CSE-213
(Data Structure)

Lecture on

Sorting

Md. Jalal Uddin

Lecturer

Department of CSE

City University

Email: jalalruice@gmail.com

No: 01717011128 (Emergency Call)

Sorting

Sorting in Linear Array:

Sorting an array is the ordering the array elements in *ascending (increasing* from min to max) or ***descending (decreasing – from max to min) order.***

Example:

{2 1 5 7 4 3} => {1, 2, 3, 4, 5,7} ***ascending order***

{2 1 5 7 4 3} => {7,5, 4, 3, 2, 1} ***descending order***

Let A be a list of n numbers. Sorting A refers to the operation of rearranging the elements of A so they are in increasing order.

i.e. so that A[a]<A[2]<A[3]<…….<A[N]

For example,

Suppose A originally is the list

8, 4, 19, 2, 7, 13, 5, 16

After sorting, A is the list

2, 4, 5, 7, 8, 13, 16, 19

Sorting: BUBBLE SORT

Suppose the following numbers are stored in an array A:

32, 51, 27, 85, 66, 23, 13, 57

We apply the bubble sort to the array A. We discuss each pass separately.

Pass 1. We have the following comparisons:

(*a*) Compare A_1 and A_2. Since $32 < 51$, the list is not altered.

(*b*) Compare A_2 and A_3. Since $51 > 27$, interchange 51 and 27 as follows:

32, (27), (51), 85, 66, 23, 13, 57

(*c*) Compare A_3 and A_4. Since $51 < 85$, the list is not altered.

(*d*) Compare A_4 and A_5. Since $85 > 66$, interchange 85 and 66 as follows:

32, 27, 51, (66), (85), 23, 13, 57

(*e*) Compare A_5 and A_6. Since $85 > 23$, interchange 85 and 23 as follows:

32, 27, 51, 66, (23), (85), 13, 57

(*f*) Compare A_6 and A_7. Since $85 > 13$, interchange 85 and 13 to yield:

32, 27, 51, 66, 23, (13), (85), 57

(*g*) Compare A_7 and A_8. Since $85 > 57$, interchange 85 and 57 to yield:

32, 27, 51, 66, 23, 13, (57), (85)

At the end of the first pass, the largest number ,85 has moved to the last position.

Rest of the number are not sorted.

Sorting: BUBBLE SORT

Pass 2. (27, 33,) 51, 66, 23, 13, 57, 85
27, 33, 51, (23, 66,) 13, 57, 85
27, 33, 51, 23, (13, 66,) 57, 85
27, 33, 51, 23, 13, (57, 66,) 85

At the end of Pass 2, the second largest number, 66, has moved its way down to the next-to-last positi

Pass 3. 27, 33, (23, 51,) 13, 57, 66, 85
27, 33, 23, (13, 51,) 57, 66, 85

Pass 4. 27, (23, 33,) 13, 51, 57, 66, 85
27, 23, (13, 33,) 51, 57, 66, 85

Pass 5. (23, 27,) 13, 33, 51, 57, 66, 85
23, (13, 27,) 33, 51, 57, 66, 85

Pass 6. (13, 23,) 27, 33, 51, 57, 66, 85

Pass 6 actually has two comparisons, A_1 with A_2 and A_2 and A_3. The second comparison does involve an interchange.

Pass 7. Finally, A_1 is compared with A_2. Since $13 < 23$, no interchange takes place.

...after the seventh pass. (Observe that in this example, the list was act

Since the list has 8 elements, it is sorted after the seventh pass.

Bubble Sort: Algorithm

BUBBLE (DATA, N)

(Here DATA is an array with N elements. This algorithm sorts the elements in DATA)

Step 1. Repeat Steps 2 and 3 for K=1 to N-1.

Step 2. Set PTR:=1 [Initialize pass pointer PTR]

Step 3. Repeat while PTR<=N-K [Execute pass.]

(a) If DATA[PTR]>DATA[PTR+1], then:

Interchange Data[PTR] and DATA[PTR+1].

[End of IF Structure]

(b) Set PTR:=PTR+1.

[End of inner loop.]

[End of Step 1. outer loop.]

Step 4. Exit.

Complexity of BUBBLE SORT

Traditionally, the time for this sorting algorithm is measured in terms of the number of comparisons. The number f(n) of comparisons in the bubble sort is easily computed.

Specifically, there are n-1 comparisons during the first pass, which placed the largest element to the last position; there are n-2 comparisons in the second step, which placed the second largest element in the next-to-the last position, and so on. Thus.

$$F(n)=(n-1)+(n-2)+....+2+1$$
$$=n(n-1)/2$$
$$=n^2/2+O(n)$$
$$=O(n^2)$$

Selection Sort

Selection sort is a simple sorting algorithm.

This sorting algorithm is an in-place comparison-based algorithm

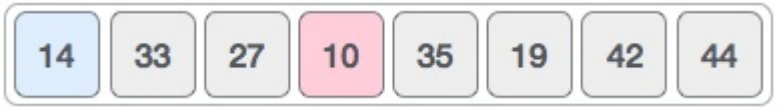
- List is divided into two parts, the sorted part at the left end and the unsorted part at the right end.
- Initially, the sorted part is empty and the unsorted part is the entire list.

The smallest element is selected from the unsorted array and swapped with the leftmost element

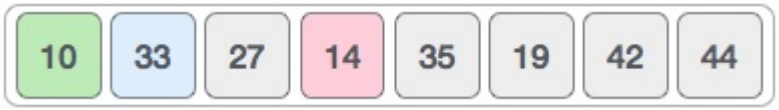
This algorithm is not suitable for large data sets as its average and worst case complexities are of $O(n^2)$, where **n** is the number of items.

Selection Sort: How it works?

- For the first pass: the lowest value 10 is selected and swapped with 14



- For the second pass: we start from 33 and scan the list for the second lowest and swap



Selection Sort: How it works?

Selection Sort: Procedure

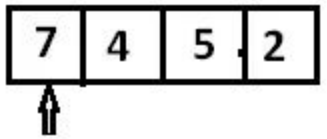
```
procedure selection sort
  list  : array of items ;   n : size of list

  for i = 1 to n - 1
   iMIN = i                        /* set current element as minimum*/
    for j = i+1 to n              /* check the element to be minimum */
      if list[j] < list[iMIN] then
        iMIN = j;
      end if
    end for

    /* swap the minimum element with the current element*/
    if iMIN != i  then
      swap list[iMIN] and list[i]
    end if
```

Insertion Sort: Procedure

To sort an array of size N in ascending order iterate over the array and compare the current element (key) to its predecessor, if the key element is smaller than its predecessor, compare it to the elements before. Move the greater elements one position up to make space for the swapped element.

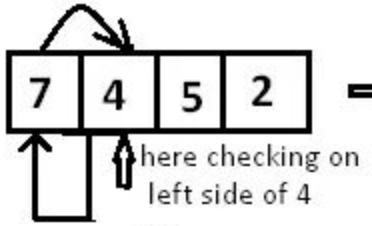
Insertion Sort: Procedure

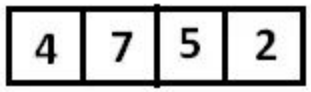
STEP 1.



No element on left side of 7,so no change in its position .

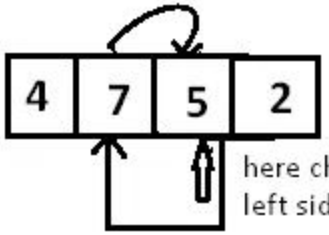
STEP 2.

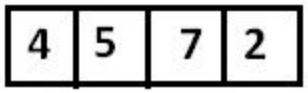




As 7>4 ,therfore 7 will be moved forward and 4 will be moved to 7's position.

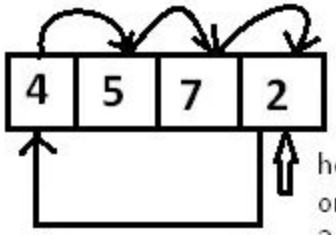
STEP 3.





As 7>5,7 will be moved forward,but 4 <5,so no change in position of 4. And 5 will be moved to position of 7.

STEP 4.



As all the element on left side of 2 are greater than 2,so all the elements will be moved forward and 2 will be shifted to position of 4

Insertion Sort: Procedure

Insertion Sort: Procedure

Algorithm

The simple steps of achieving the insertion sort are listed as follows -

Step 1 - If the element is the first element, assume that it is already sorted. Return 1.

Step 2 - Pick the next element, and store it separately in a **key.**

Step 3 - Now, compare the **key** with all elements in the sorted array.

Step 4 - If the element in the sorted array is smaller than the current element, then move to the next element. Else, shift greater elements in the array towards the right.

Step 5 - Insert the value.

Step 6 - Repeat until the array is sorted.

Insertion Sort: Procedure

Pseudocode:

```
Algorithm: Insertion-Sort(A)
for j = 2 to A.length
   key = A[j]
   i = j – 1
   while i > 0 and A[i] > key
      A[i + 1] = A[i]
      i = i -1
   A[i + 1] = key
```

Radix sort

Example, For the given unsorted list of elements, 236, 143, 26, 42, 1, 99, 765, 482, 3, 56, we need to perform the radix sort and obtain the sorted output list −

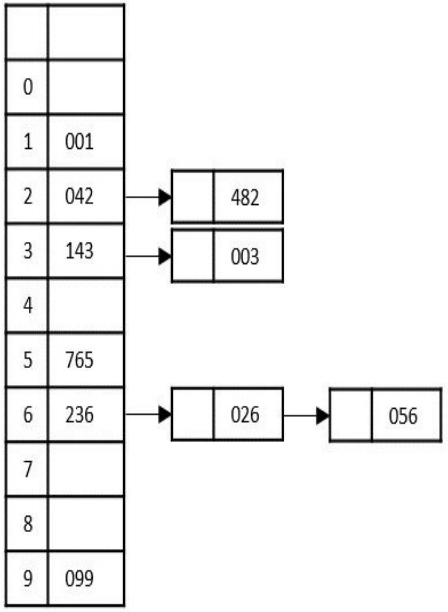
Step 1: Check for elements with maximum number of digits, which is 3. So we add leading zeroes to the numbers that do not have 3 digits. The list we achieved would be −

236, 143, 026, 042, 001, 099, 765, 482, 003, 056

Step 2: Construct a table to store the values based on their indexing. Since the inputs given are decimal numbers, the indexing is done based on the possible values of these digits, i.e., 0-9.

Radix sort

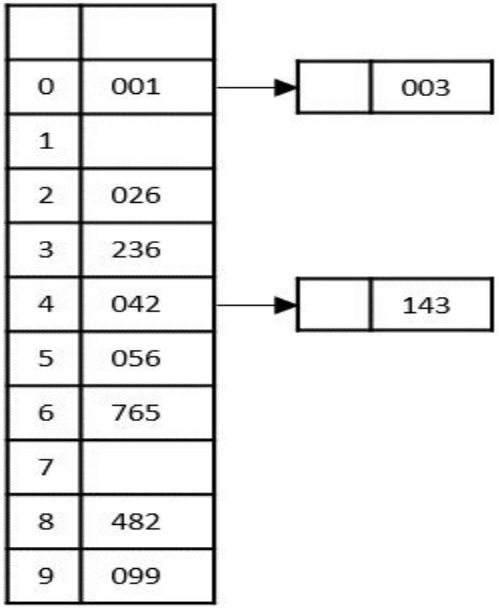
Step 3: Based on the least significant digit of all the numbers, place the numbers on their respective indices.



The elements sorted after this step would be 001, 042, 482, 143, 003, 765, 236, 026, 056, 099

Radix sort

Step 4: The order of input for this step would be the order of the output in the previous step. Now, we perform sorting using the second least significant digit.

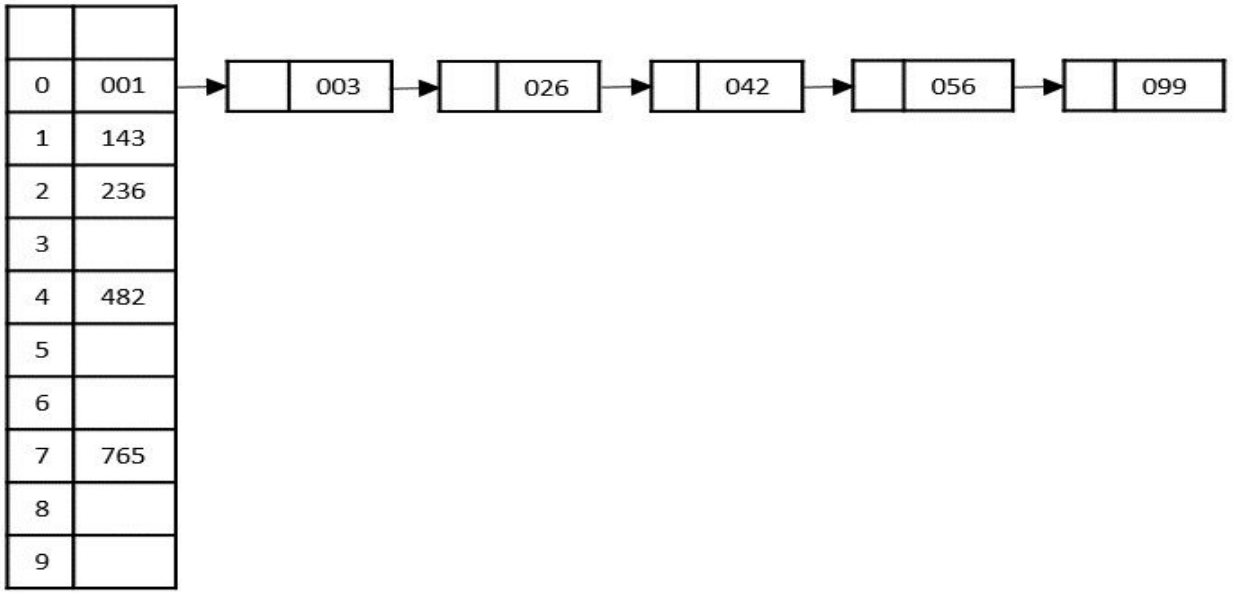


The order of the output achieved is 001, 003, 026, 236, 042, 143, 056, 765, 482, 099.

Radix sort

Step 5:The input list after the previous step is rearranged as − 001, 003, 026, 236, 042, 143, 056, 765, 482, 099

Now, we need to sort the last digits of the input elements.



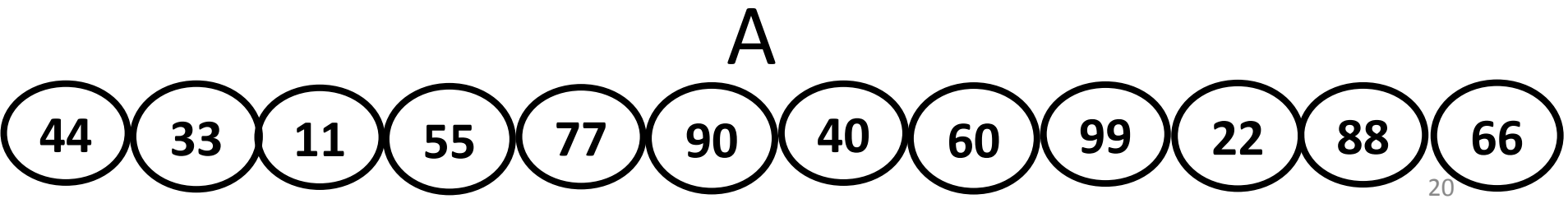
- Since there are no further digits in the input elements, the output achieved in this step is considered as the final output.
- The final sorted output is − 1, 3, 26, 42, 56, 99, 143, 236, 482, 765

QUICKSORT: An Application of STACKS

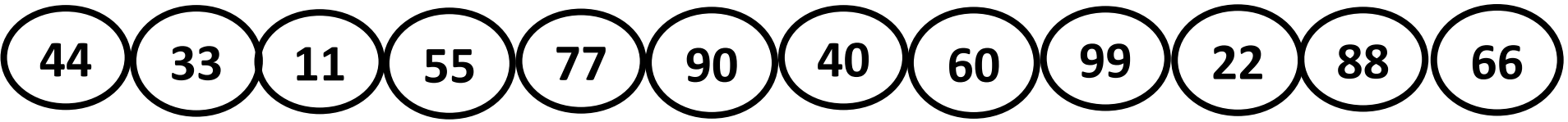
Quicksort is an algorithm of the divide-and-conquer type.

- The problem of sorting a set is reduced to the problem of sorting two smaller sets.

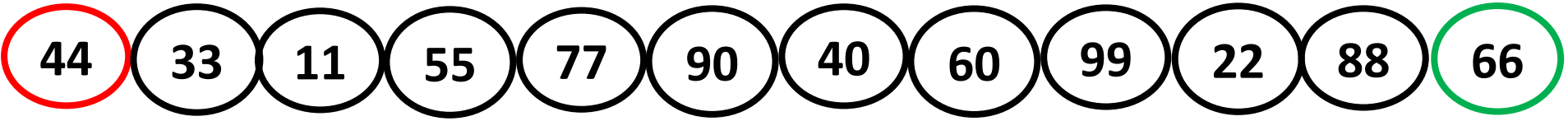
Quicksort is used to illustrate the application of Stacks



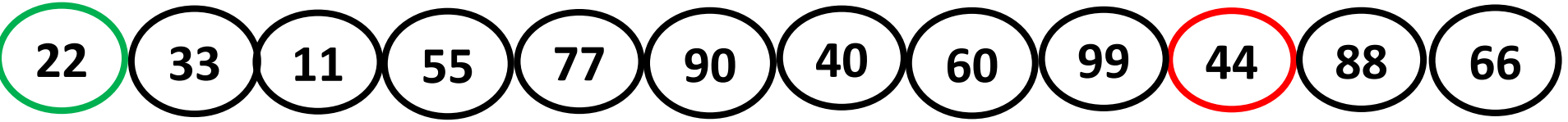
Quicksort/ Reduction Step



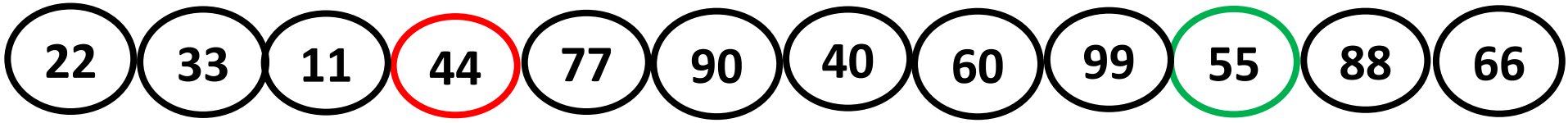
- The **REDUCTION STEP** of the quick sort algorithm finds the final position of **ONE** of the numbers



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

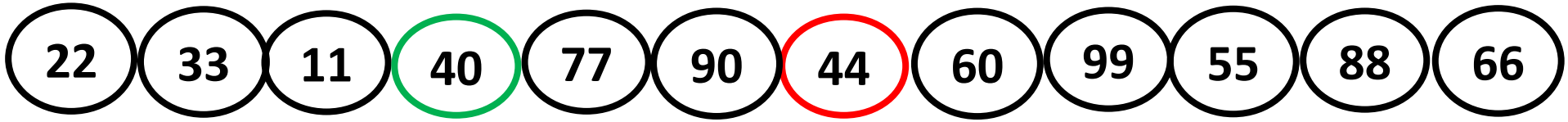


- Beginning with, 22 , scan left to right till greater than 44
- Interchange 44 and 55

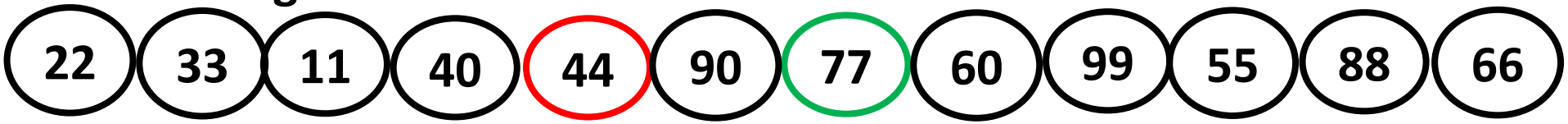


- Beginning with, 55, scan the list from right to left till less than 44.
- Interchange 44 and 40

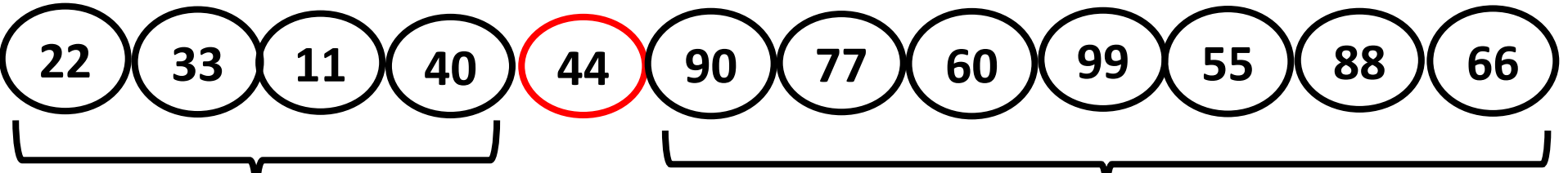
Quicksort/ Reduction Step



- 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.



- 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.

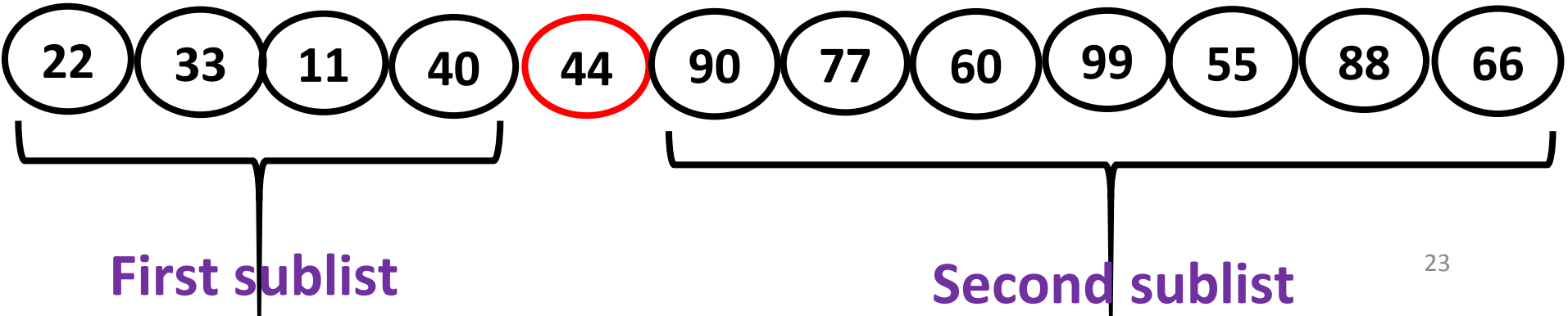
Quicksort/ Reduction Step

- 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**

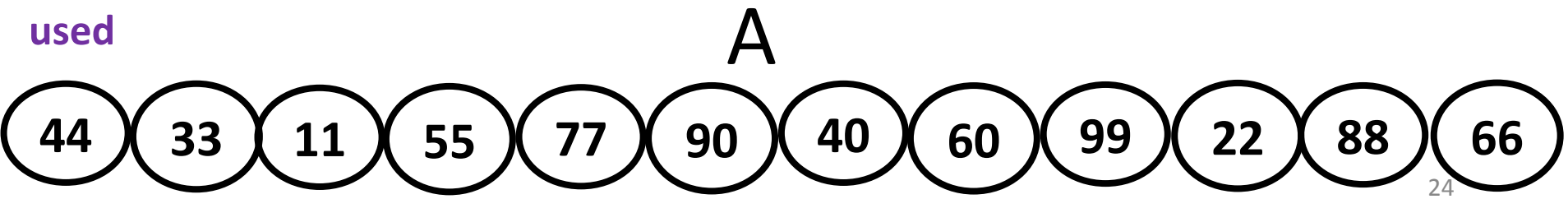
LOWER and UPPER

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.



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



N=12 elements,

Thus,

Boundary values are ()?

1 and 12.

Now,

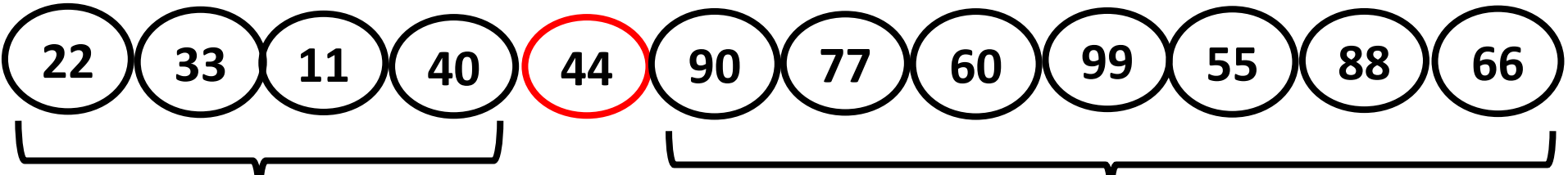
1 and 12 should be Stacked

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

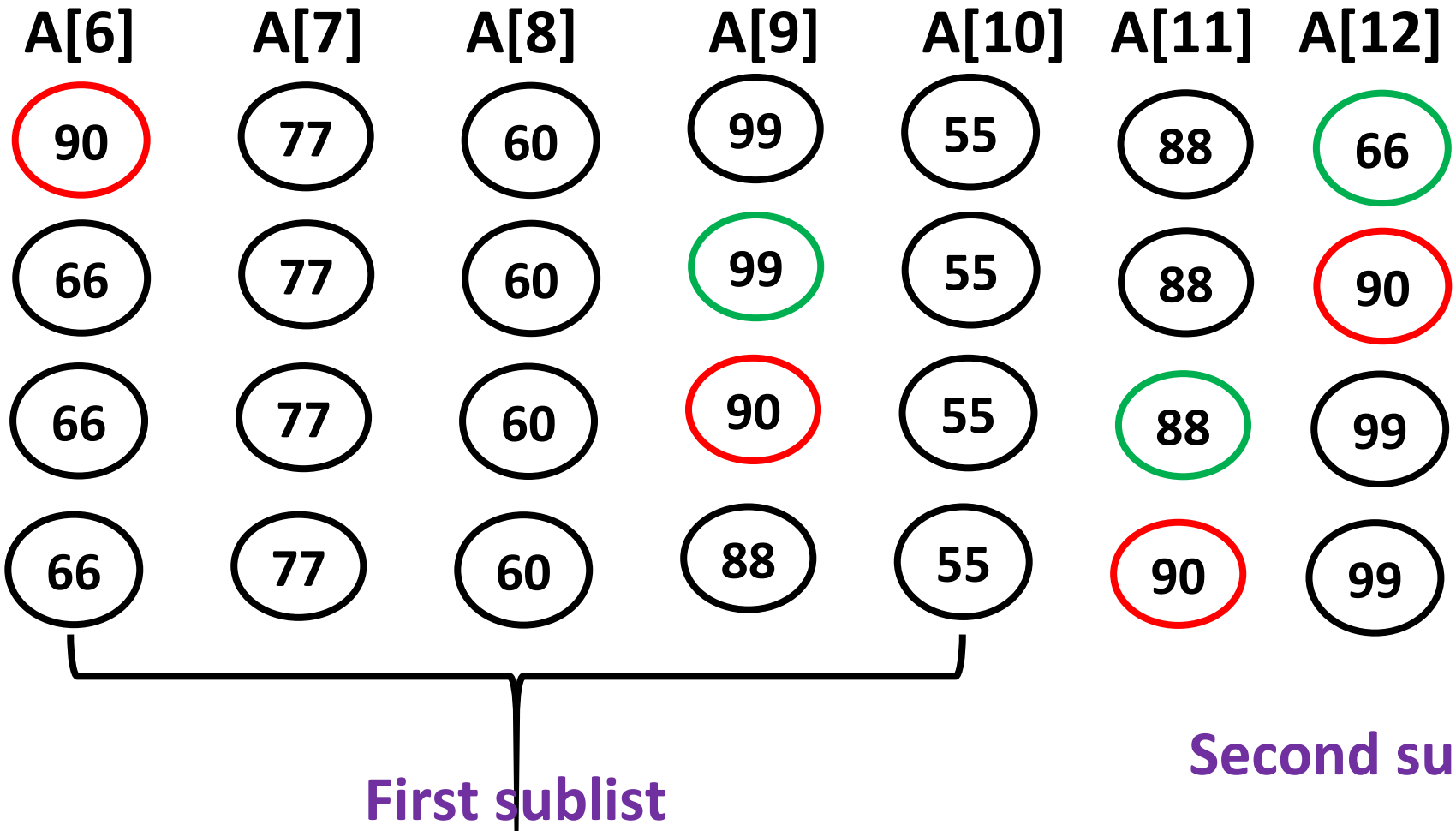
- Accordingly, the algorithm pushes the boundary values
  - 1 and 4 of the first sublist, and
  - 6 and 12 of the second sublist on to the STACK to yield

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**