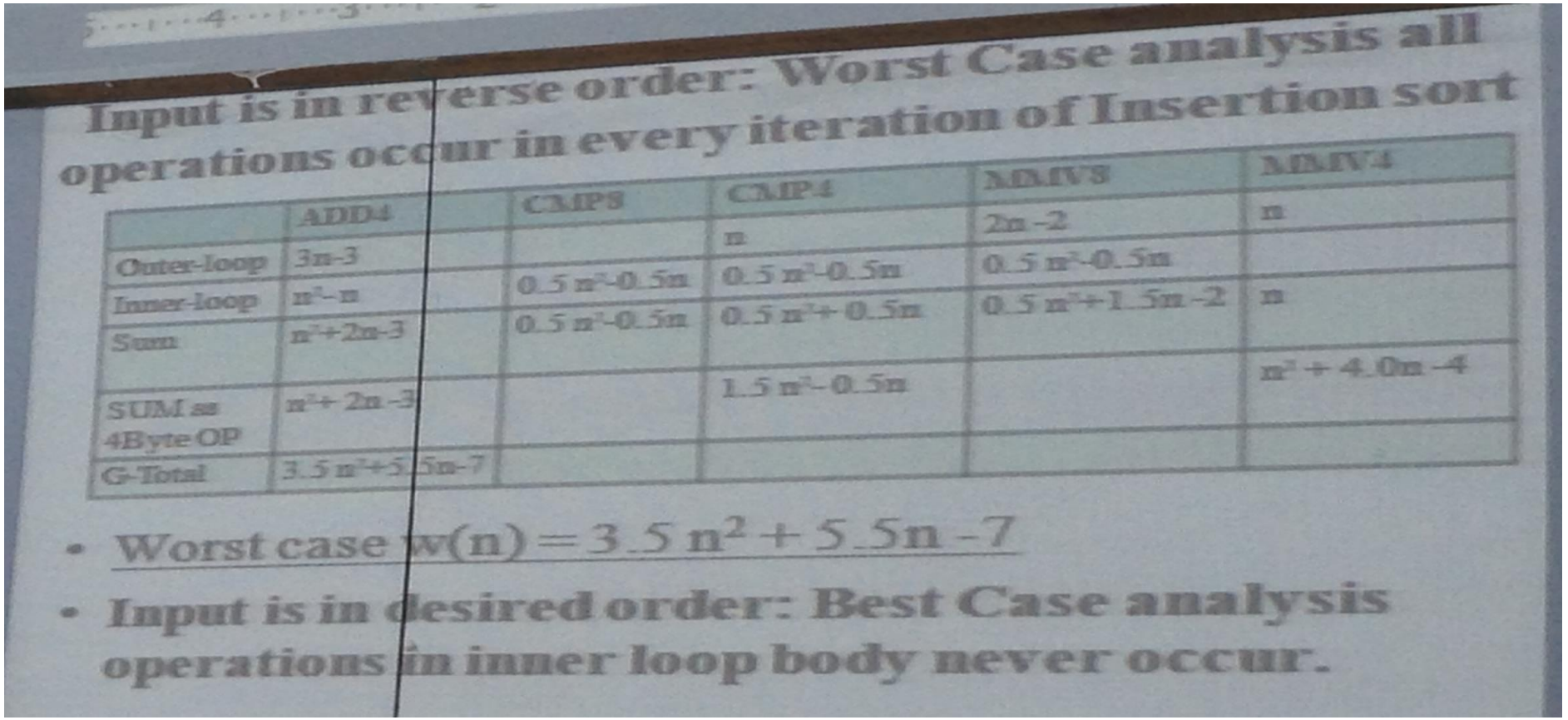
- 1. Addition ADD4: ++j, i-1, j+1, --j
- 2. comparison CMP4: j<n, j<0,
- 3. comparison CMP8: A[j]> item
- 4. memory move MMV4: j=1, j=i-1, j=0
- 5. memory move MMV8: item=A[i]; A[j+1]=A[j];

From program point of view we do regard operations 4 byte value different from operations on double values of 8bytes, that is why we have CMP8 and MMV8 which might take twice as much time as CMP4 and MMV4. However, we assume that each 4byte operations take same time and similar is also for 8byte operations.

Insertion sort algo made easy for analysis



- **Worst Case analysis**

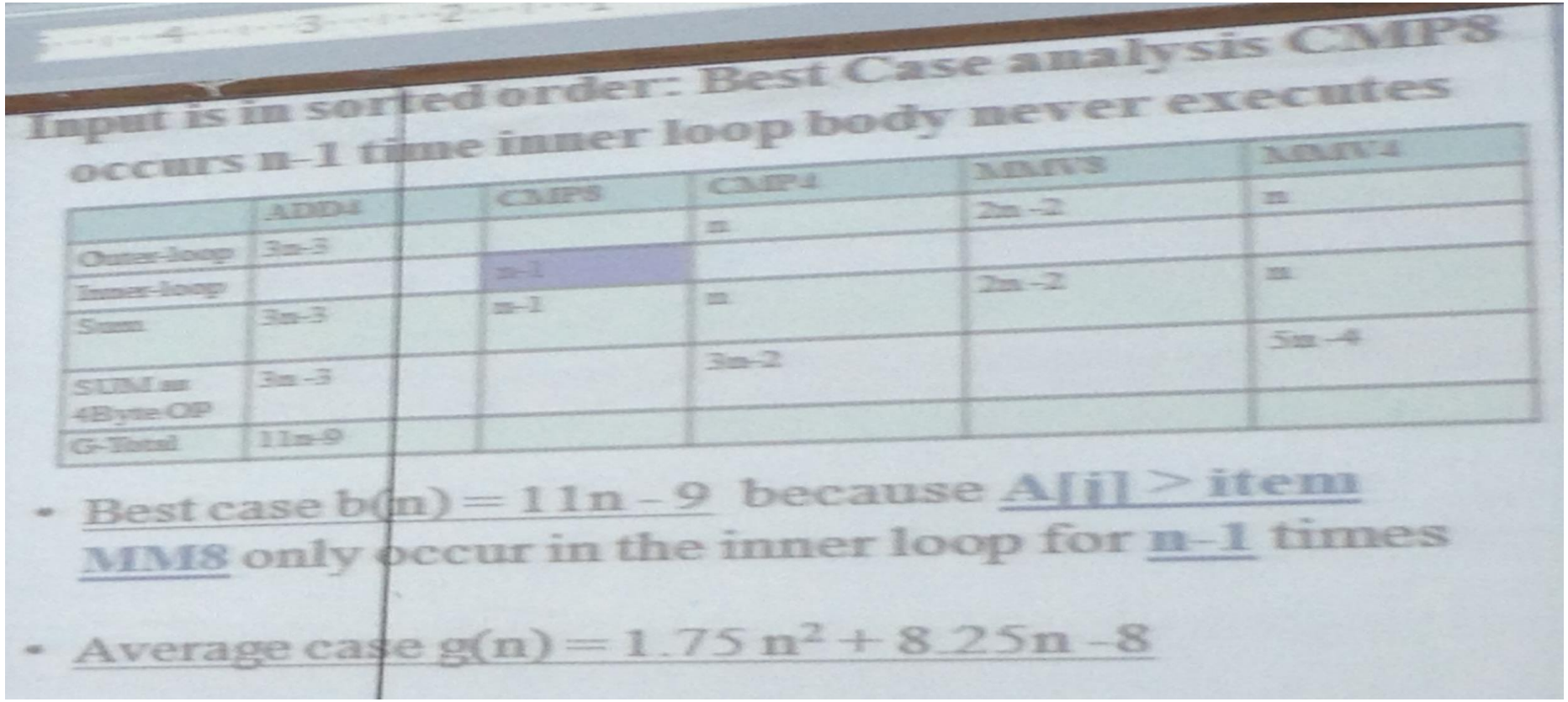


Input is in reverse order: Worst Case analysis all operations occur in every iteration of Insertion sort

	ADD4	CMP3	CMP4	MOV3	MOV4
Outer-loop	3n-3		n	2n -2	n
Inner-loop	n^2-n	$0.5n^2-0.5n$	$0.5n^2-0.5n$	$0.5n^2-0.5n$	
Sum	n^2+2n-3	$0.5n^2-0.5n$	$0.5n^2+0.5n$	$0.5n^2+1.5n-2$	n
SUM as 4Byte OP	n^2+2n-3		$1.5n^2-0.5n$		$n^2+4.0n-4$
G-Total	$3.5n^2+5.5n-7$				

- Worst case $w(n) = 3.5n^2 + 5.5n - 7$
- **Input is in desired order: Best Case analysis operations in inner loop body never occur.**

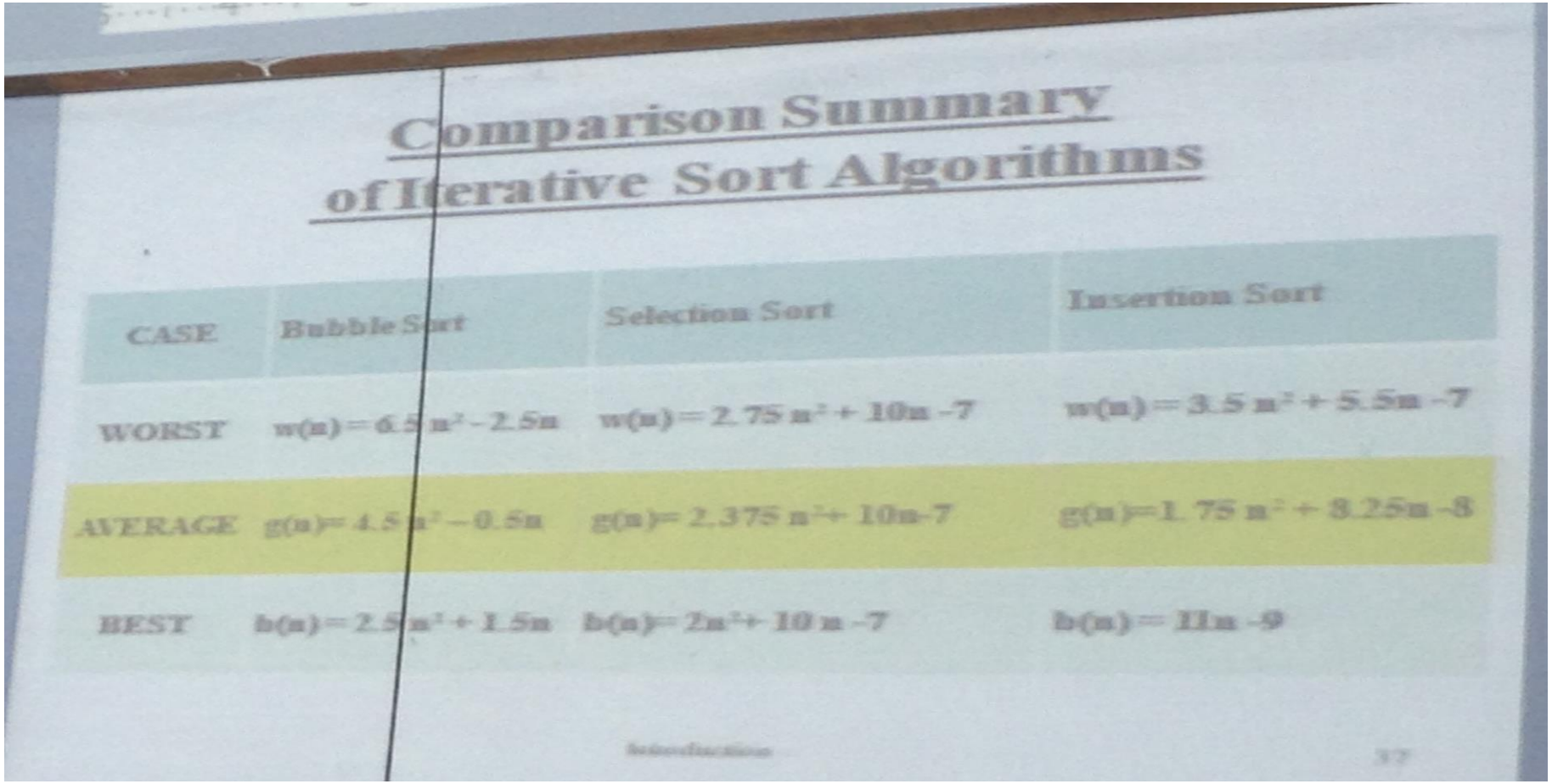
- **Best Case analysis**



Input is in sorted order: Best Case analysis CMP8 occurs n–1 time inner loop body never executes

	ADD4		CMP8	CMP4	MM8	MM4
Outer-loop	3n-3			n	2n-2	n
Inner-loop			n-1			
Sum	3n-3		n-1	n	2n-2	n
SUM as 4Byte OP	3n-3			3n-2		5n-4
G-Total	11n-9					

- Best case $b(n) = 11n - 9$ because A[j] > item MM8 only occur in the inner loop for n–1 times
- Average case $g(n) = 1.75\,n^2 + 8.25n - 8$

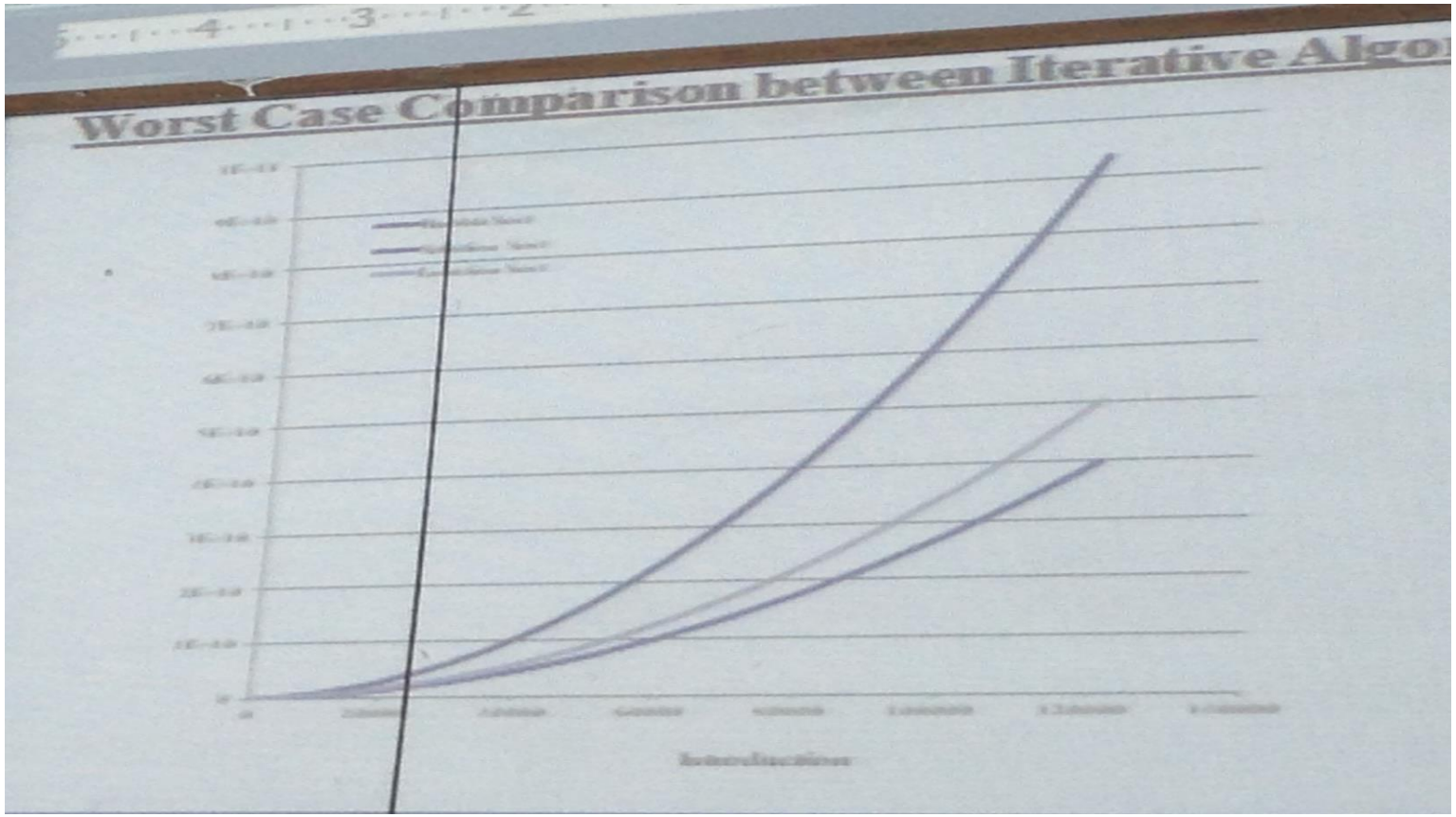


Comparison Summary of Iterative Sort Algorithms

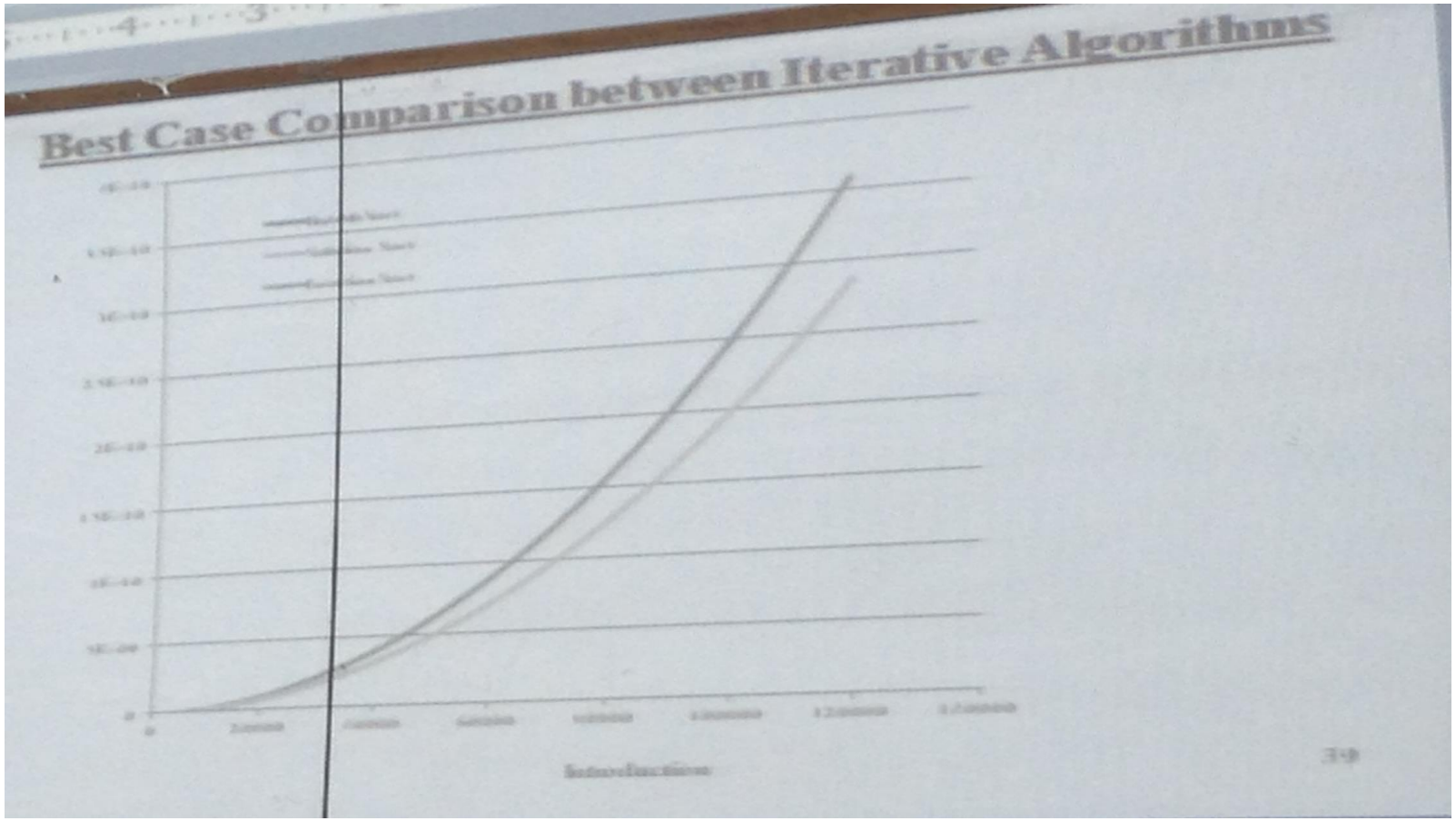
CASE	Bubble Sort	Selection Sort	Insertion Sort
WORST	$w(n) = 6.5n^2 - 2.5n$	$w(n) = 2.75n^2 + 10n - 7$	$w(n) = 3.5n^2 + 5.5n - 7$
AVERAGE	$g(n) = 4.5n^2 - 0.5n$	$g(n) = 2.375n^2 + 10n - 7$	$g(n) = 1.75n^2 + 8.25n - 8$
BEST	$b(n) = 2.5n^2 + 1.5n$	$b(n) = 2n^2 + 10n - 7$	$b(n) = 11n - 9$

Comparison b/w Bubble sort, Selection sort and Insertion sort (Graph)

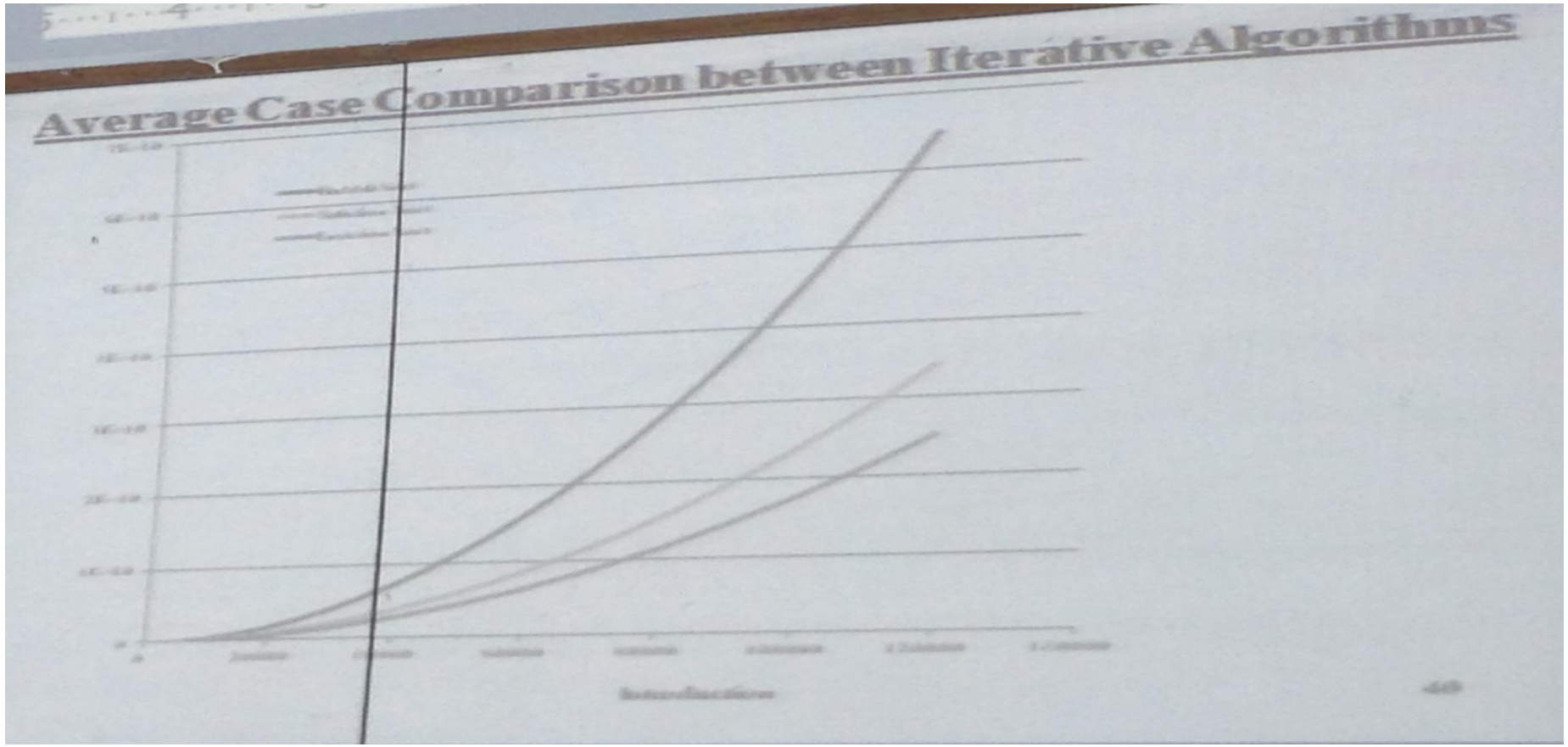
- **Worst Case comparison**



- **Best Case Comparison**



- **Average Case Comparison**

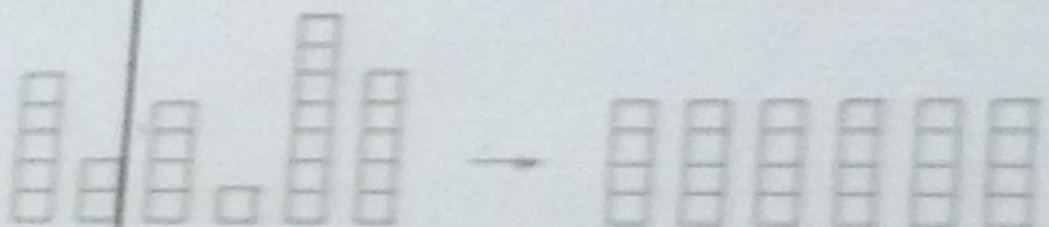


Computation Model for Algorithm Analysis

- Algorithms are meant to be understood by human that is why flexible description is permitted.
- But on the other end algorithms are to be run on machines, so we have to establish one common mathematical computation model to say any thing meaningfully about algorithms we analyze. Such model is to be a reasonable abstraction of a standard generic single-processor computer.
- **We will call this model a Random Access Machine or RAM. It is an idealized machine with an infinitely large random access memory. It executes instructions one-by-one & not parallel. Each instruction performs primitive operation on two values in memory.**

Problem :

Bob likes to play building blocks. He can stack the building blocks into many heap of different heights. Bob is very happy to tell his sister Alice: "You see, I put the wall up!" The sister retorted: "There is none, the real wall should have the same height, you make each stack of blocks of same height." After some thought, Bob felt that his sister was right, so he decided to re-stack the building blocks. But the little Bob is too lazy, he wants to move the smallest number of building blocks to make each stack of the same height. Can you help him?



Exercises

Input: There are several groups of measurements. The beginning of each group of tests is an integer **n**, representing the number of small pile stacks. Then the next line will contain **n** integers, representing the height of the different building heap. You can assume that there is a gap of 1 and 4. The total number of blocks can be divisible by the stack, so we can pile up the same height of the stack. When **n = 0**, the input ends.

Output: For each group of measurements, first print out the number of the test, as shown in the output example. Then the "The minimum number of moves is k.", Where **k** represents the number of blocks that need to be moved at least so that all the blocks have the same height. Print a blank line after the output of each group.

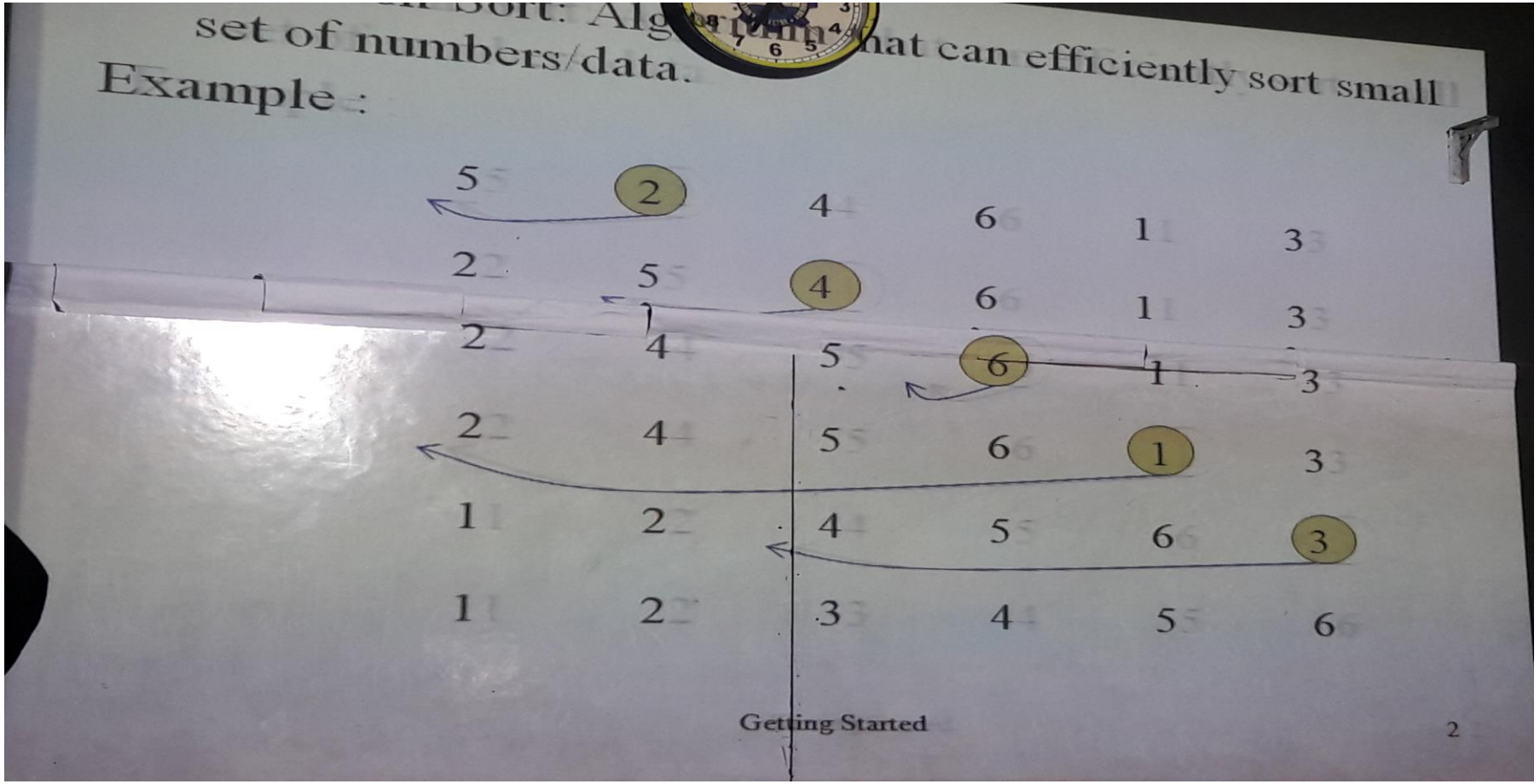
Exercises

Problem:

One bank, according to a reliable line, points out that one of the N coins in the last batch of banks is fake and the weight is different from the other coins. (The weight of the other coins is the same). At present the bank only a simple scales that can be used, can only measure the left or right of the items who are heavier.

Devise an algorithm

Insertion sort: algorithm that can efficiently sort small set of numbers/data

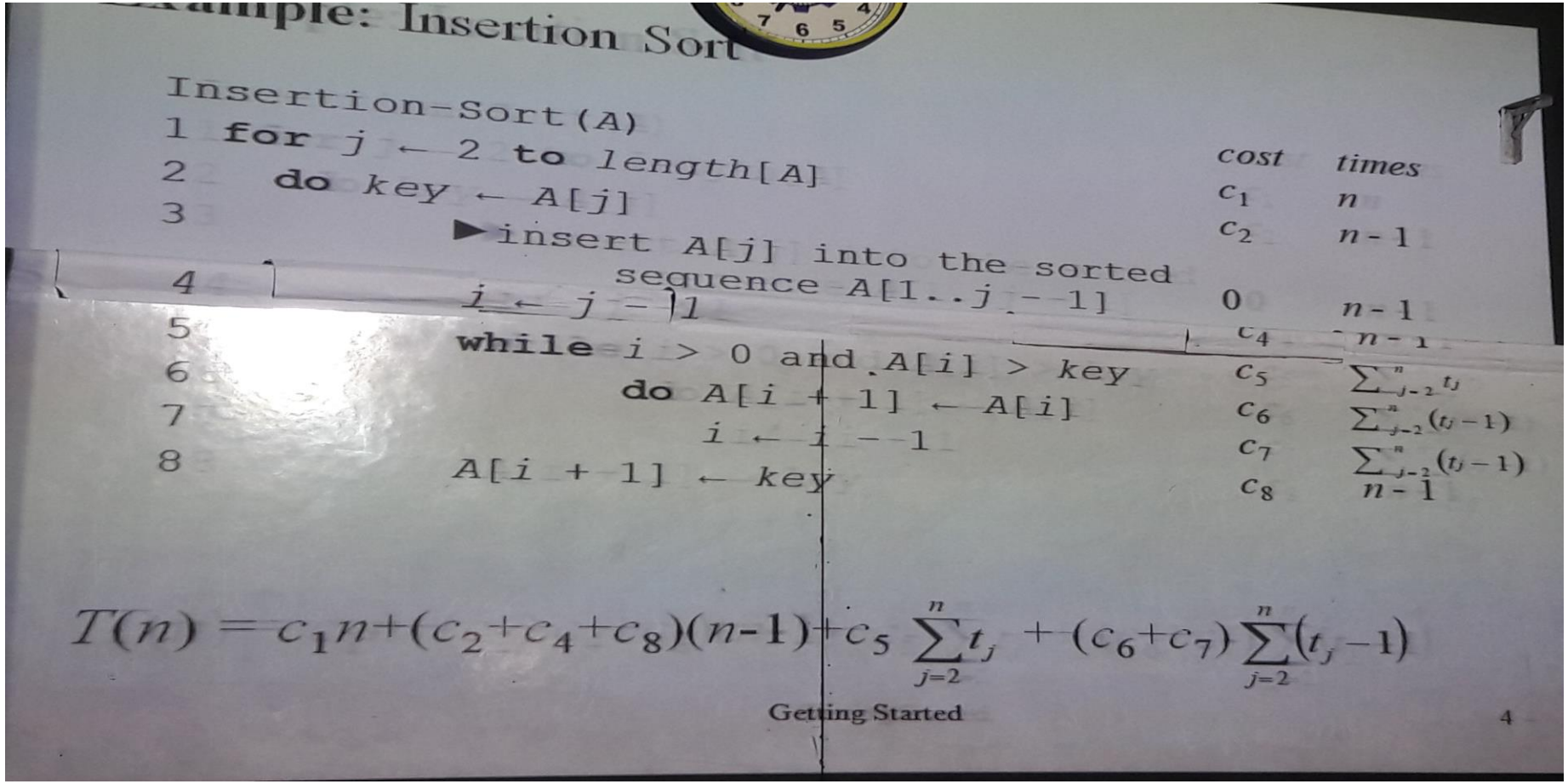


2.2 Analyzing Algorithms

RAM: *Random-access machine*, Performing memory access on this machine takes only a single unit of time and the instructions are executed sequentially one by one.

Running time : The total number of steps performed, expressed as a function of input size .

Example: Insertion Sort



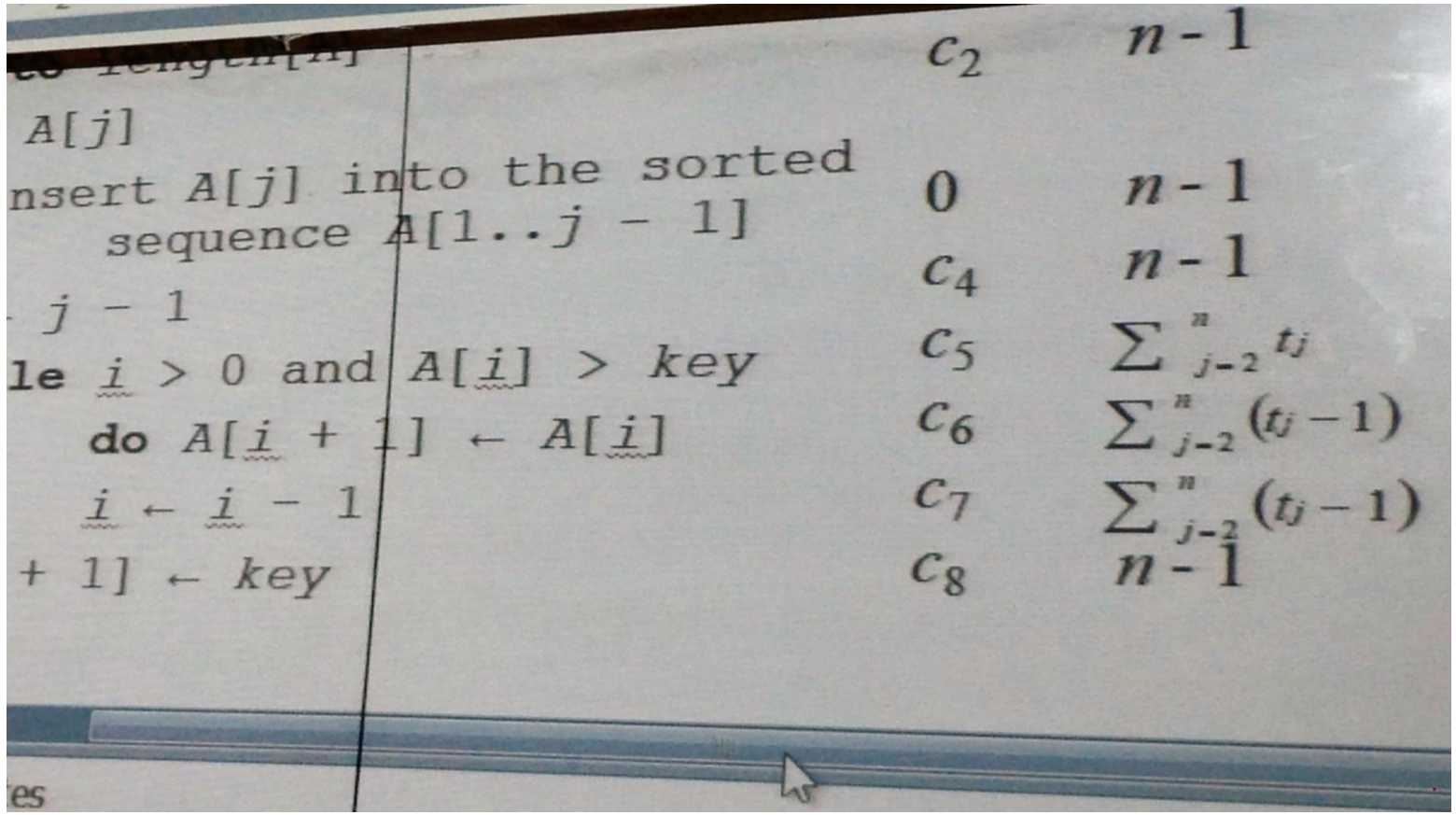
ample: Insertion Sort

```
Insertion-Sort(A)
1 for j ← 2 to length[A]
2    do 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                do A[i + 1] ← A[i]
7                   i ← i - 1
8          A[i + 1] ← key
```

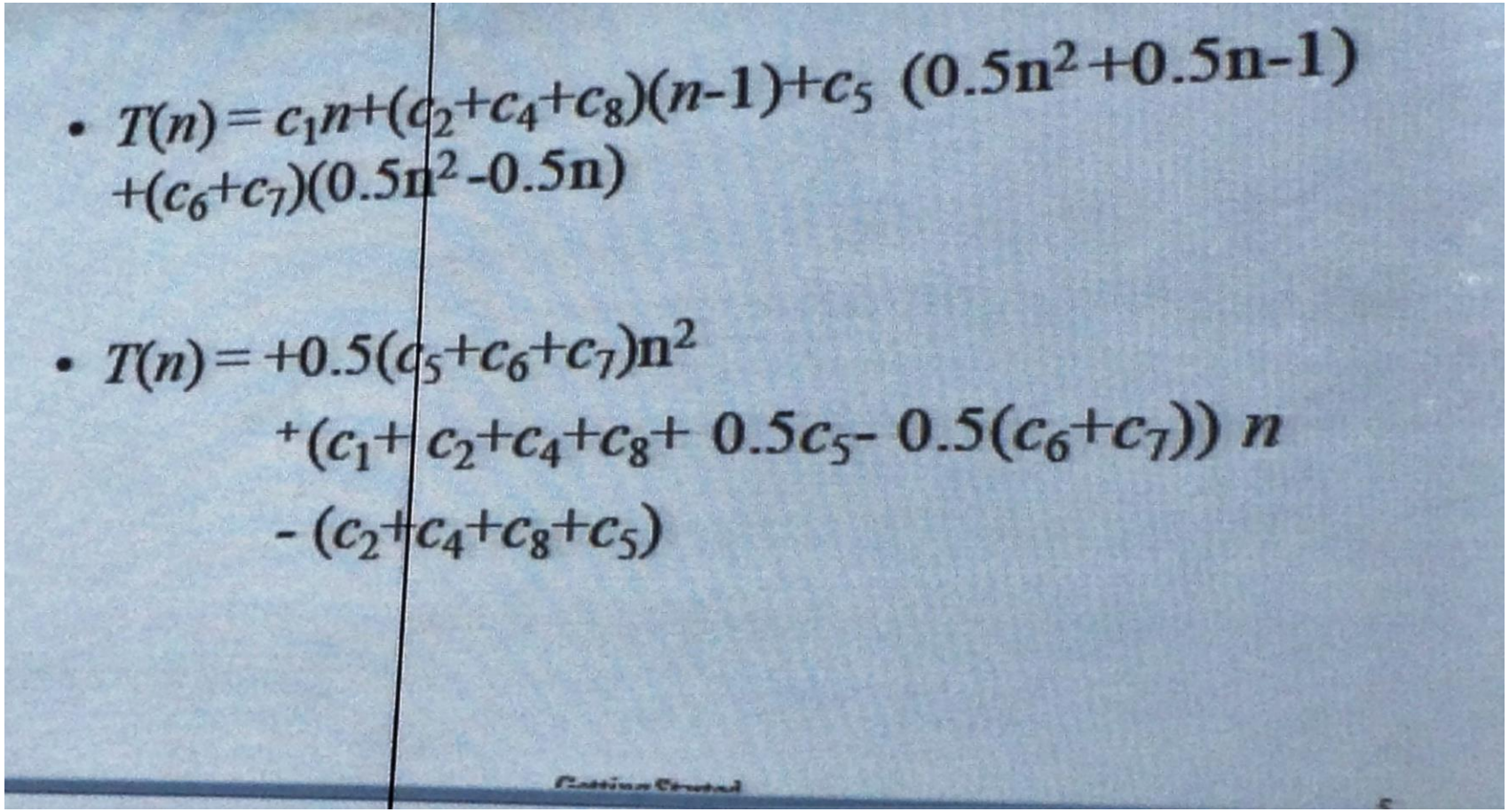
cost	times
c_1	n
c_2	$n-1$
0	$n-1$
c_4	$n-1$
c_5	$\sum_{j=2}^{n} t_j$
c_6	$\sum_{j=2}^{n} (t_j - 1)$
c_7	$\sum_{j=2}^{n} (t_j - 1)$
c_8	$n-1$

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

Getting Started 4


A[j]
nsert A[j] into the sorted
sequence A[1..j - 1]
j - 1
le i > 0 and A[i] > key
do A[i + 1] ← A[i]
i ← i - 1
+ 1] ← key
es
c_2 $n-1$
0 $n-1$
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 $n-1$

Growth of function



Best-case: Each $t_i = 1$. Enter sorted A

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

$= \Theta(n)$ ***(rate of growth, order of growth)***

Worst-case: (upper bound)

Each $t_i = j$.

$T(n) = k_1n^2 + k_2n + k_3$

$= \Theta(n^2)$

The best is when the input has been sorted, the time spent with n is proportional to n. The worst case scenario is the opposite of the best case and the time it takes is proportional to n^2.

Average-case: (Expected running time)

Each $t_i = j/2$

$T(n) = t_1 n^2 + t_2 n + t_3$

$= \Theta(n^2)$ ***(rate of growth, order of growth)***

On the average, insertion sort time is proportional to n^2.

According to the probability, when the figure t_i forward inspection, the average will be j / 2 step forward will stop

The best time to stop, the worst case is j step will stop.

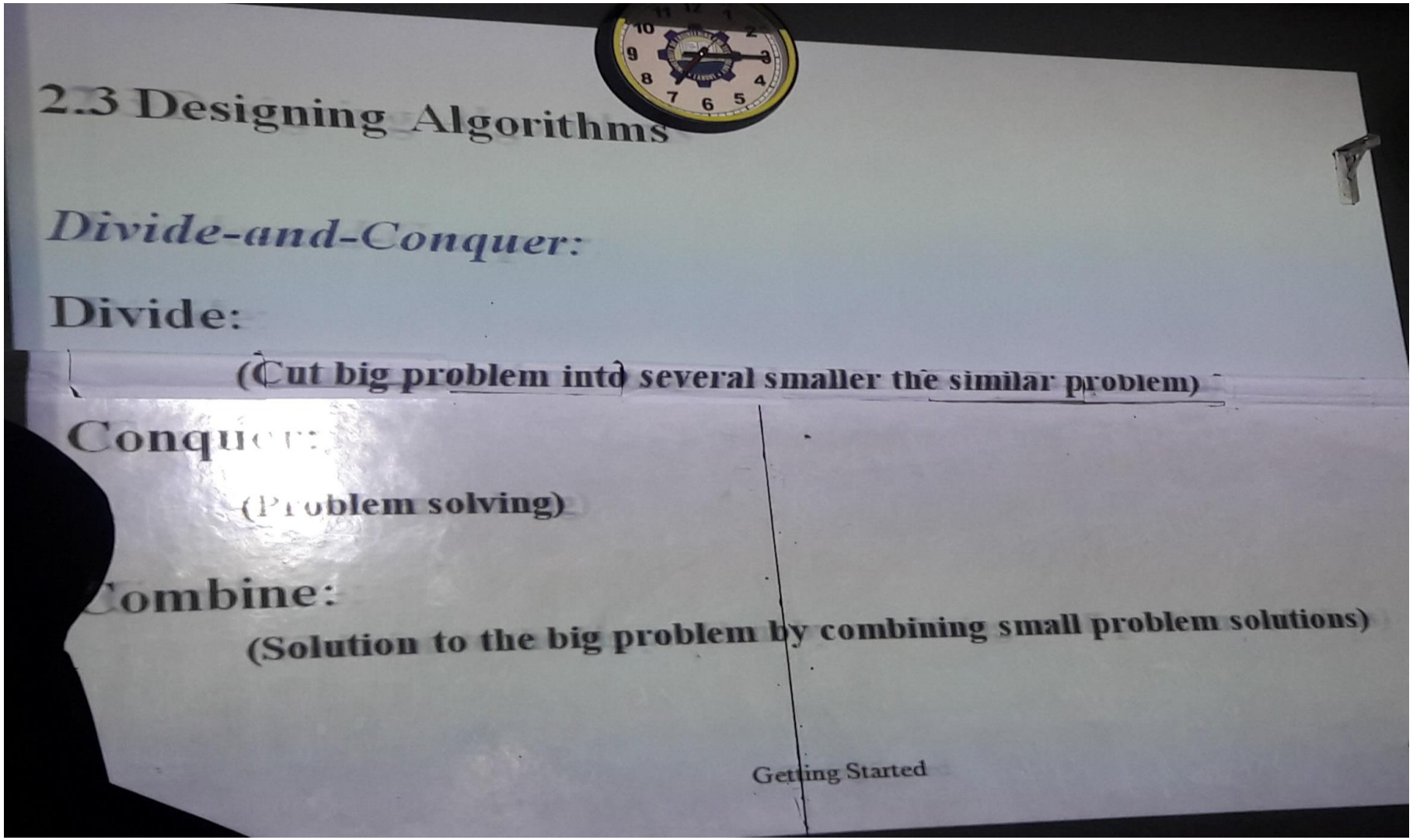
Exercises

2.2-1
Express the function $n^3/1000 - 100n^2 - 100n + 3$ in terms of Θ-notation.

2.2-2
Consider sorting n numbers stored in array A by first finding the smallest element of A and exchanging it with the element in $A[1]$. Then find the second smallest element of A, and exchange it with $A[2]$. Continue in this manner for the first $n - 1$ elements of A. Write pseudocode for this algorithm, which is known as *selection sort*. What loop invariant does this algorithm maintain? Why does it need to run for only the first $n - 1$ elements, rather than for all n elements? Give the best-case and worst-case running times of selection sort in Θ-notation.

2.2-3
Consider linear search again (see Exercise 2.1-3). How many elements of the input sequence need to be checked on the average, assuming that the element being searched for is equally likely to be any element in the array? How about in the worst case? What are the average-case and worst-case running times of linear search in Θ-notation? Justify your answers.

2.2-4
How can we modify almost any algorithm to have a good best-case running time?

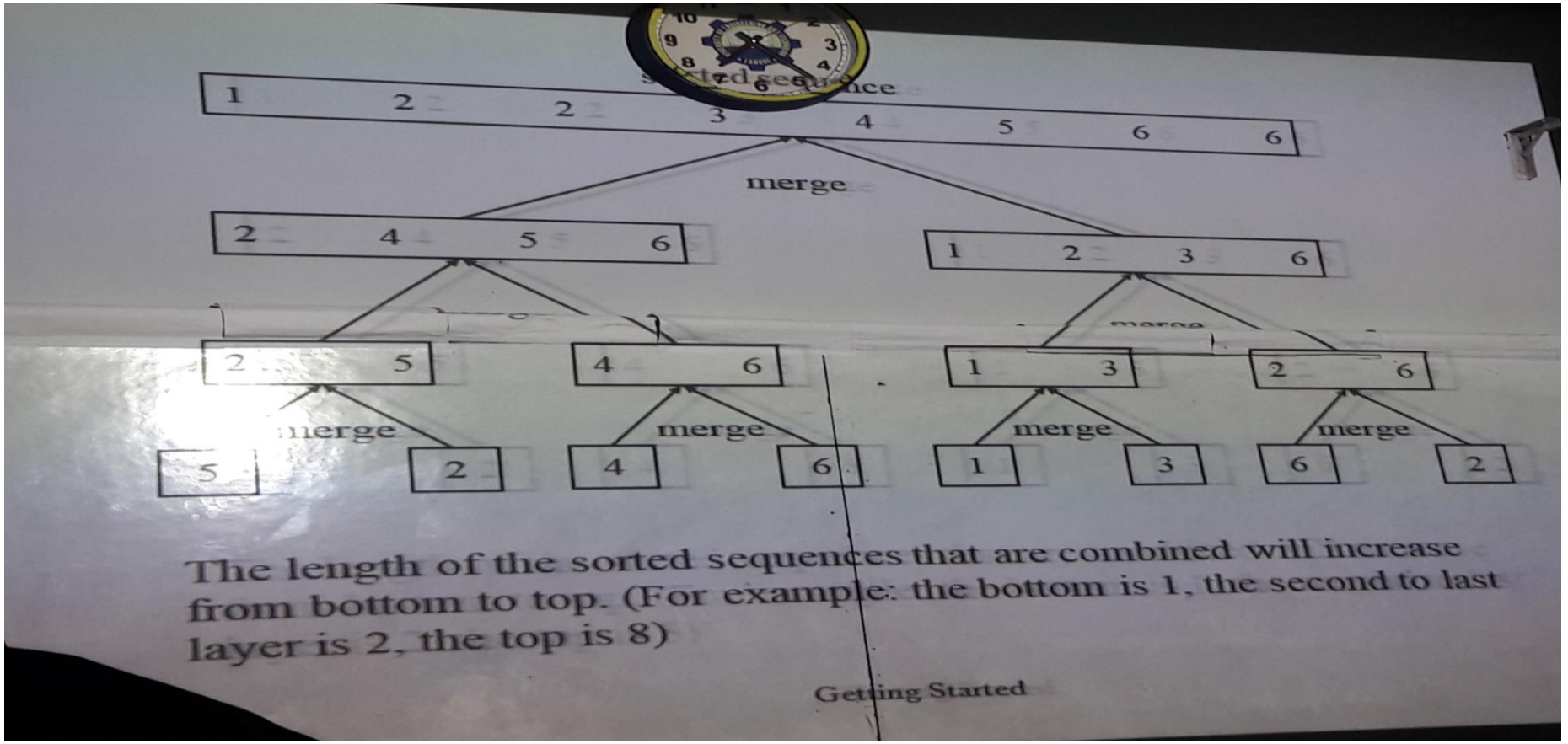

2.3 Designing Algorithms
Divide-and-Conquer:
Divide:
(Cut big problem into several smaller the similar problem)
Conquer:
(Problem solving)
Combine:
(Solution to the big problem by combining small problem solutions)
Getting Started

Example: Merge Sort

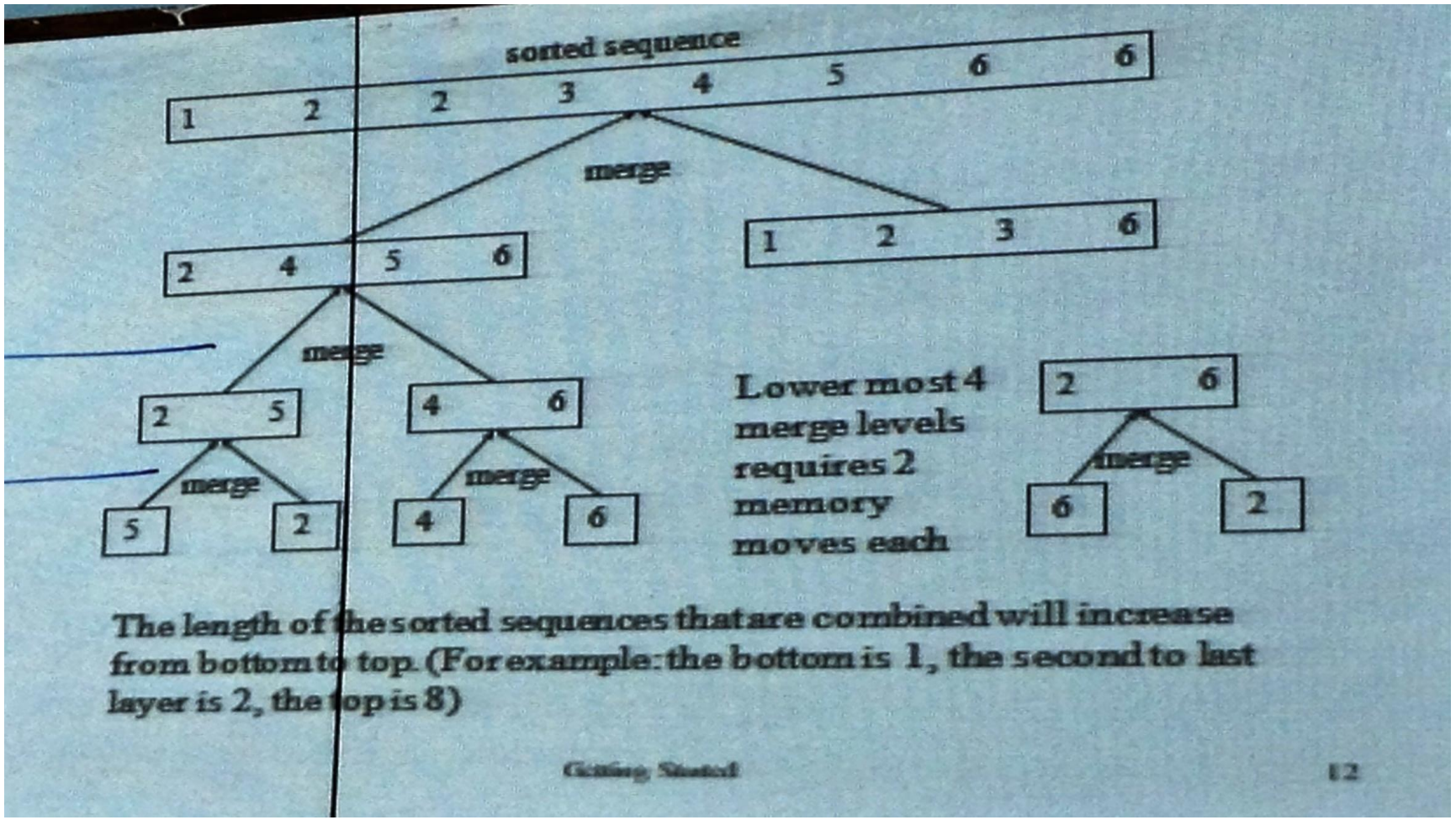
```
MERGE-SORT(A, p, r)
1  if p < r
2     then q ← ⌊(p + r)/2⌋
3          MERGE-SORT(A, p, q)
4          MERGE-SORT(A, q + 1, r)
5          MERGE(A, p, q, r)
```

Merge sort is a classic example of divide-and-conquer.
To sort a set of numbers, it will be cut into several small sets until it is cut into a size which is quickly sort-able.
Then merge the answer (Merge), the solution of small problems into the solution to the big problem.

Sorted Sequence



The length of the sorted sequences that are combined will increase from bottom to top. (For example: the bottom is 1, the second to last layer is 2, the top is 8)



The length of the sorted sequences that are combined will increase from bottom to top. (For example: the bottom is 1, the second to last layer is 2, the top is 8)

Merge Sor

The merging pattern

- The lowermost sorted length is 1 (we'll have a bunch of sorted sequences with length 1)
- The second best sorted length is 2 (we will have a bunch of sorted sequences that ~~are 2 in length~~)
- The next best k-th level ordered length is 2^k
- And so on, assuming N numbers, the merging of layers will end at the layer = log (N)

Analysis: (*recurrence*)

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 2T(n/2) + \Theta(n) & \text{if } n > 1 \end{cases}$$

$$= \Theta(n \log n)$$

The next chapter will teach Θ and the definition of other symbols, Further chapters teach how to analyze recurrence to get the result of T (n) = Θ (n log n)

Exercises

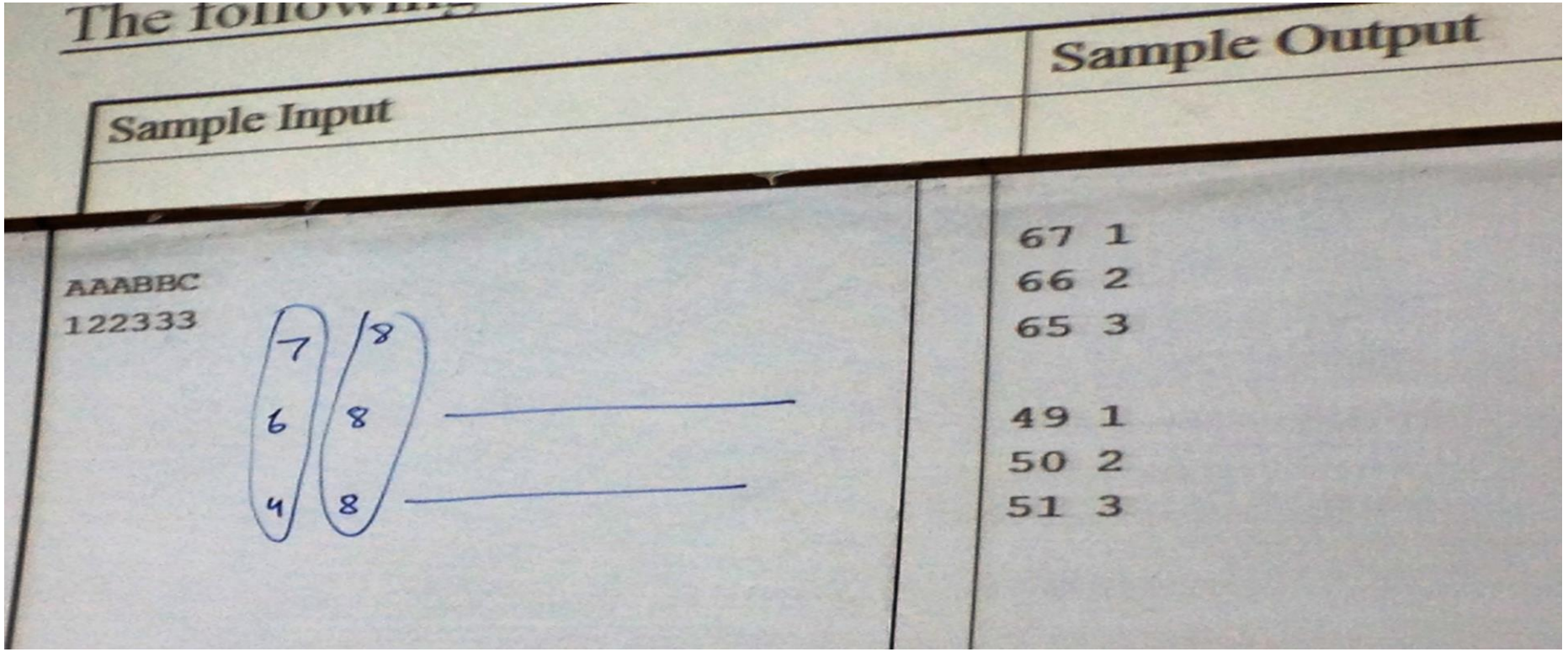
Problem 1:

Given a line of text, please help me to list the frequency of appearance of ASCII characters. You can assume that the first 32 characters of ASCII and the last 128 characters do not appear. The end of each line of text may end with '\n' and '\r', but do not take those characters into account.

INPUT: Will have a few lines of text. The maximum length of each line will be 1000. Enter the end of the file to mark end of input.

OUTPUT: For each line of input, the ASCII value of the ASCII character is printed according to the frequency of occurrence and the location at which the character appears (first print a low frequency character). Print a blank line between each output. If two ASCII characters have the same frequency, then character with smaller ASCII value has print priority.

The following is an example of input and output



The following

Sample Input	Sample Output
AAABBC 122333	67 1 66 2 65 3 49 1 50 2 51 3

7 8
6 8
4 8

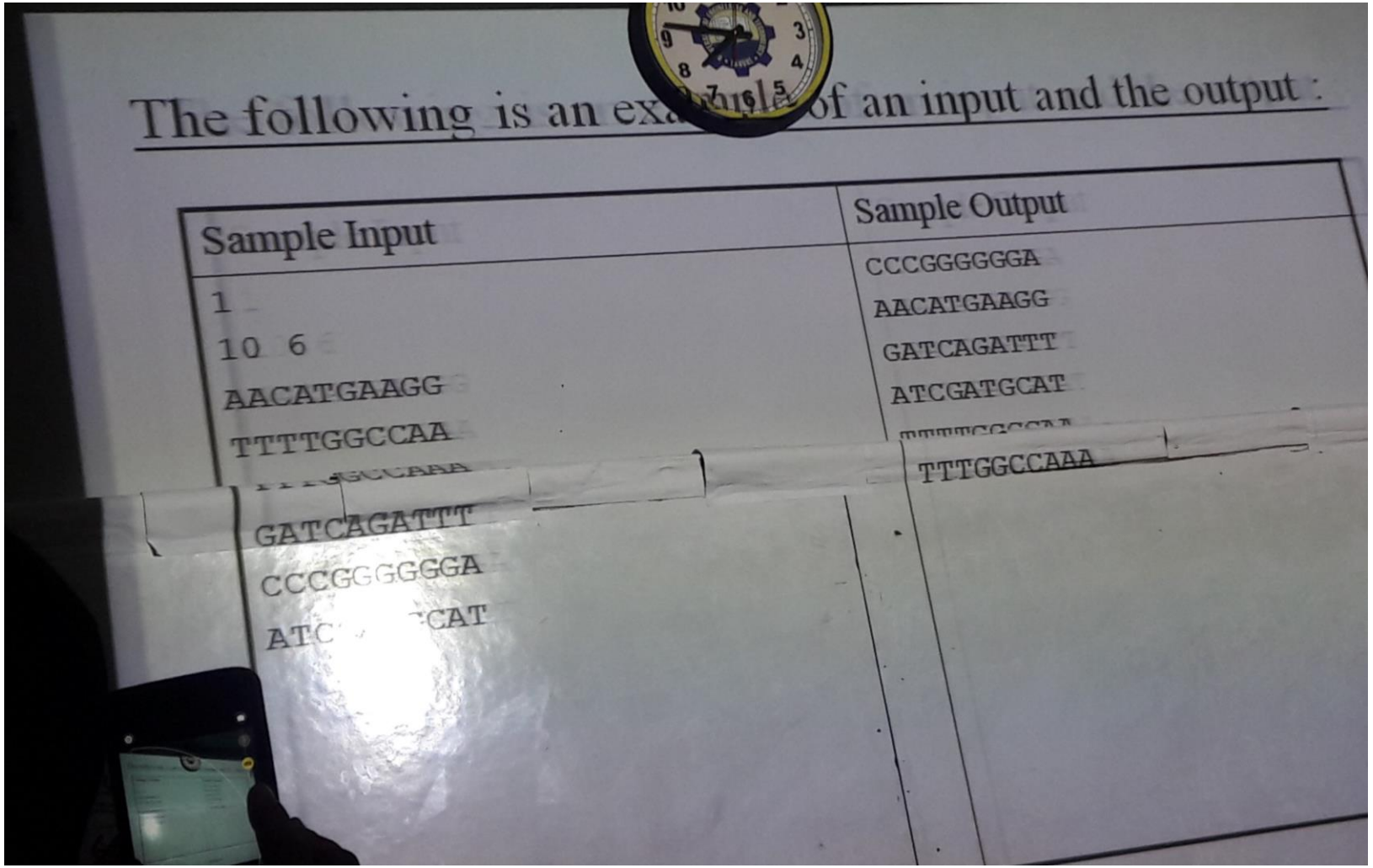
Problems:

The measure of the extent to which a sequence is "chaotic" is to figure out how many pairs of numbers are not in order. For example, in the sequence "DAABEC", the degree of chaos is 5 in accordance with the above measure because D is larger than the four letters to the right and E is also larger than one letter to the right. In fact, the degree of disorder algorithm is the number of inversion of this sequence. The sequence "AACEDGG" has only one inversion (E and D) which is almost ordered; however, "ZWQM" has six inversion (almost no sort, in fact exactly the opposite of sort). You are responsible for sorting a bunch of DNA sequences (the sequence consists of only four letters A, T, C, G). However, instead of sorting the sequences according to their lexical order, you are going to list them in order of "closest to the sorted ones" to "almost unordered ones," based on their "level of chaos." All sequences have same length.

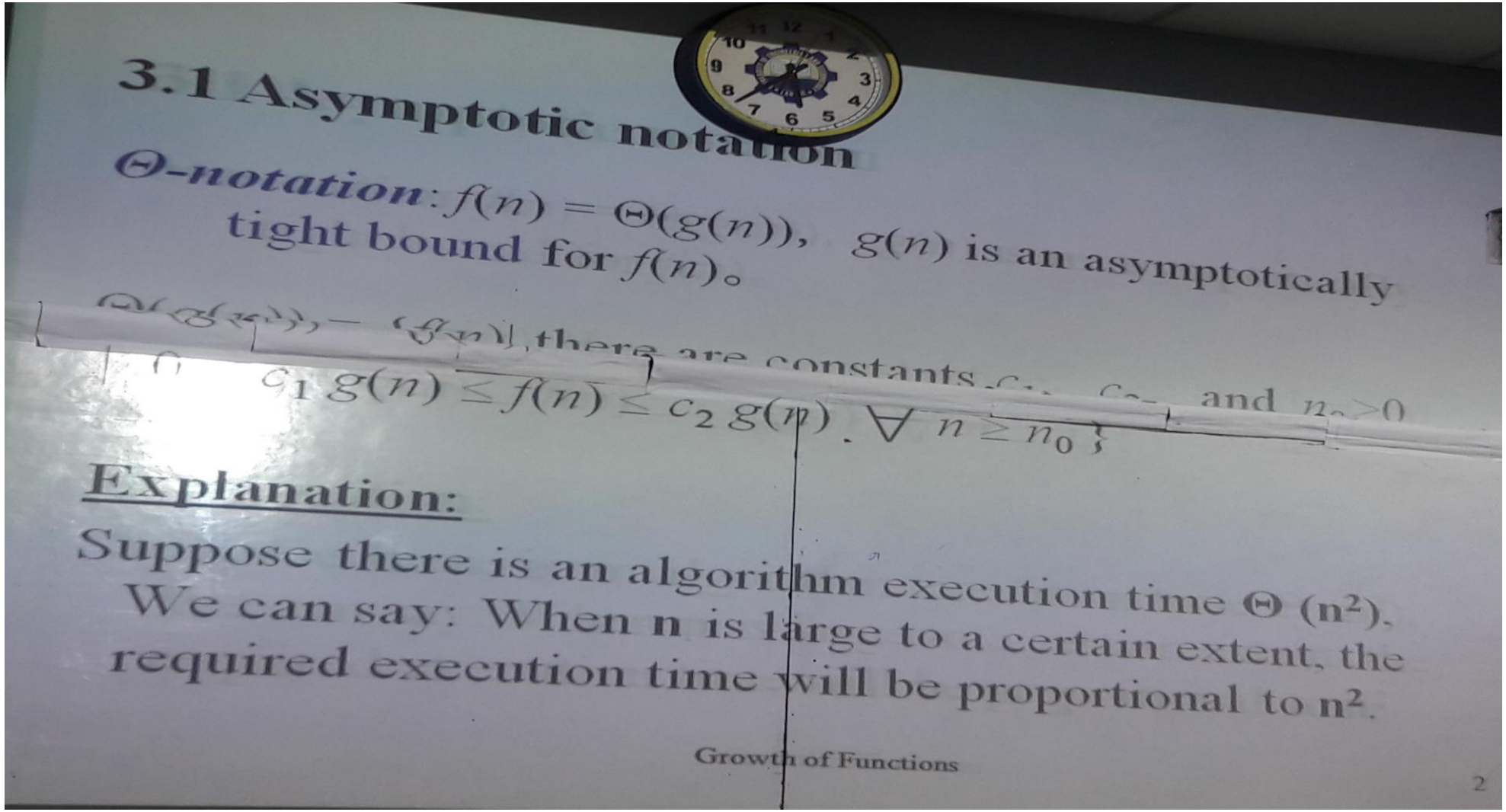
Exercises

INPUT: The first line is an integer M, then a blank line followed by M group test. There will be a blank line between each set of capital as interval. The first line of each group contains two positive integers: n ($0 < n \leq 50$) indicates the length of the sequence; m ($0 < m \leq 100$) indicates the total number of sequences. Next there will be m lines, each of which is a DNA sequence of length n.

OUTPUT: For each group of test/asset collected from the "best sorted" sequence to "almost no sorted" sequence. If the chaos of the two sequences is the same, first appear in the input file is priority.


The following is an exa of an input and the output :
Sample Input
1
10 6
AACATGAAGG
TTTTGGCCAA
GATCAGATTT
CCCGGGGGGA
ATC CAT
Sample Output
CCCGGGGGGA
AACATGAAGG
GATCAGATTT
ATCGATGCAT
TTTGGCCAAA

Asymptotic Notation



3.1 Asymptotic notation

Θ-notation: $f(n) = \Theta(g(n))$, $g(n)$ is an asymptotically tight bound for $f(n)$.

$\Theta(g(n)) = \{f(n)$: there are constants c_1, c_2 and $n_0 > 0$

$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n), \forall n \geq n_0\}$

Explanation:

Suppose there is an algorithm execution time $\Theta(n^2)$. We can say: When **n** is large to a certain extent, the required execution time will be proportional to n^2.

3.1 Asymptotic notation

Θ-notation: $f(n) = \Theta(g(n))$, $g(n)$ is an asymptotically tight bound for $f(n)$.

$\Theta(g(n)) = \{f(n) |$ there are constants c_1, c_2, and $n_0 > 0$ $| \ 0 \le c_1 g(n) \le f(n) \le c_2 g(n) \ \forall \ n \ge n_0 \}$

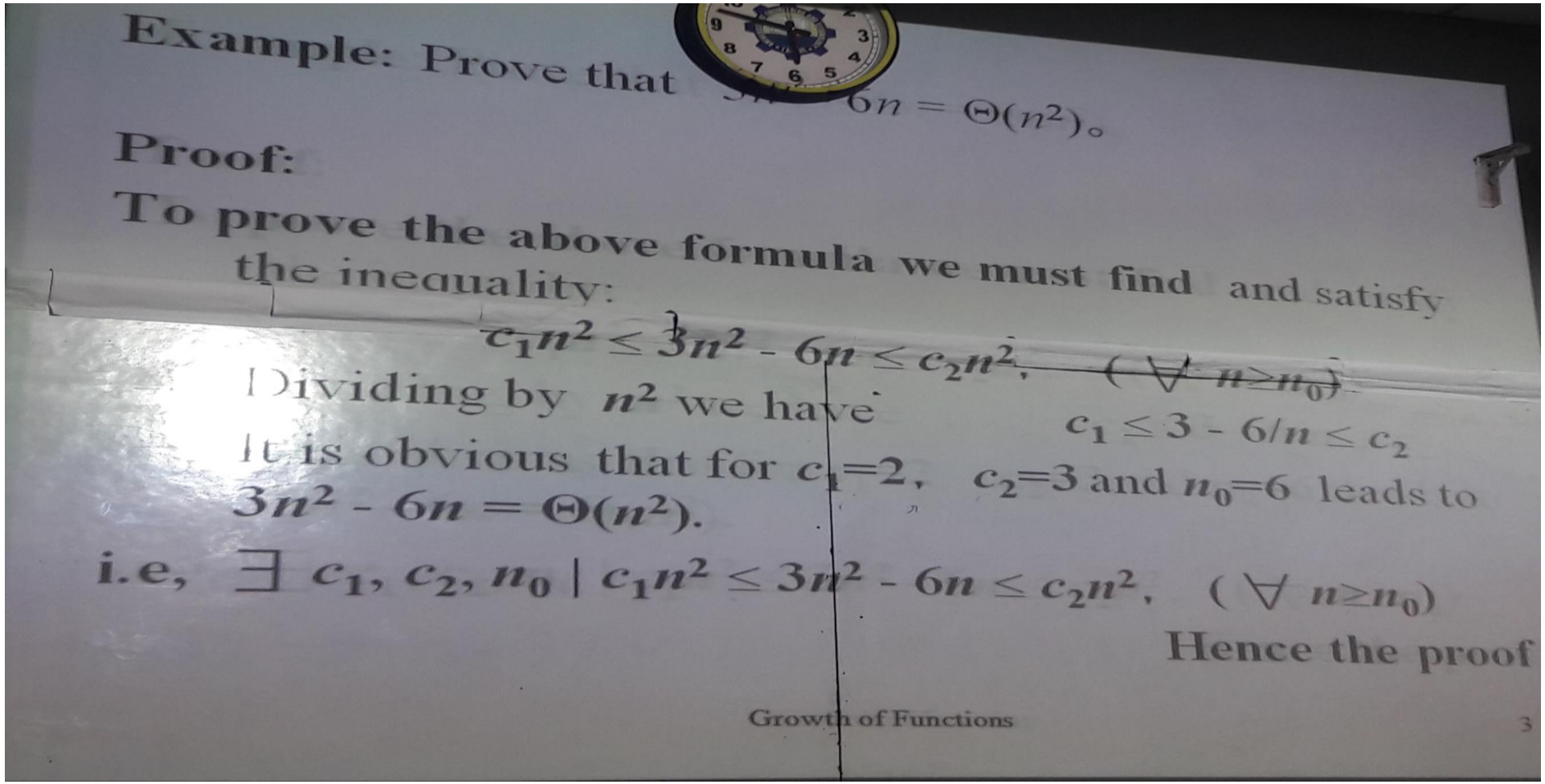
Explanation:

Suppose there is an algorithm execution time $\Theta(n^2)$.

We can say: When **n** is large to a certain extent, the required execution time will be proportional to n^2

notes

Example: Prove that $3n^2-6n = \text{theta}(n^2)$



Example: Prove that $6n = \Theta(n^2)$.

Proof:

To prove the above formula we must find and satisfy the inequality:

$$c_1 n^2 \le 3n^2 - 6n \le c_2 n^2, \quad (\forall\ n \ge n_0)$$

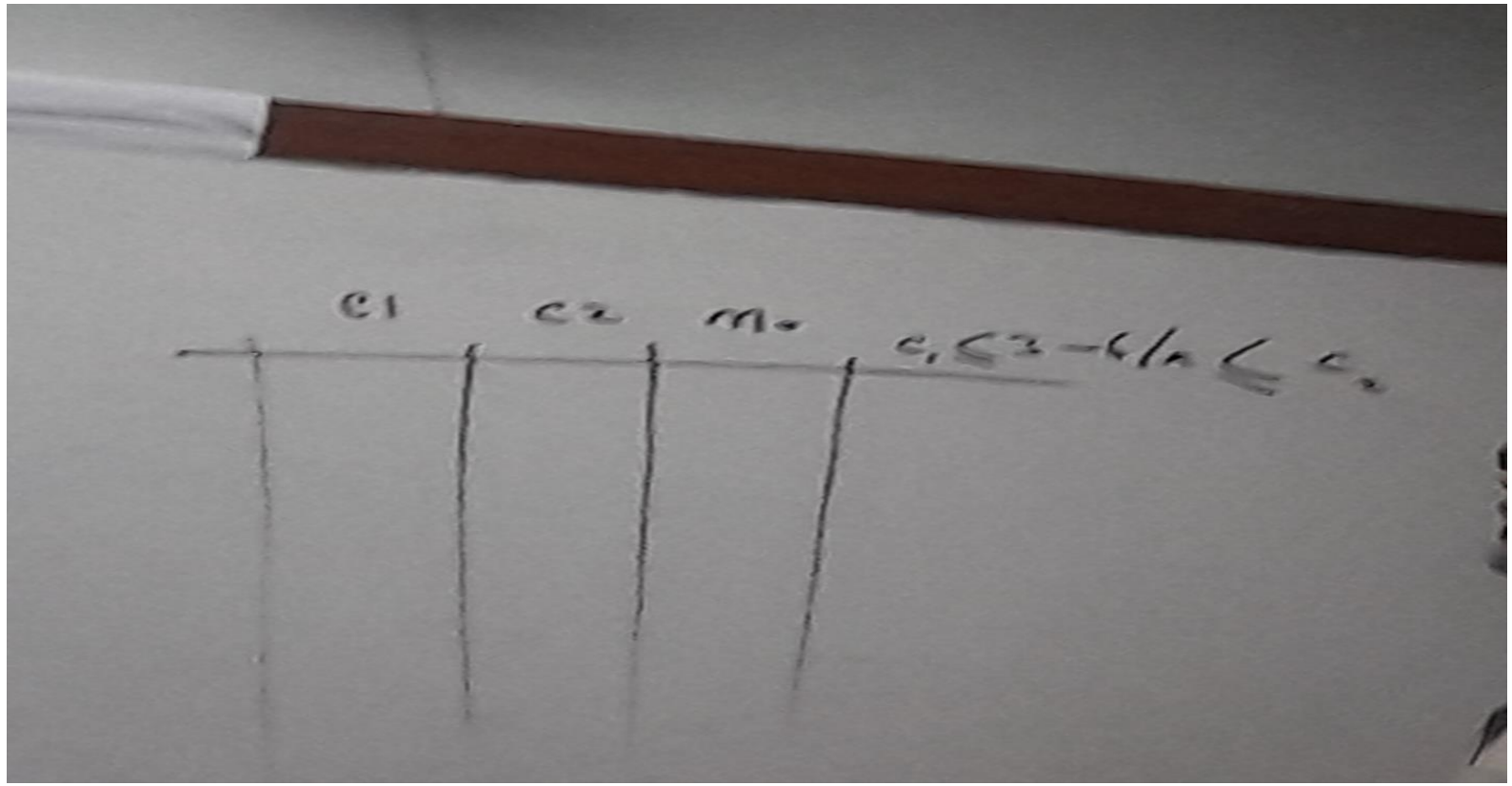
Dividing by n^2 we have $\quad c_1 \le 3 - 6/n \le c_2$

It is obvious that for $c_1=2$, $c_2=3$ and $n_0=6$ leads to $3n^2 - 6n = \Theta(n^2)$.

i.e, $\exists\ c_1, c_2, n_0 \mid c_1 n^2 \le 3n^2 - 6n \le c_2 n^2, \quad (\forall\ n \ge n_0)$

Hence the proof

Growth of Functions 3


c1
c2
m
c1 ≤ 3 − c/a ≤ c2

CASE	Bubble Sort	Selection Sort	Insertion Sort
WORST	$w(n) = 6.5n^2 - 2.5n$	$w(n) = 2.75n^2 + 10n - 7$	$w(n) = 3.5n^2 + 5.5n - 7$

Example: Prove that $6.5n^2 - 2.5n = \Theta(n^2)$.

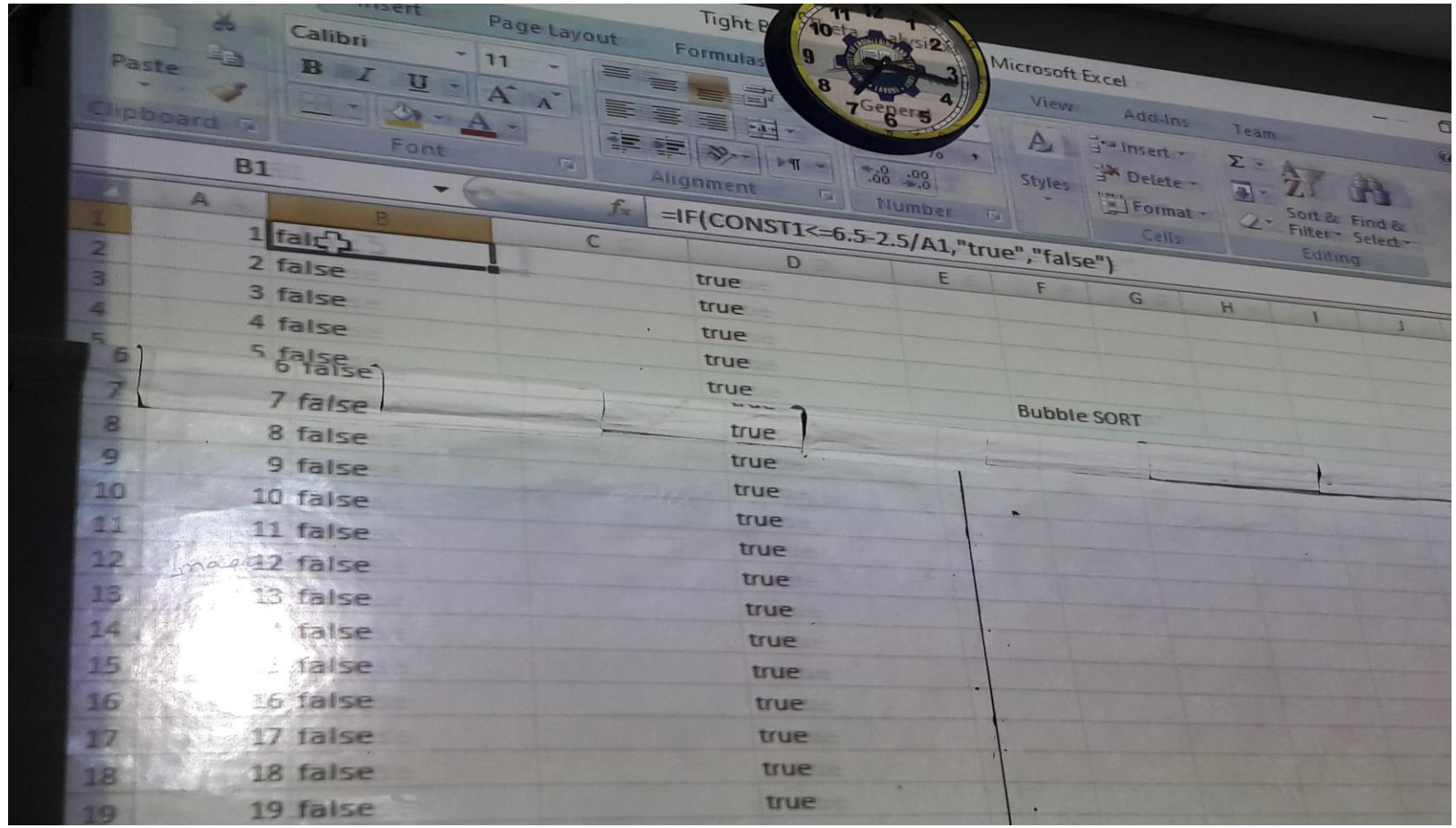
Proof: To prove it we must find c_1, c_2, and n_0 that satisfy : $c_1 n^2 \leq 6.5n^2 - 2.5n \leq c_2 n^2$, $\forall n \geq n_0$)

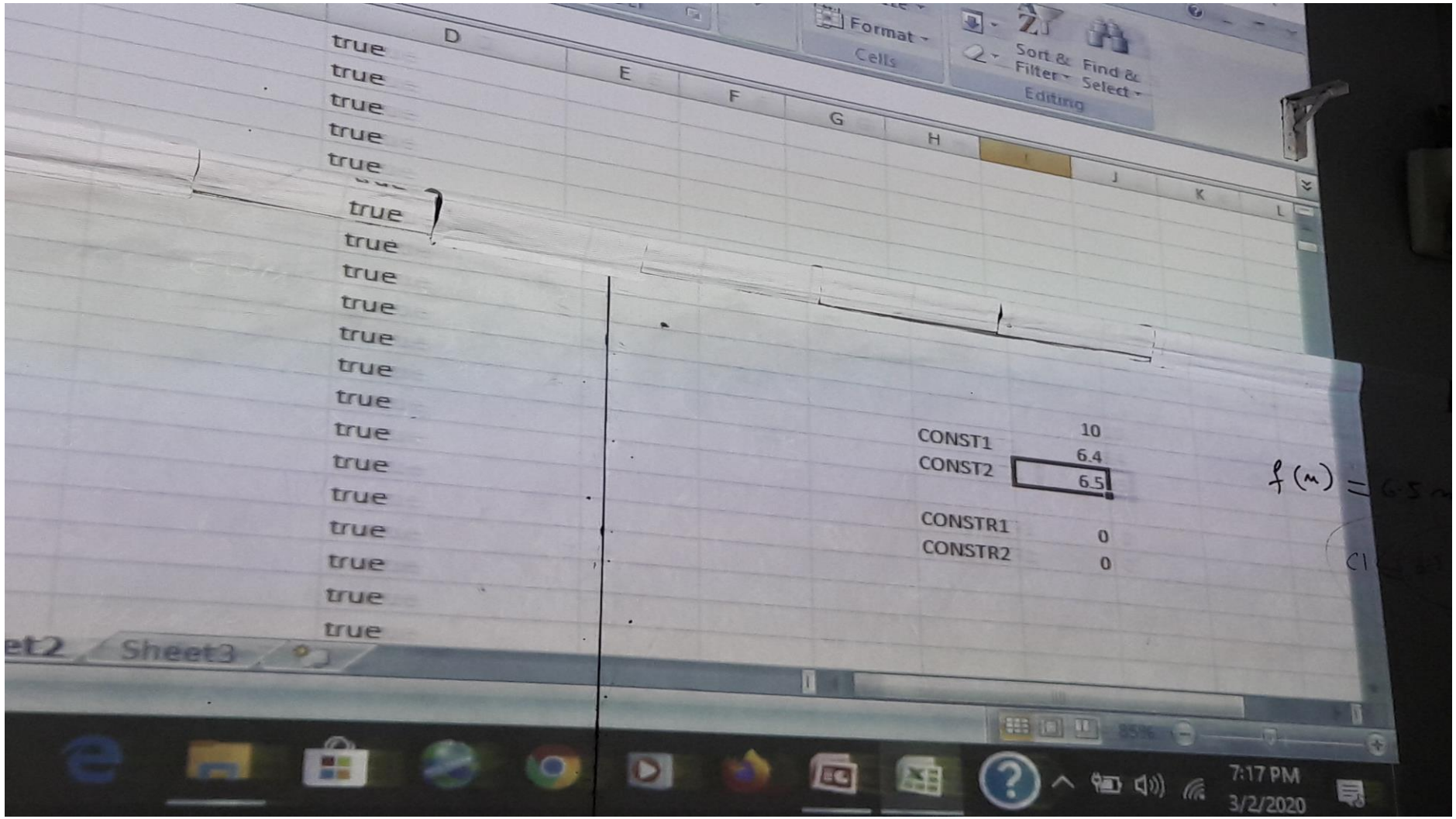
Dividing by n^2 we have.

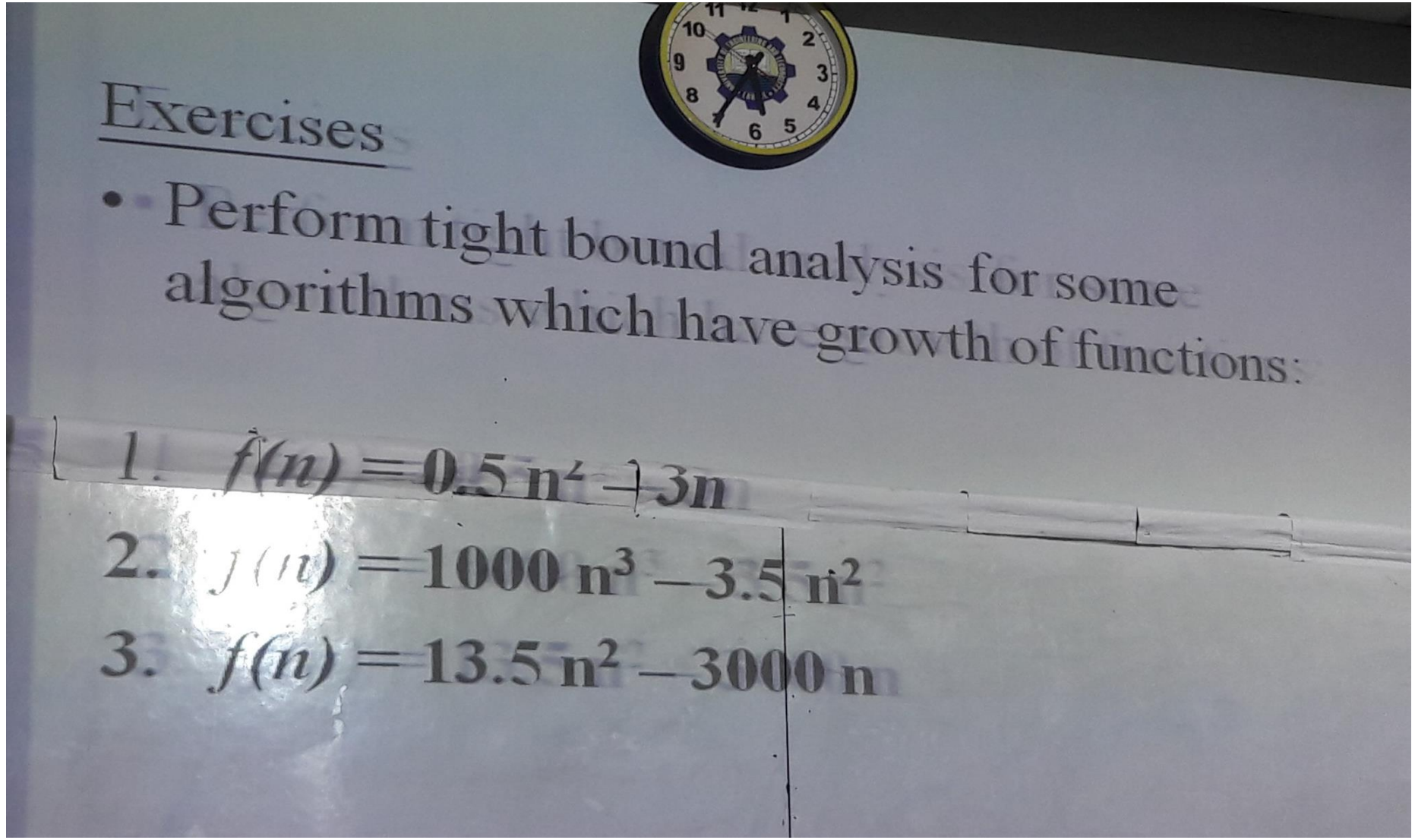
$$c_1 \leq 6.5 - 2.5/n \leq c_2$$

It is obvious that for $c_1 = 6$, $c_2 = 6.5$ and $n_0 = 5$ leads to $6.5n^2 - 2.5n = \Theta(n^2)$.

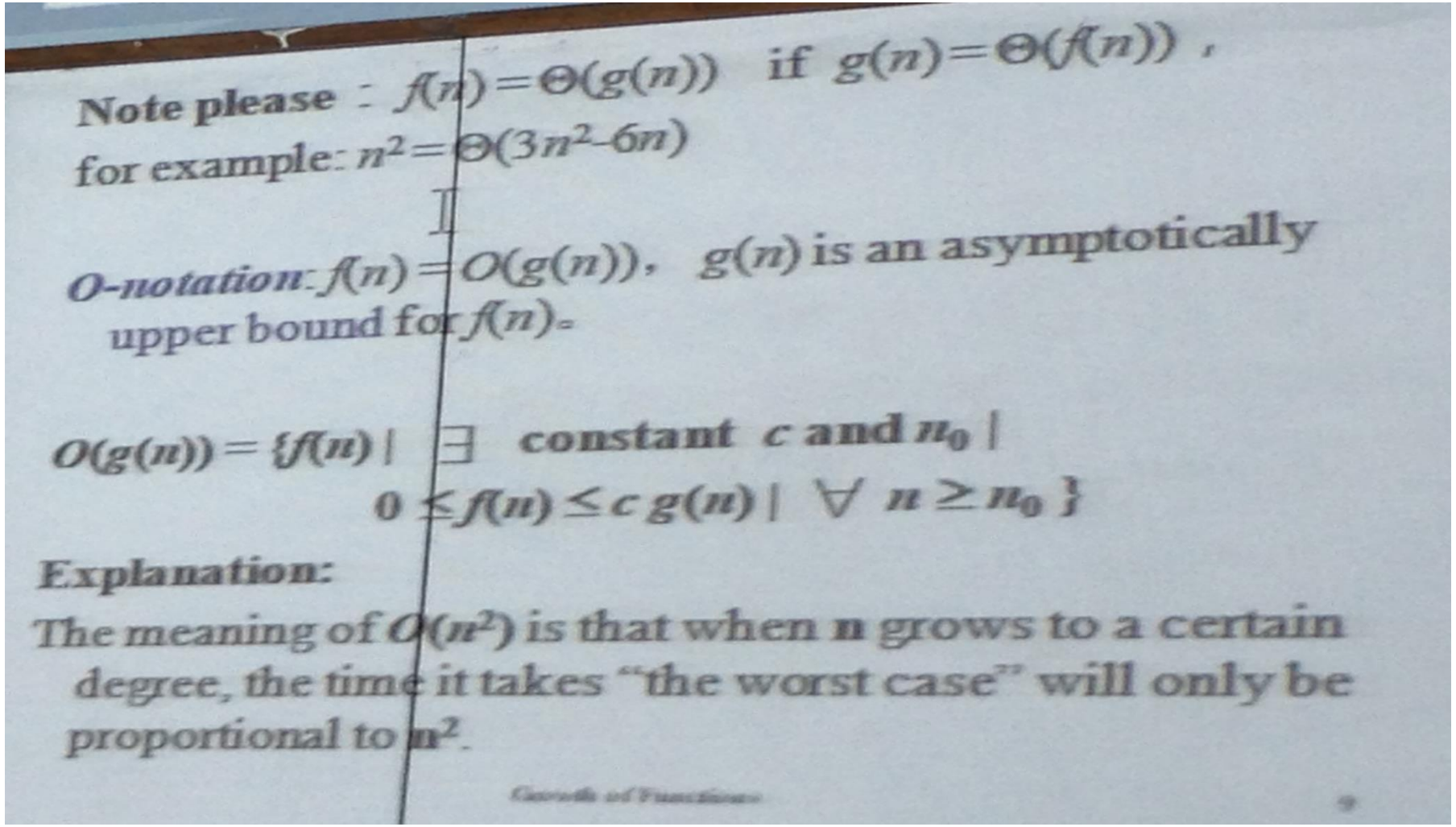
Hence the proof


Microsoft Excel
Page Layout
Formulas
Calibri
Paste
Clipboard
Font
Alignment
Number
Styles
Cells
Editing
Insert
Delete
Format
Sort & Filter
Find & Select
B1
=IF(CONST1<=6.5-2.5/A1,"true","false")
Bubble SORT
1 false
2 false
3 false
4 false
5 false
6 false
7 false
8 false
9 false
10 false
11 false
12 false
13 false
16 false
17 false
18 false
19 false
true


D
true
true
true
true
true
true
true
true
true
true
true
true
true
true
true
true
true
true
true
Format
Cells
Sort & Filter
Find & Select
Editing
E
F
G
H
J
K
L
CONST1 10
CONST2 6.4
6.5
CONSTR1 0
CONSTR2 0
f(x) =
Sheet3
7:17 PM
3/2/2020


Exercises
• Perform tight bound analysis for some algorithms which have growth of functions:
1. $f(n) = 0.5\,n^2 + 3n$
2. $f(n) = 1000\,n^3 - 3.5\,n^2$
3. $f(n) = 13.5\,n^2 - 3000\,n$

- **Big O notation (conditions)**



Note please : $f(n)=\Theta(g(n))$ if $g(n)=\Theta(f(n))$,
for example: $n^2=\Theta(3n^2-6n)$

$\Downarrow$

O-notation: $f(n)=O(g(n))$, $g(n)$ is an asymptotically upper bound for $f(n)$.

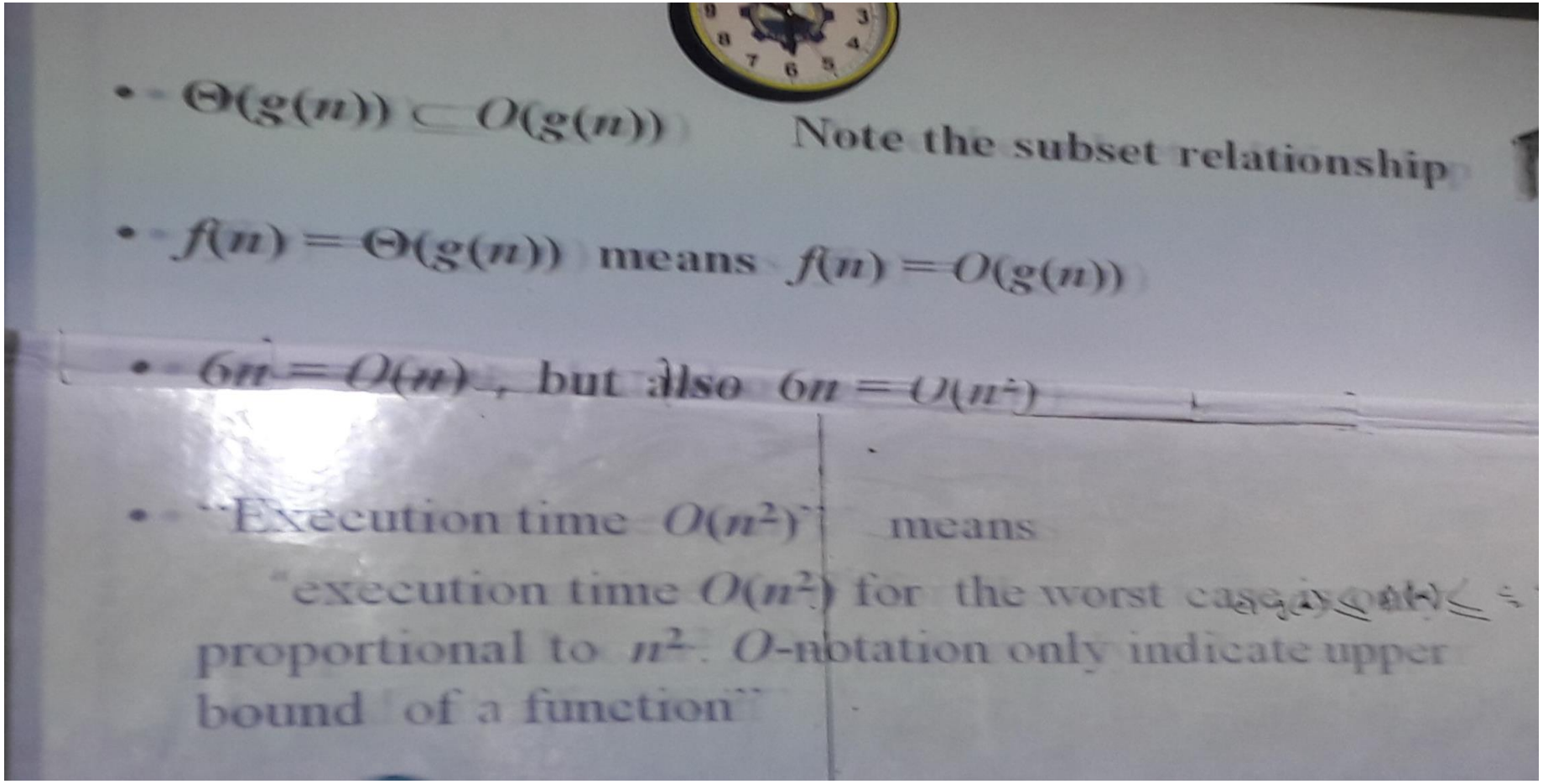
$$O(g(n)) = \{f(n) \mid \exists \text{ constant } c \text{ and } n_0 \mid 0 \le f(n) \le c\, g(n) \mid \forall\ n \ge n_0\}$$

Explanation:

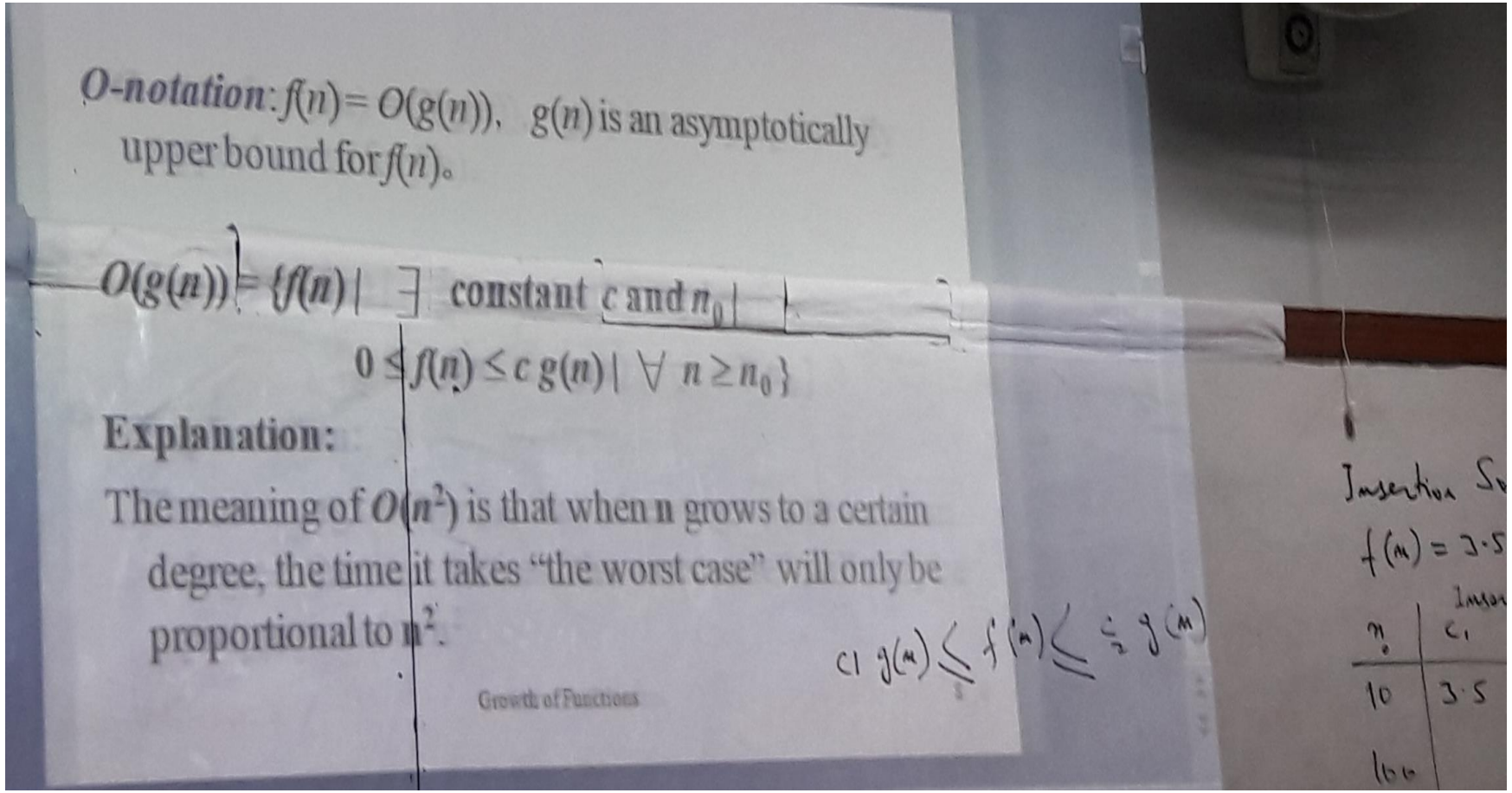
The meaning of $O(n^2)$ is that when **n** grows to a certain degree, the time it takes "the worst case" will only be proportional to n^2.

Growth of Functions 9

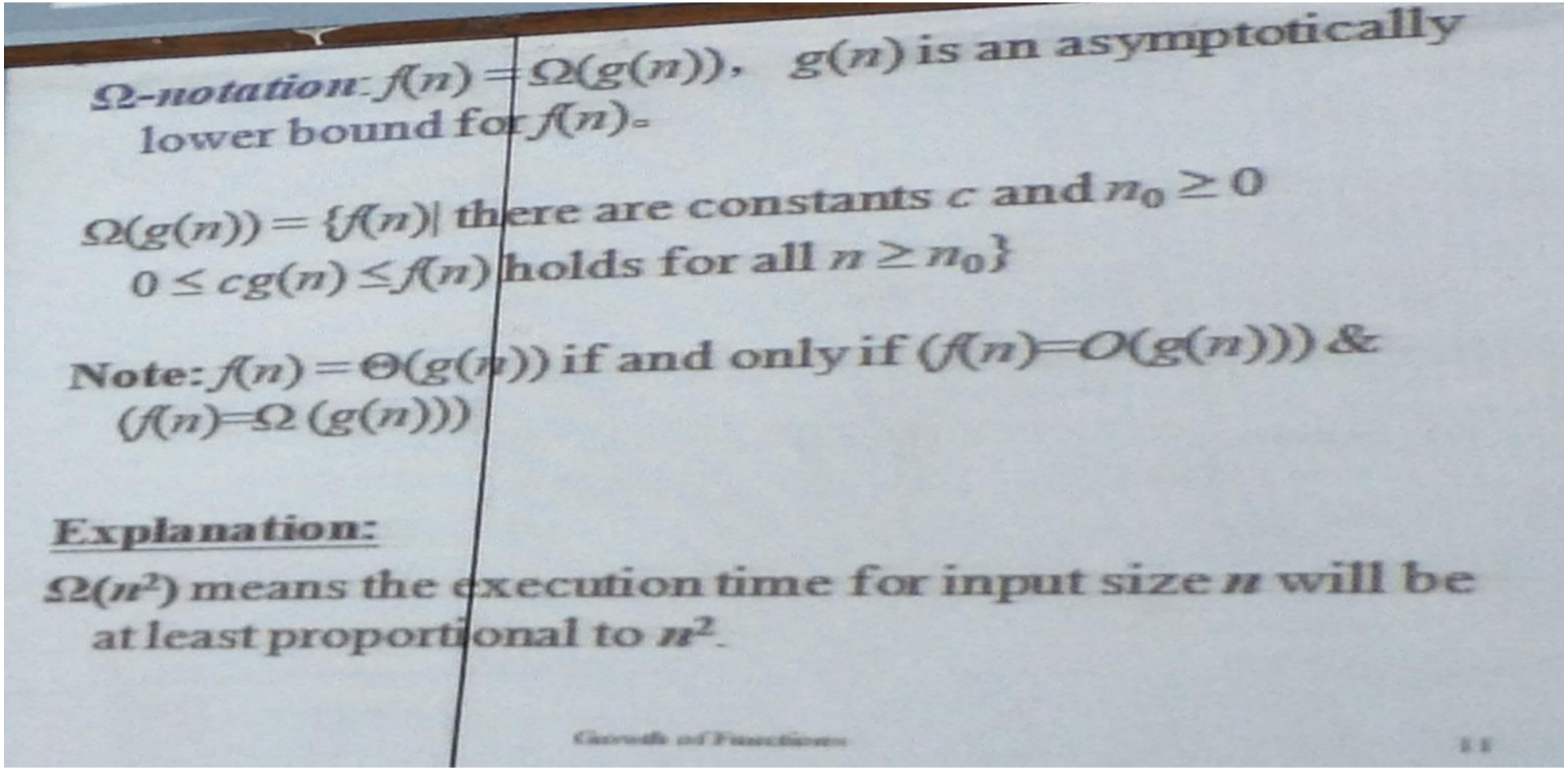
- **Big O notation**



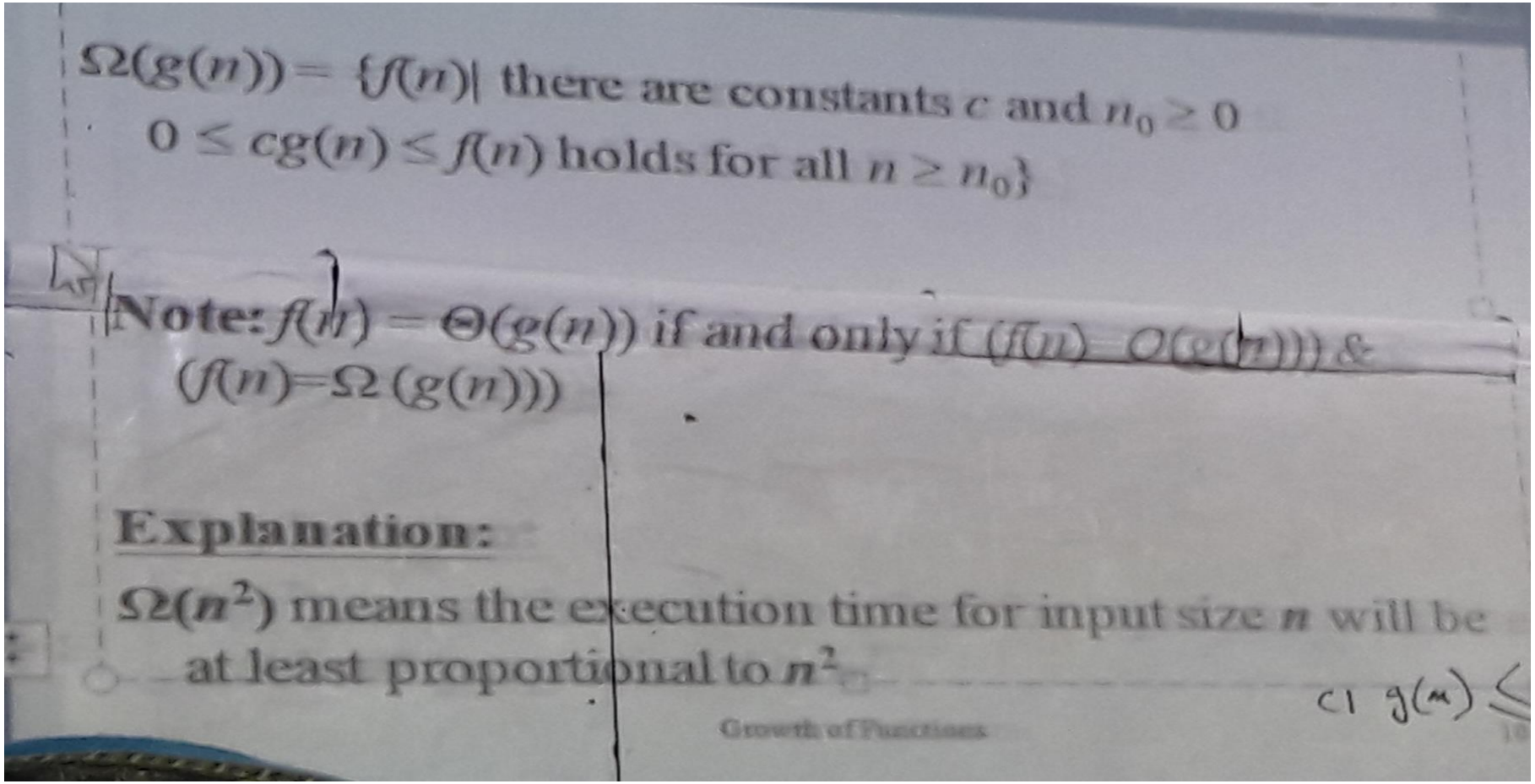
- **Big O notation (cont……)**

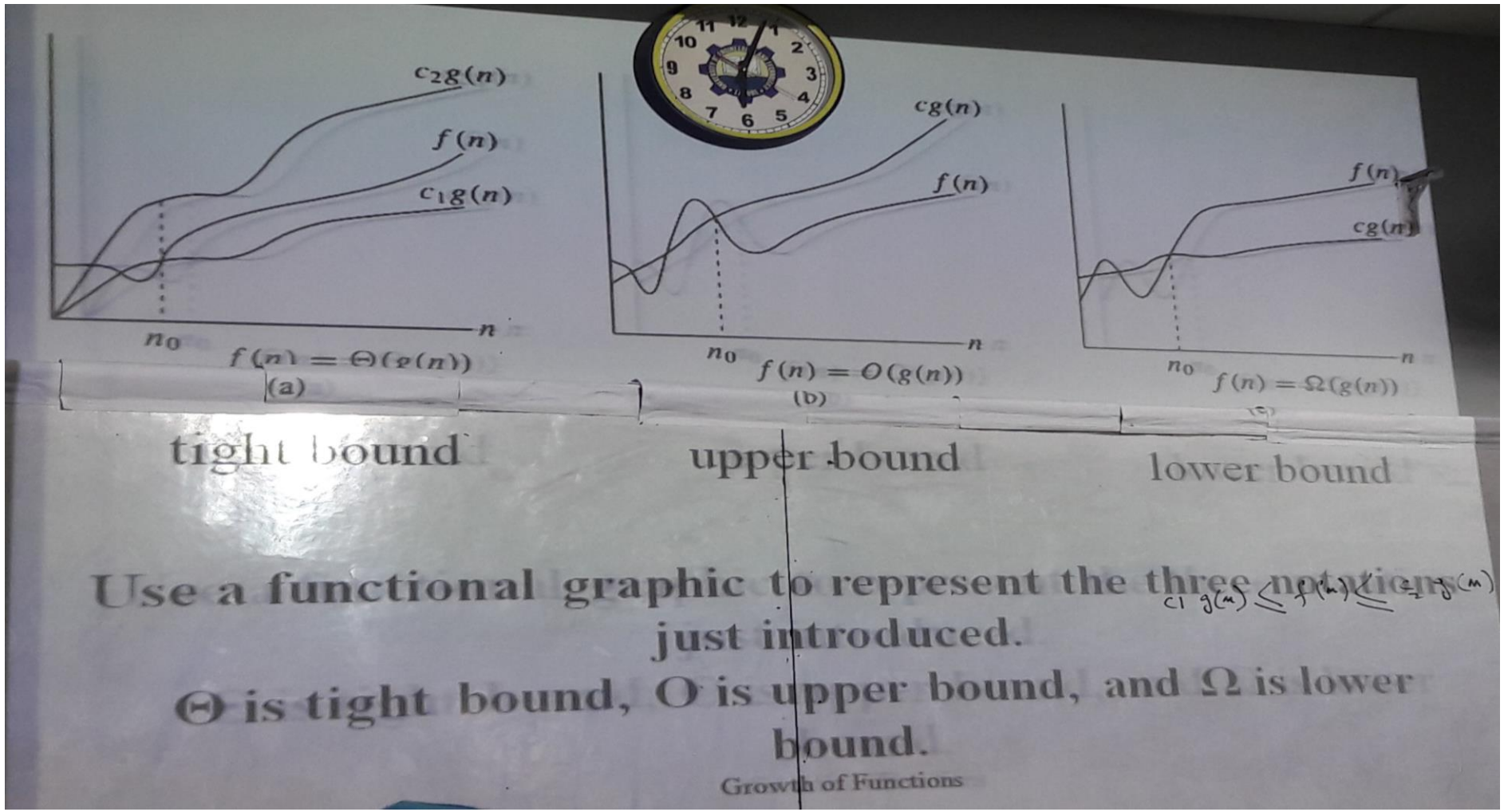


- **Big Omega notation (conditions)**

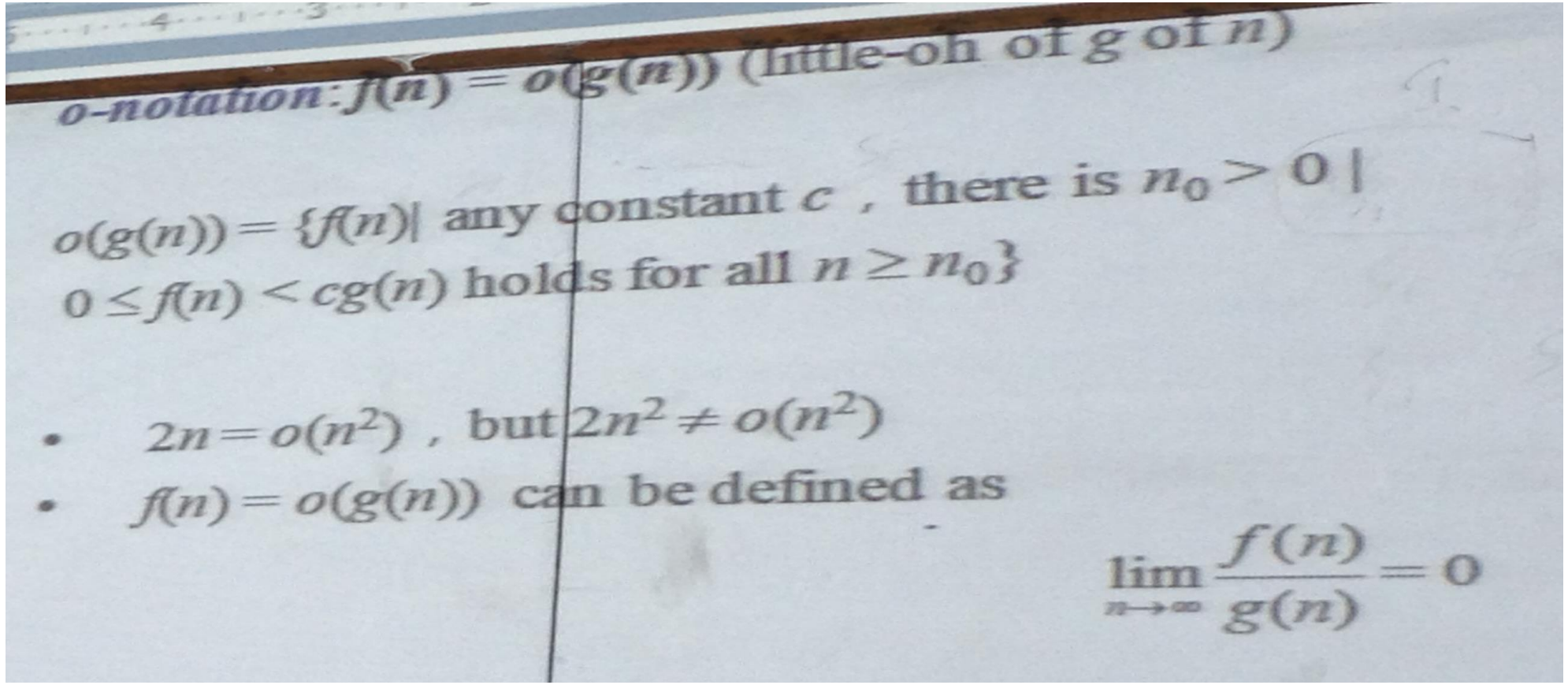


- **Big Omega (cont.…)**

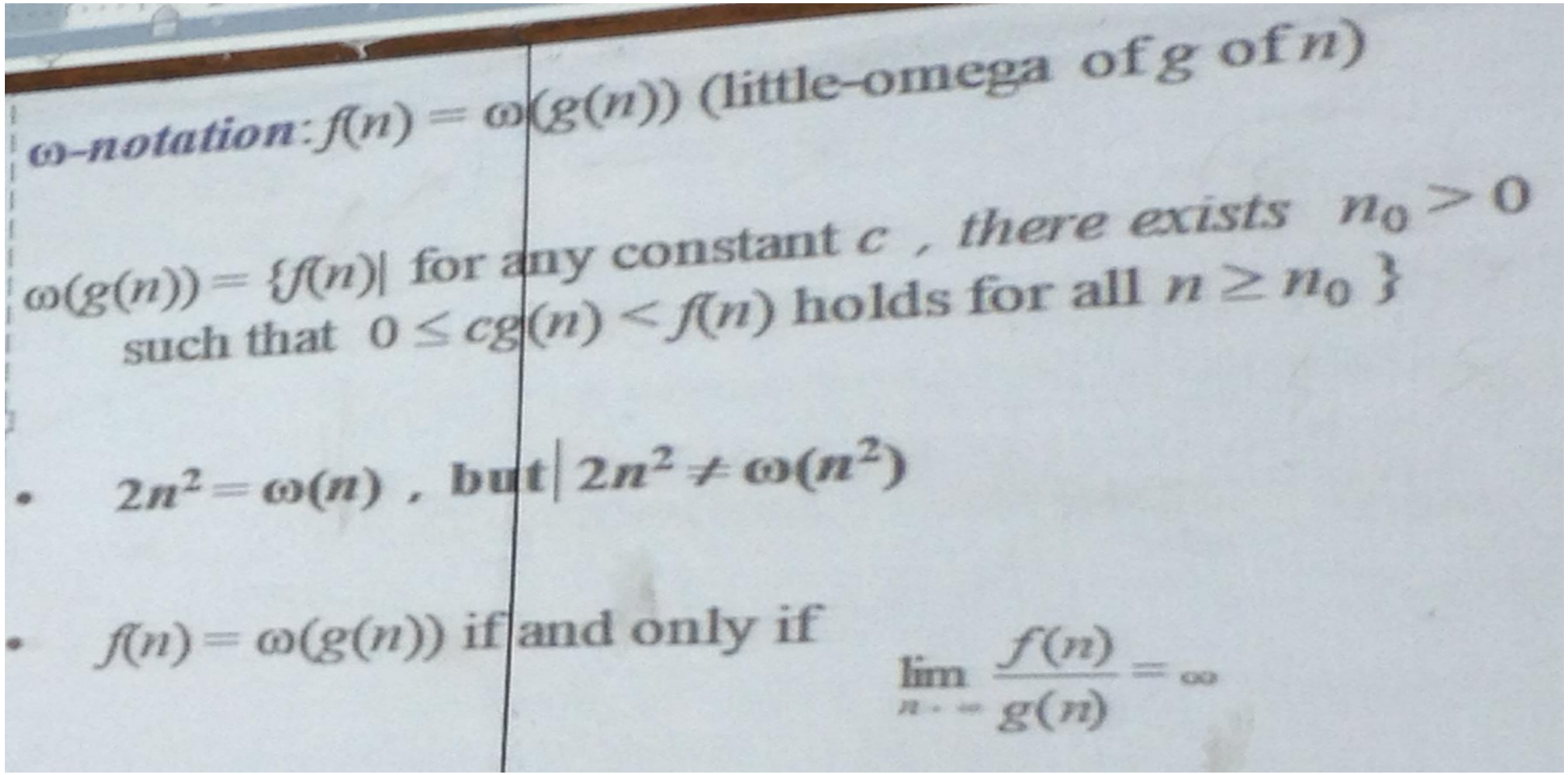



$c_2g(n)$
$f(n)$
$c_1g(n)$
n_0
n
$f(n) = \Theta(g(n))$
(a)
$cg(n)$
$f(n)$
n_0
n
$f(n) = O(g(n))$
(b)
$f(n)$
$cg(n)$
n_0
n
$f(n) = \Omega(g(n))$
(c)
tight bound
upper bound
lower bound
Use a functional graphic to represent the three notations just introduced.
Θ is tight bound, O is upper bound, and Ω is lower bound.
Growth of Functions

- **Little O notation**



- **Little Omega notation**



- **Comparison functions**