CSE 2103
(Data Structures)

Lecture on
Chapter-6: Stacks, Queues, Recursion

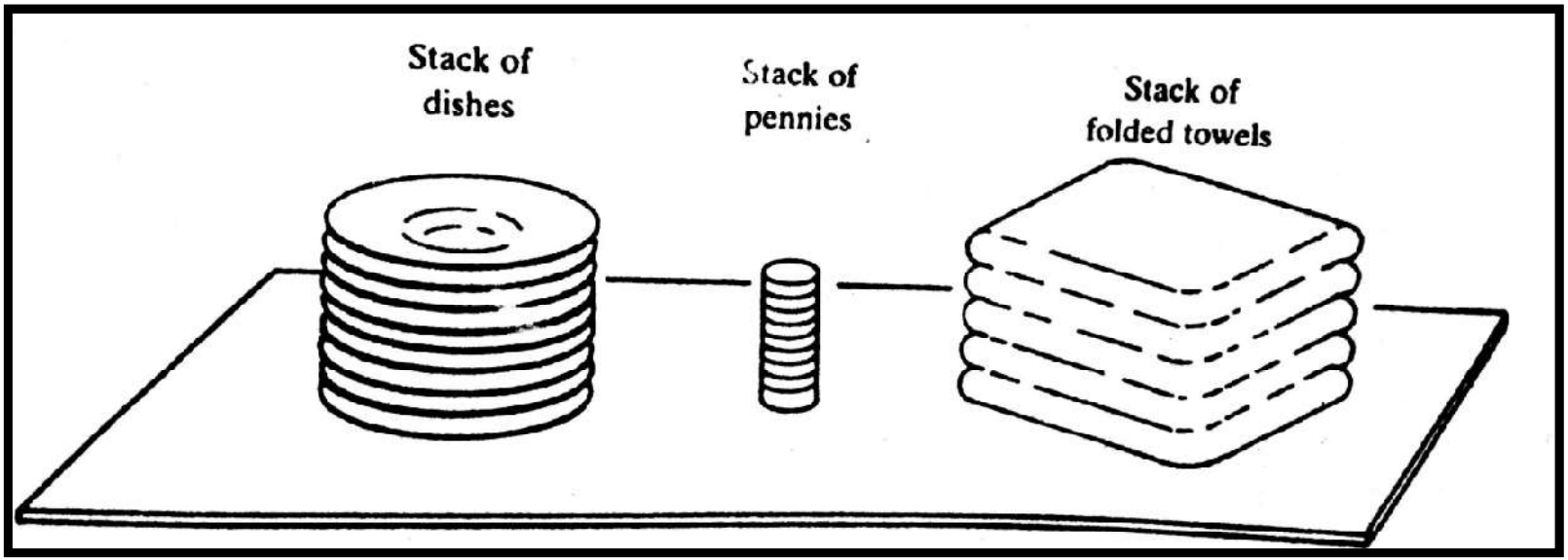
By

Dr. M. Golam Rashed

(golamrashed@ru.ac.bd)

Department of Computer Science and Engineering (CSE)
Varendra University, Rajshahi, Bangladesh

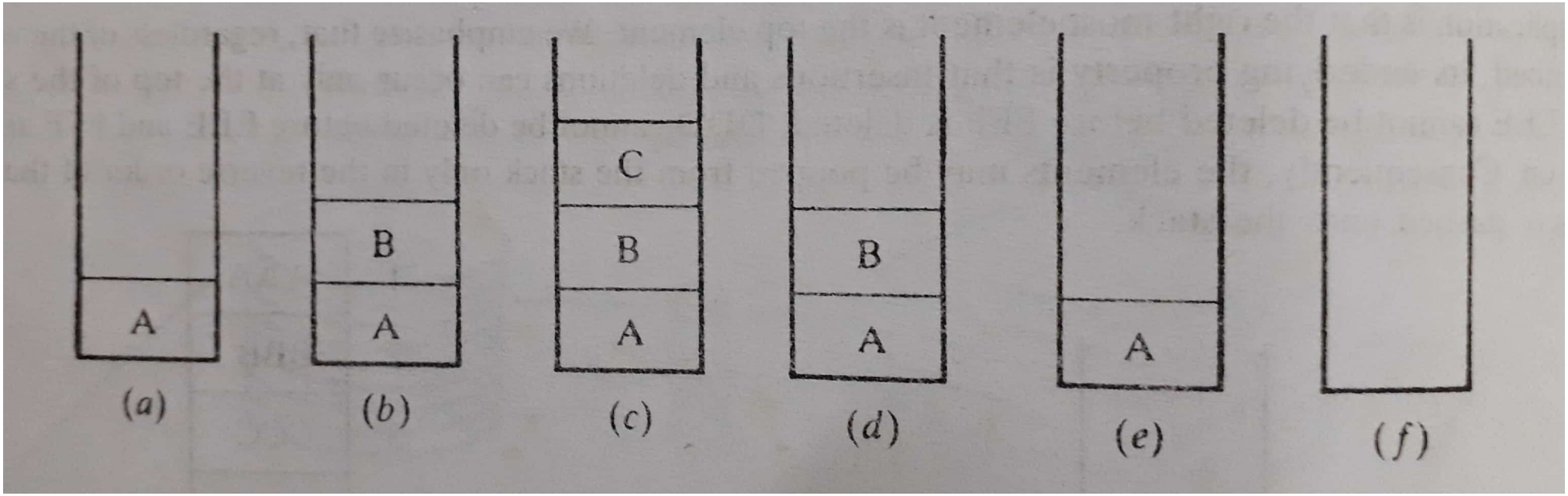
STACKS: Introduction



- A stack is a linear structure in which items may be added or removed only at one ends
- Stacks are also called Last-in-First-out (LIFO) list.
- Other names:
  - Piles
  - Push-down lists.
- Although the stack may be a very restricted type of data structure, it has many important applications in computer science.

STACKS: Dealing Postponed Decisions

Stacks are frequently used to indicate the order of the processing of data when certain steps of the processing must be postponed until other conditions are fulfilled.



STACKS: Special Terminology

Special terminology is used for two basic operations associated with stacks:

(a) “Push”-used to insert an element into a stack

(b) “Pop”-used to delete an element from a stack.

STACKS: Push Example

Suppose the following 6 elements are pushed, in order, onto an empty stack:

AAA, BBB, CCC, DDD, EEE, FFF

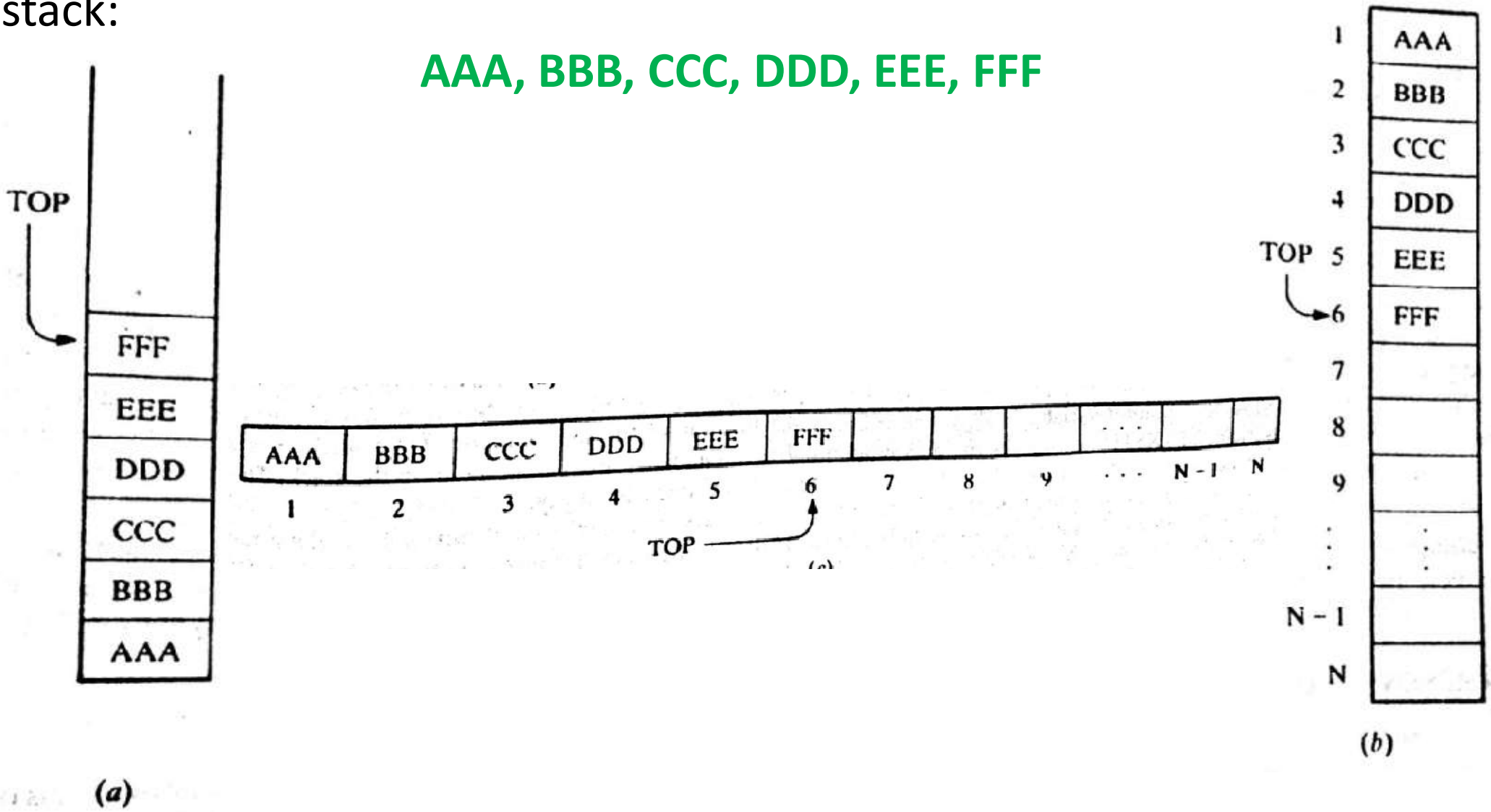


Diagram of Stack

STACKS: Array Representation

- Stack may be represented in the computer in various ways,
  - ✓ usually by means of a one-way list, or
  - ✓ A linear array
- Usually Stacks will be maintained by……
  - A linear array, **STACK**,**(**main lists**)**
  - A pointer variable, **TOP**, **(**contains the locations of the top element of the stack**)**
  - A variable **MAXSTK**, (gives the maximum number of elements that can be held by the stack)

The condition TOP=0 or **TOP=NULL** of a **STACK** indicates…..?

EMPTY

STACKS: Array Representation Example

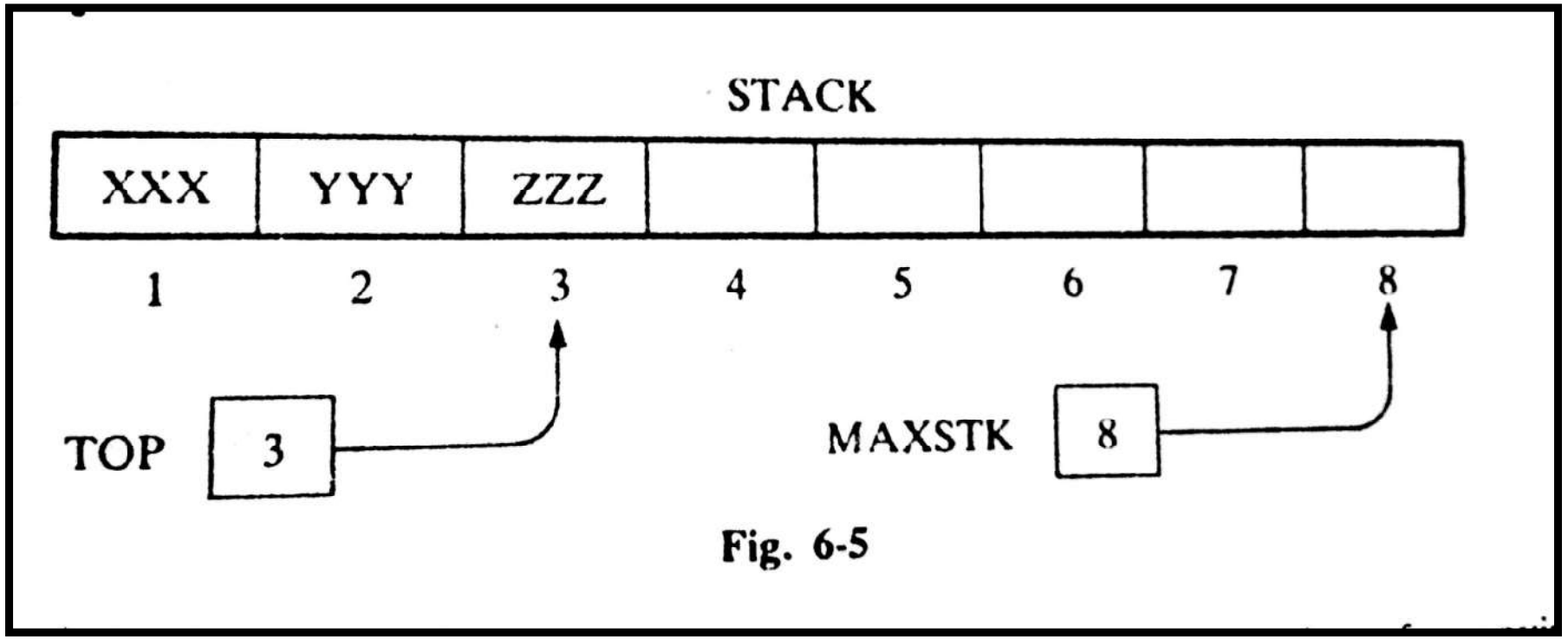


Fig. 6-5

Present Status: TOP=3;

MAXSTK=8;

There is room for 5 more items in the stack

Future Implementation:

Pushing an item onto a stack

Popping an item from a stack

STACKS: Pushing Algorithm

PUSH (STACK, TOP, MAXSTK, ITEM)

Step 1. [Stack already filled?]

If **TOP=MAXSTK,** then print: **OVERFLOW**, and Return.

Step 2. Set **TOP**=**TOP**+1. [Increase TOP by 1.]

Step 3. Set **STACK**[**TOP**]:=**ITEM.** [Insert ITEM in new TOP position]

Step 4. Return

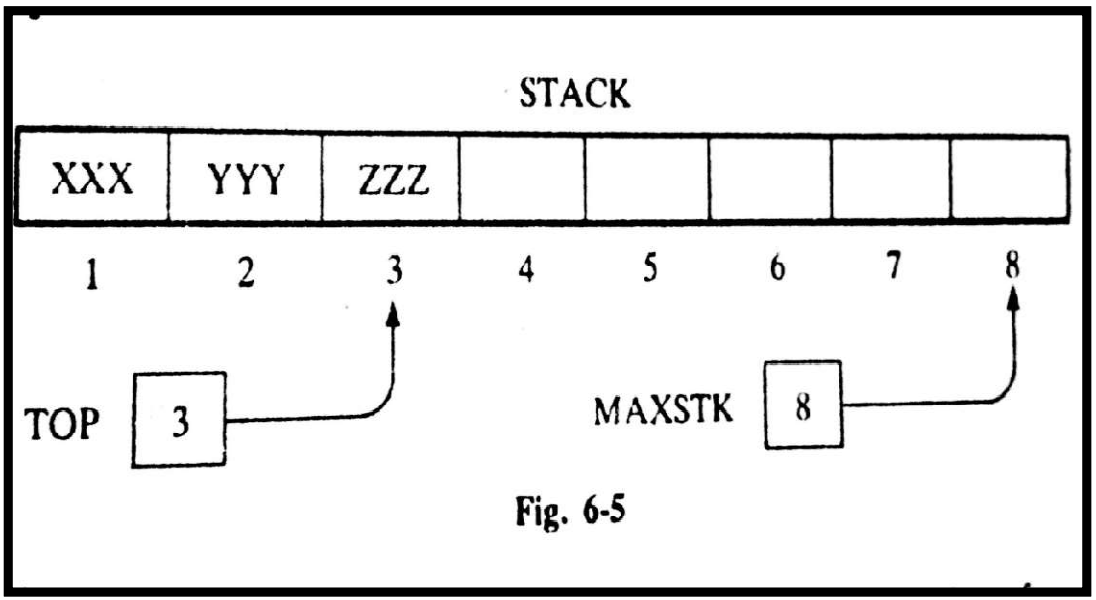


Fig. 6-5

1. Since TOP=3, go to Step 2

2. TOP=3+1=4

3. STACK [TOP]=STACK[4]= **www**

4. Return

STACKS: Popping Algorithm

POP (STACK, TOP, ITEM)

Step 1. [Stack has an item to be removed?]

If **TOP=0,** then print: **Underflow**, and Return.

Step 2. Set **ITEM**= **STACK**[**TOP**] [Assign TOP element to ITEM.]

Step 3. Set **TOP**:= **TOP-1.** [Decreases TOP by 1]

Step 4. Return

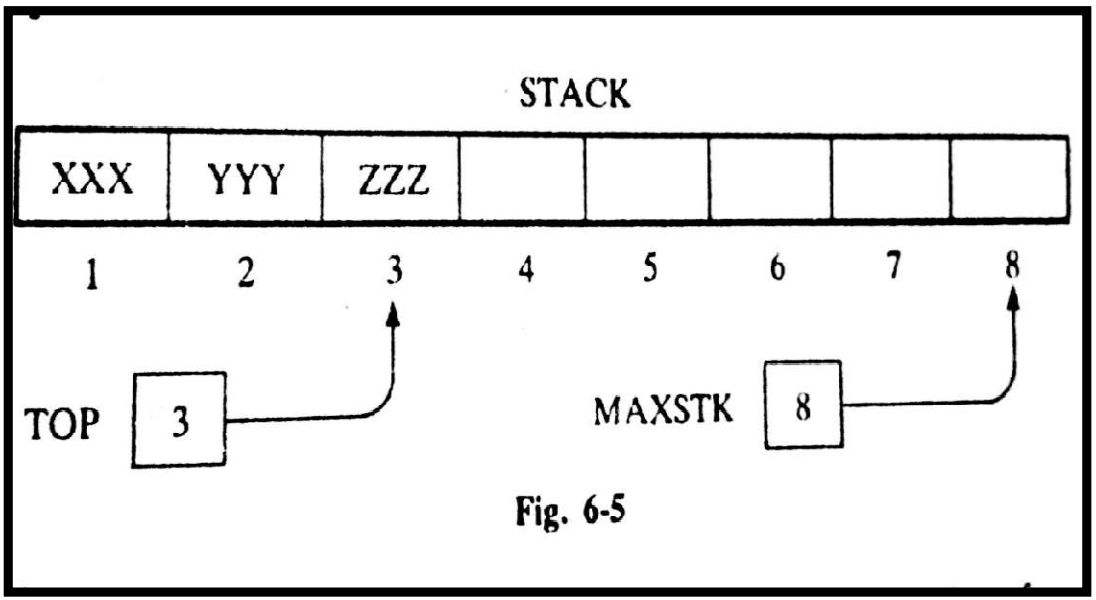


Fig. 6-5

1. Since TOP=3, go to Step 2

2. ITEM=ZZZ.

3. TOP=3-1=2.

4. Return.

STACKS:

PUSH

1. Since TOP=3, go to Step 2

2. TOP=3+1=4

3. STACK [TOP]=STACK[4]= **www**

4. Return

POP

1. Since TOP=3, go to Step 2

2. ITEM=ZZZ.

3. TOP=3-1=2.

4. Return.

- ✓ **TOP** is changed before the insertion in PUSH, whereas
- ✓ **TOP** is changed after the deletion in POP

STACKS: Arithmetic Expressions;

- ✓ Let Q be an arithmetic expression involving constants and operation.

Q: 2 ↑3+5*2 ↑2-12/6, we have to find the value of Q

- ✓ Q may have different levels of precedence in its binary operations.
- ✓ We assume the following three levels of precedence for the usual **FIVE** binary operations.
  - ➢ Highest: Exponential (↑)
  - ➢ Next Highest: Multiplication (*) and division(/)
  - ➢ Lowest: Addition (+) and Subtraction(-)
- ✓ Thus, we obtain after exponentiations (8+5*4-12/6)
- ✓ After, multiplication and division (8+20-2)
- ✓ After. Addition and subtraction , 26

STACKS: Polish Notation

For most common arithmetic operations:

A+B, C-D, E*F, G/H (*infix notation*)

Polish notation refers to the notation in which the operator symbol is placed before its two operands.

For example: +AB, -CD, *EF, /GH

Instant Exercise: (***Infix expression*** to ***polish notation***)

(A+B)*C = ?
= *[A+B]C=?
= *+ABC

A+(B*C) = ?
= A+[*BC]=?
= +A*BC

STACKS: Polish Notation

(A+B)/(C-D) = ?

= [+AB]/[-CD] = ?

= /+AB-CD

- ✓ The fundamental property of polish notation is that the order in which the operations are to be performed is completely determined by
  - ✓ The positions of the operators, and
  - ✓ Operands in the expression.
- ✓ **One never needs parentheses when writing expressions in polish notation.**

STACKS: Reverse Polish Notation

Reverse Polish Notation refers to the analogous notation in which the operator symbol is placed after its two operands:

AB+, CD-, EF*, GH/

- ✓ This notation is frequently called ***postfix notation***
- ✓ One never needs parentheses to determine the order of the operations in any arithmetic expression in ***postfix notation***

STACKS: Evaluation of Postfix Expression

P: 5, 6, 2, +, *, 12, 4, /, -

Symbol Scanned	STACK
(1) 5	5
(2) 6	5, 6
(3) 2	5, 6, 2
(4) +	5, 8
(5) *	40
(6) 12	40, 12
(7) 4	40, 12, 4
(8) /	40, 3
(9) -	37
(10))	

STACKS:

Algorithm to find the VALUE of an arithmetic expression P written in postfix notation.

1. Add a right parenthesis “(” at the end of P.
2. Scan P from left to right and repeat Steps 3 and 4 for each of P until the “)” is encountered.
3. If an operand is encountered, put it on STACK
4. If an operator ⌘ is encountered, then:
   a) Remove the two top elements of the STACK, where A is the top element and B is the next-to-top element
   b) Evaluate B ⌘ A
   c) Place the result of (b) back on STACK.

   [End of If structure]

   [End of Step 2 loop]
5. Set VALUE equal to the TOP element on STACK.

Exit.

STACKS: Transforming Q into P expression

Q: A + (B * C- (D / E ↑ F) * G) * H = (?) P expression

CSE 2103

Symbol Scanned	Stack	Expression P
(1) A	(	A
(2) +	(+	A
(3) (	(+ (	A
(4) B	(+ (	A B
(5) *	(+ (*	A B
(6) C	(+ (*	A B C
(7) -	(+ (-	A B C *
(8) (	(+ (- (	A B C *
(9) D	(+ (- (	A B C * D
(10) /	(+ (- (/	A B C * D
(11) E	(+ (- (/	A B C * D E
(12) ↑	(+ (- (/ ↑	A B C * D E
(13) F	(+ (- (/ ↑	A B C * D E F
(14))	(+ (-	A B C * D E F ↑ /
(15) *	(+ (- *	A B C * D E F ↑ /

STACKS: Transforming Q into P expression

Q: A + (B * C- (D / E ↑ F) * G) * H = (?) P expression

CSE 2103

Symbol Scanned	Stack	Expression P
(7) -	(+ (-	A B C *
(8) (	(+ (- (	A B C *
(9) D	(+ (- (	A B C * D
(10) /	(+ (- (/	A B C * D
(11) E	(+ (- (/	A B C * D E
(12) ↑	(+ (- (/ ↑	A B C * D E
(13) F	(+ (- (/ ↑	A B C * D E F
(14))	(+ (-	A B C * D E F ↑ /
(15) *	(+ (- *	A B C * D E F ↑ /
(16) G	(+ (- *	A B C * D E F ↑ / G
(17))	(+	A B C * D E F ↑ / G * -
(18) *	(+ *	A B C * D E F ↑ / G * -
(19) H	(+ *	A B C * D E F ↑ / G * - H
(20))		A B C * D E F ↑ / G * - H * +

Instant TEST

12 ,7 ,3 ,- ,/ ,2 ,1 ,5 ,+ ,* ,+

Which Expression?

Postfix or Prefix

✓ **Postfix**

Equivalent infix expression ?

P = 12, [7-3], / , 2 ,1, 5, +, *, +

= [12/(7-3)], 2, 1, 5, +, *, +

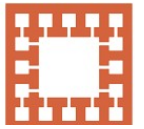
= [12/(7-3)], 2, [1+5], *, +

= [12/(7-3)], [2* (1+5)], +

= 12/(7-3) + 2* (1+5)

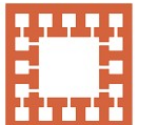
Result: 15

Instant TEST



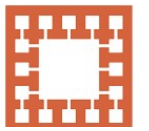
Infix	Prefix	Postfix
A+B * C+D	++A*BCD	ABC*+D+
(A+B) * (C+D)	*+AB+CD	AB+CD+*
A*B + C*D	+*AB*CD	AB*CD*+
A+B+C+D	++AB+CD	AB+CD++

Instant TEST



Infix Expression	Prefix Expression	Postfix Expression
1-4^3+7*(9^1/5)-2	-1+^43-*7/^9152	143^-791^5/*+2-
A+B-(C*D^E)*X+Y	+A-B+**C^DEXY	AB+CDE^*X*-Y+

QUICKSORT: An Application of STACKS



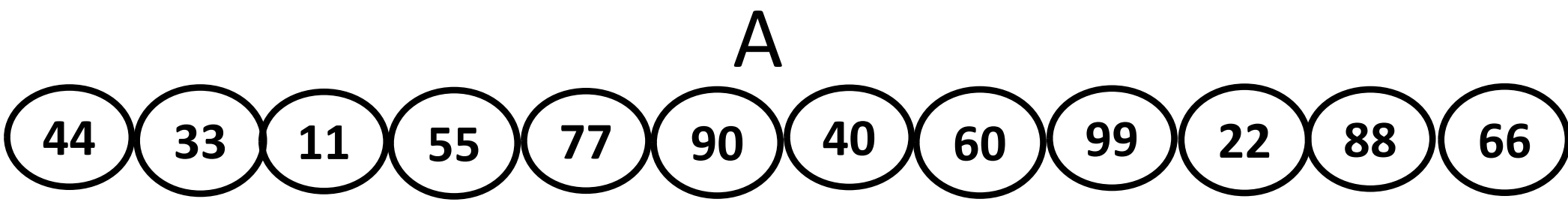
- What is Sorting?
  - ✓ The operation of rearranging the elements of a list so that they are in some logical order.
    - Numerically ordered (When list contain numerical data)
    - Alphabetically ordered (When list contains character data)

- Quicksort is an algorithm of the **DIVIDE-AND-CONQUER** type.
  - ➢ The problem of sorting a set is reduced to the problem of sorting two smaller sets.
- We illustrate this "Reduced Step" by means of a specific example
- Suppose A is the following list of 12 numbers.

A

44	33	11	55	77	90	40	60	99	22	88	66

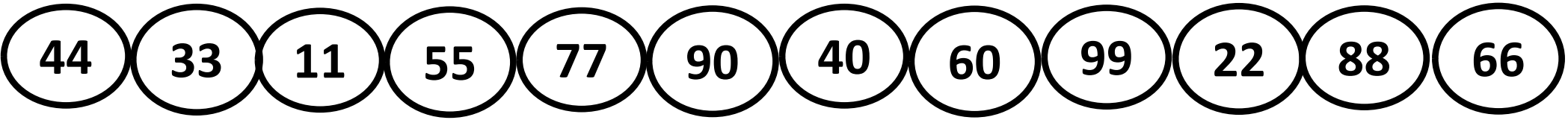
Reduction Step of the Quicksort Algorithm

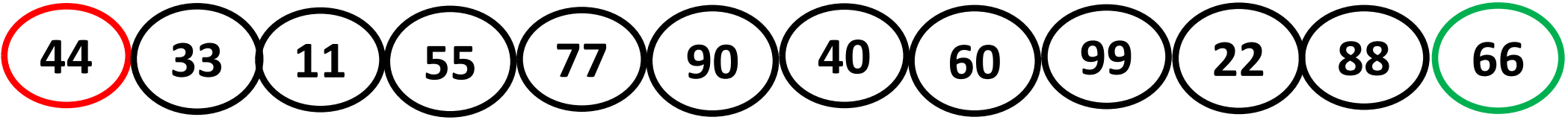


- The reduction step of the quicksort algorithm finds the final position of one of the numbers.
- In this illustration, we use the first number 44.

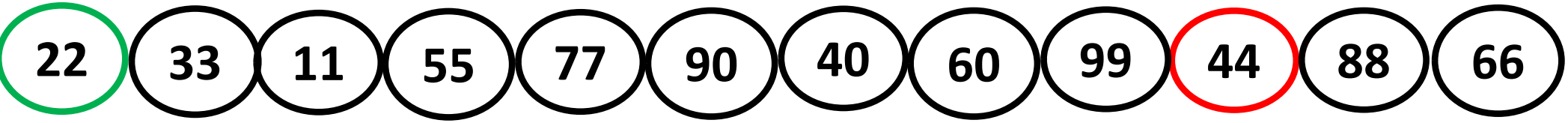
STACKS: Quicksort/ Reduction Step

CSE 2103

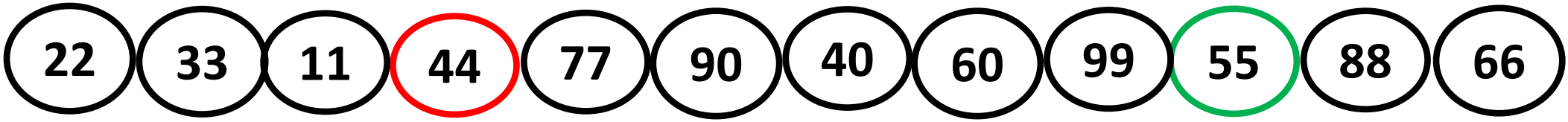




- ✓ Beginning with the last number, 66, scan the list from right to left till less than 44.
- ✓ Interchange 44 and 22



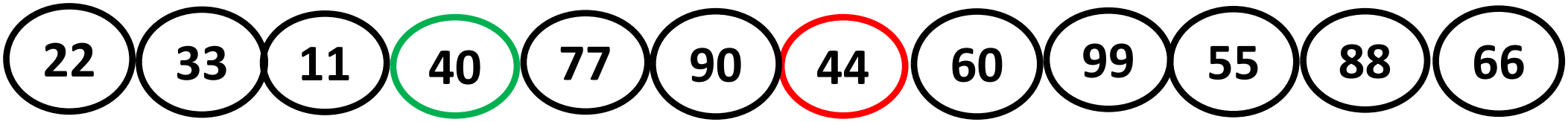
- ✓ Beginning with, 22 , scan left to right till greater than 44
- ✓ Interchange 44 and 55



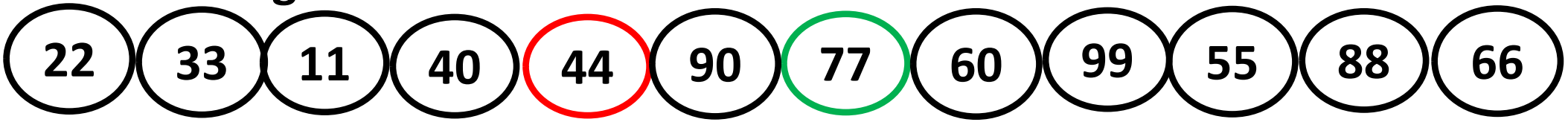
- ✓ Beginning with, 55, scan the list from right to left till less than 44.
- ✓ Interchange 44 and 40

STACKS: Quicksort/ Reduction Step

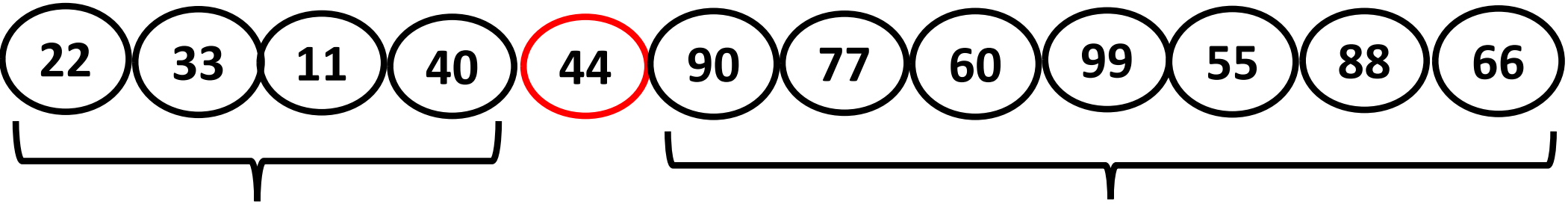
CSE 2103



- ✓ Beginning with, 40, scan left to right till greater than 44
- ✓ Interchange 44 and 77



- ✓ Beginning with, 77, scan the list from right to left till less than 44.
- ✓ Do not meet such a number before meeting 44.



First sublist **Second sublist**

- ✓ Thus, 44 is correctly placed in its final position.
- ✓ The task of sorting the original list A has now been reduced to the task of sorting each of the above sublists.

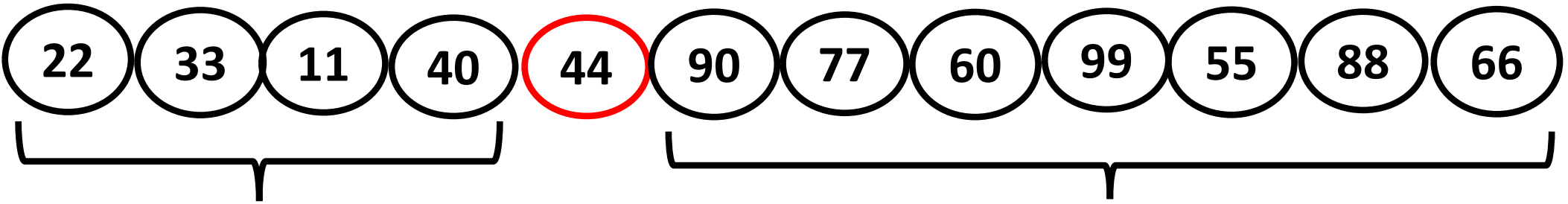
STACKS: Quicksort/ Reduction Step

- The reduction step is repeated with each sublist containing 2 or more elements.
- Since we can process one sublist at a time, we must be able to keep track of some sublist for future processing.
- **This is accomplished by using two STACKS**

LOWER and UPPER

to temporarily hold such sub-lists.

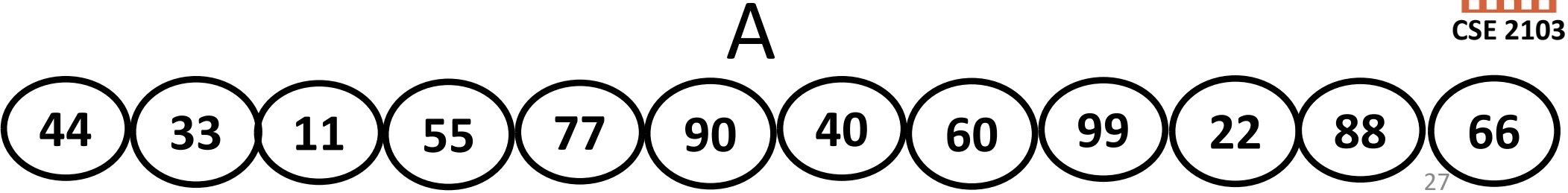
- The addresses of the first and last elements of each sublist, called its *"**BOUNDARY VALUES**"* , are pushed onto the STACKs **LOWER** and **UPPER**, respectively.
- The reduction steps is applied to a sublist only after its ***BOUNDARY VALUES*** are removed from the STACKs.



First sublist

Second sublist

STACKS: Illustration of the way the STACKS LOWER and UPPER are used



N=12 elements,

Thus,

Boundary values are ()?

1 and 12.

Now,

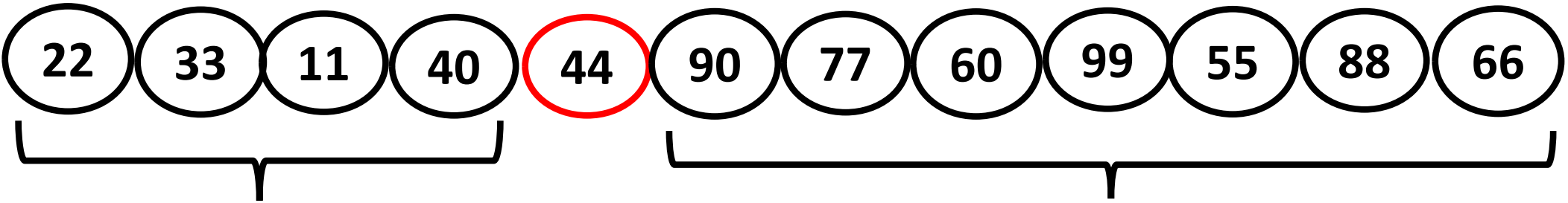
1 and 12 should be Stacked

LOWER:1 and UPPER:12

- **In order to apply the REDUCTION STEP, the algorithm first removes the top values 1 and 12.**
- **After removing the top values 1 and 12 from the STACK, leaving LOWER:(Empty) and UPPER: (Empty)**
- **Then Applies the REDUCTION STEP to the corresponding list A[1], A[2],….,A[12].**

STACKS: Illustration of the way the STACKS LOWER and UPPER are used

- After executing **REDUCTION STEP** to the list A[1] to A[12]
  - Finally places the first element 44, in A[5].



First sublist **Second sublist**

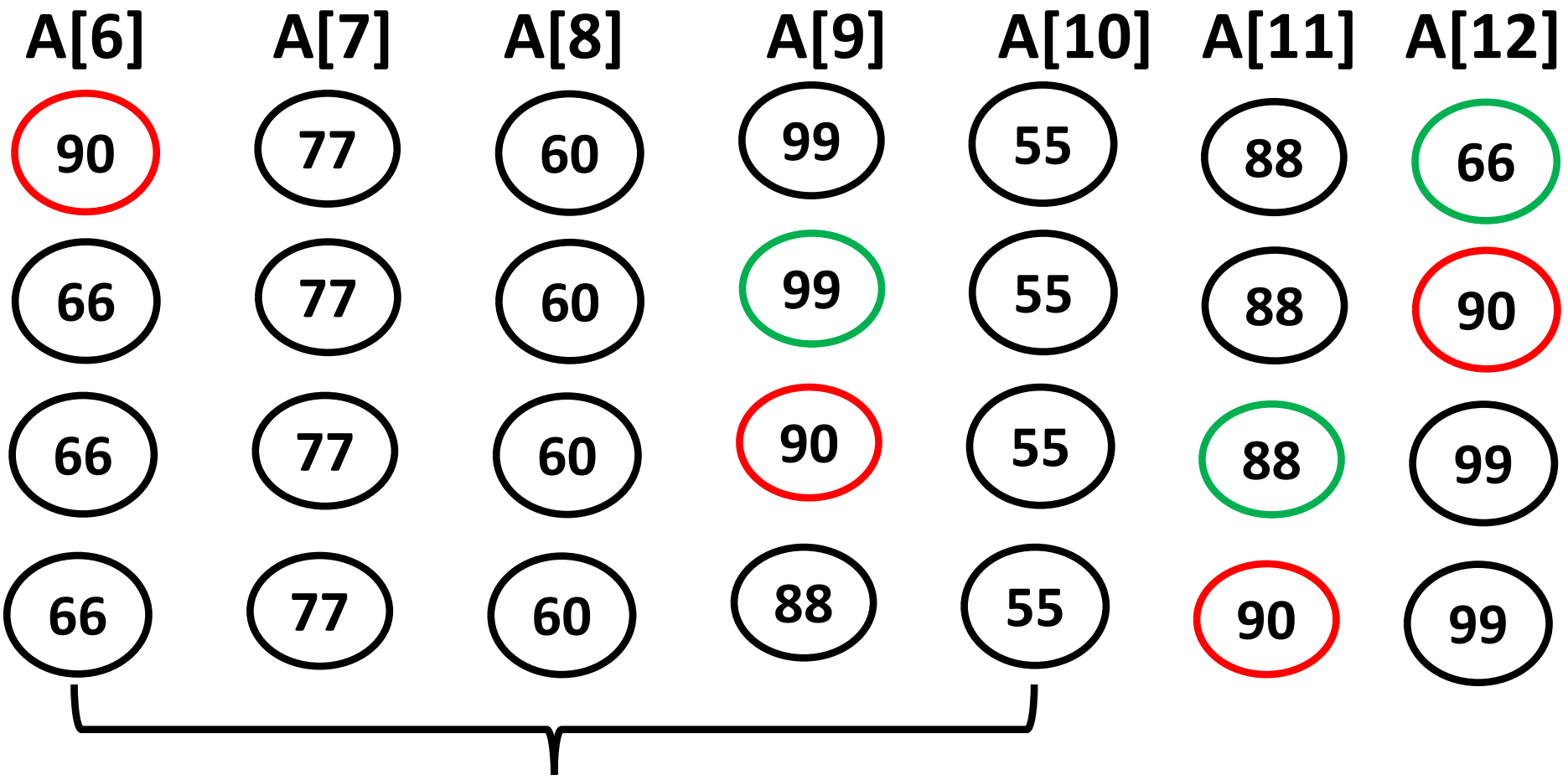
- Accordingly, the algorithm pushes the boundary values
  - 1 and 4 of the first sublist, and
  - 6 and 12 of the second sublist on to the STACK to yield

LOWER= **1, 6** and UPPER= **4, 12**

- In order to apply the **REDUCTION STEP** again, the algorithm removes the TOP values 6 and 12 from the STACKs, leaving

LOWER= **1,** and UPPER= **4**

STACKS: Illustration of the way the STACKS LOWER and UPPER are used



- The second sublist has only one element, Accordingly
- The algorithm pushes only the boundary values 6 and 10 of the first sublist on the STACKs to yield
  - LOWER= **1, 6** and UPPER= **4, 10**

QUICKSORT

- The quick sort is regarded as the best sorting algorithm.
- This is because of its significant advantage in terms of efficiency because it is able to deal well with a huge list of items.
- Because it sorts in place, no additional storage is required as well.
- The slight disadvantage of quick sort is that its worst-case performance is similar to average performances of the bubble, insertion or selections sorts.
- In general, the quick sort produces the most effective and widely used method of sorting a list of any item size.