Scripting Language

UNIT-III

K.RaviKanth, Ast.Prof., Dept. of CSE, IIIT- RGUKT- Basara, T.S, IND.

IIIT BASARA

SYLLABUS

- **UNIT – III [11 Lectures]**

- Python memory model: Mutable and Immutable
- Algorithm Complexity Factors
- Algorithm Asymptotic notations - Big O, Theta, Omega
- Sorting – Selection sort, Insertion sort, Bubble sort, Merge sort
- Searching – Linear searching, binary searching

python
Programming Language

Algorithm Complexity Factors

- Algorithm:
  - It is written in simple English Language
  - Algorithm Steps are unique and should be self explanatory
  - An Algorithm Must have at least one input
  - An Algorithm Must have at least one output
  - An Algorithm has finite number of steps

K.RaviKanth, Ast.Prof., Dept. of CSE, IIIT- RGUKT- Basara, T.S, IND.

Types of Algorithms

- **Simple Recursive Algorithms** Ex: Searching an element in a list
- **Backtracking Algorithms** Ex: Depth-first recursive search in a tree
- **Divide and conquer Algorithm** Ex: Quick Sort & Merge Sort
- **Dynamic Programming Algorithms** Ex: Generation of Fibonacci Series
- **Greedy Algorithms** Ex: Counting currency
- **Branch and Bound Algorithms** Ex: Travelling salesman(Visiting each city once and minimize the total distance travelled
- **Brute force Algorithm** Ex: Finding the best path for a travelling salesman
- **Randomized Algorithms** Ex: Using a random number to choose a pivot in quick sort

Algorithm Asymptotic notations

- Accurate measurements of Time Complexity is possible with asymptotic notations (meaningful approximations of functions)

- There are 5 important asymptotic notations

1. Big Oh Notation (O)
2. Big Omega Notation (Ω)
3. Big Theta Notation (Θ)
4. Little oh Notation (o)
5. Little omega Notation (ω)

K.RaviKanth, Ast.Prof., Dept. of CSE, IIIT- RGUKT- Basara, T.S, IND.

Asymptotic Complexity

Notation	Meaning
Big –Oh	f(n)=O(g(n)) if f(n) is asymptotically <=g(n)
Big - Omega	f(n)= Ω(g(n)) if f(n) is asymptotically >=g(n)
Big- Theta	f(n)=ϴ(g(n)) if f(n) is asymptotically =g(n)
Little – oh	f(n)=o(g(n)) if f(n) is asymptotically strictly < <g(n)
Little - Omega	f(n)=ω(g(n)) if f(n) is asymptotically > g(n)

K.RaviKanth, Ast.Prof., Dept. of CSE, IIIT- RGUKT- Basara, T.S, IND.

Big Oh Notation

- It is a characterization scheme that allows us to measure properties of algorithms
- There is a complexity function f(n) of an algorithm which increases as n increases. It is the rate of increase of f(n) which we want to examine

Let f(n) & g(n) be two functions, $n>0$

then $f(n) = O[g(n)]$,

read as 'f of n is big oh of g of n' or f(n) is of the order of g(n)

Based on 'Big oh ' notation the algs can be classified as,

1. Constant Time Algorithm - O(1)
2. Logarithmic Time Algorithm – O(log n) or O(n log n)
3. Linear Time Algorithm – O(n)
4. Polynomial Time Algorithm – O(nK)

Quadratic Time O(n2) Cubic Time(n3) Exponential Time (2n)

Big-O Notation is used in Computer Science to describe the performance of Sorts and Searches on arrays.

The Big-O is a proportion of speed or memory usage to the size of the array.

- n
- n log n
- n2

Compare the values of these sorts.

- Which sort(s) is/are quickest when n is 100?
- Which sort(s) is/are quickest when n is 1 Billion?

Big O

Name	Best	Average	Worst	Memory	Stable	Method
Quicksort	$n \log n$	$n \log n$	n^2	$\log n$	Depends	Partitioning
Merge sort	$n \log n$	$n \log n$	$n \log n$	Depends; worst case is n	Yes	Merging
In-place Merge sort	—	—	$n (\log n)^2$	1	Yes	Merging
Heapsort	$n \log n$	$n \log n$	$n \log n$	1	No	Selection
Insertion sort	n	n^2	n^2	1	Yes	Insertion
Introsort	$n \log n$	$n \log n$	$n \log n$	$\log n$	No	Partitioning & Selection
Selection sort	n^2	n^2	n^2	1	No	Selection
Timsort	n	$n \log n$	$n \log n$	n	Yes	Insertion & Merging
Shell sort	n	$n(\log n)^2$ or $n^{3/2}$	Depends on gap sequence; best known is $n(\log n)^2$	1	No	Insertion
Bubble sort	n	n^2	n^2	1	Yes	Exchanging
Binary tree sort	n	$n \log n$	$n \log n$	n	Yes	Insertion
Cycle sort	—	n^2	n^2	1	No	Insertion
Library sort	—	$n \log n$	n^2	n	Yes	Insertion
Patience sorting	—	—	$n \log n$	n	No	Insertion & Selection
Smoothsort	n	$n \log n$	$n \log n$	1	No	Selection
Strand sort	n	n^2	n^2	n	Yes	Selection
Tournament sort	—	$n \log n$	$n \log n$	n[5]		Selection
Cocktail sort	n	n^2	n^2	1	Yes	Exchanging
Comb sort	n	$n \log n$	n^2	1	No	Exchanging
Gnome sort	n	n^2	n^2	1	Yes	Exchanging
Bogosort	n	$n \cdot n!$	$n \cdot n! \to \infty$	1	No	Luck

Sorting

- Sorting is one of the most common tasks performed by computers today
- It has been estimated that as much as one quarter of the running time of all the worlds computers is spent on performing various types of sorting.
- Sorting is used to arrange names and numbers in meaning ways
- Ex: Dictionary, Phonebook
- i.e., Sorting greatly improves the efficiency of Searching
- **Sorting** is a process that organizes a collection of data into either ascending or descending order.
- An **internal sort** requires that the collection of data fit entirely in the computer's main memory.
- We can use an **external sort** when the collection of data cannot fit in the computer's main memory all at once but must reside in secondary storage such as on a disk.

SORTING

Internal [Uses Main Memory]

Selection , Insertion, Bubble, Quick, Merge, Heap, Tree, Radix, Bucket, Binary, Address Calculation, Partition Exchange Sorting

External [Large in size]

It requires extended memory like magnetic tape, floppy disk so on

Note: As per our requirement we will analyze only internal sorting algorithms.

Any significant amount of computer output is generally arranged in some sorted order so that it can be interpreted.

Mainly used Sorting Algorithms

- There are many sorting algorithms, such as:
  - Selection Sort
  - Insertion Sort
  - Bubble Sort
  - Merge Sort
- Majority of programming projects use a sort somewhere, and in many cases, the sorting cost determines the running time.

K.RaviKanth, Ast.Prof., Dept. of CSE, IIIT- RGUKT-Basara, T.S, IND.

Selection Sort

- It is one of the easiest ways to sort data.Rather than swapping neighbours continuously like the Bubble sort
- The list is divided into two sublists, *sorted* and *unsorted*, which are divided by an imaginary wall.
- This alg. Finds the samllest ele. of the array & interchanges it with the ele. in the first position of the array
- After that, it re-examines the remaining ele.s in the array to find the second smallest ele. Which is interchanged with the ele. in the second position of the array
- After each selection and swapping, the imaginary wall between the two sublists move one element ahead, increasing the number of sorted elements and decreasing the number of unsorted ones.
- This process continues until all the ele.s are placed in their proper order,which is to be defined by the user i.e., ascending or descending order
- A list of *n* elements requires *n-1* passes to completely rearrange the data.

26	54	93	17	77	31	44	55	20	93 is largest
26	54	20	17	77	31	44	55	93	77 is largest
26	54	20	17	55	31	44	77	93	55 is largest
26	54	20	17	44	31	55	77	93	54 is largest
26	31	20	17	44	54	55	77	93	44 is largest stays in place
26	31	20	17	44	54	55	77	93	31 is largest
26	17	20	31	44	54	55	77	93	26 is largest
20	17	26	31	44	54	55	77	93	20 is largest
17	20	26	31	44	54	55	77	93	17 ok list is sorted

```
# Source Code of Selection sort

def selection_sort(my_list):     # Function Definition
    for cur_pos in range(len(my_list)):
        min_pos = cur_pos
        for scan_pos in range(cur_pos + 1, len(my_list)):
            if my_list[scan_pos] < my_list[min_pos]:
                min_pos = scan_pos
        temp = my_list[min_pos]            # Swapping the values
        my_list[min_pos] = my_list[cur_pos]
        my_list[cur_pos] = temp
    return my_list

size = int( input("Enter a size of the array: ") )
list1=[]
for i in range(0,size):
element=input("Enter a element: ")
list1.append(element)
print ("Entered elements into list: ",(list1) )
print ("After doing SELECTION SORT: ",selection_sort(list1)) #Function call
```


K.RaviKanth, Ast.Prof., Dept. of CSE, IIIT- RGUKT- Basara, T.S, IND.

Selection Sort – Analysis

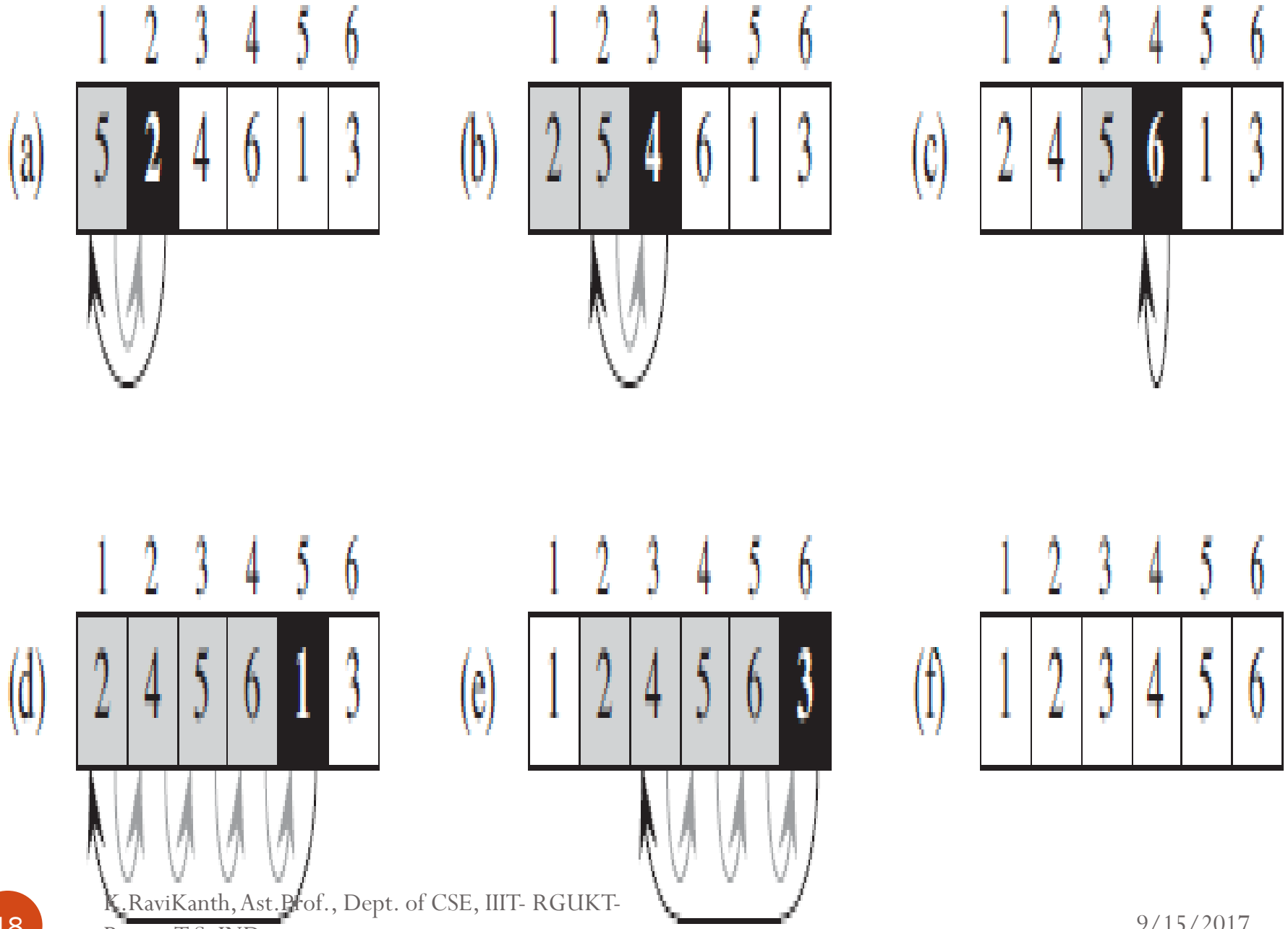
- Selection sort best case: O(n2)
- Selection sort worst case: O(n2)
- Selection sort average case: O(n2)

- **Advantages:** The selection sort is easy to understand & this makes it easy to implement the alg. correctly & thus makes the sorting procedure easy to write
- **Dis-Advantage:**
- It uses Internal Sorting mechanisms, thus requires large amount of memory
- The programmers performance would become quite slow on large amount sod data.
- ***Video Link: https://www.youtube.com/watch?v=Ns4TPTC8whw***

Insertion Sort

- Here the elements are inserted at appropriate places, by swapping the elements
- The greater numbers are shifted towards the end locations of the array and smaller elements are shifted at the beginning of the array
- It s one of the most common sorting tech. used by card players
- The list is divided into two parts: sorted and unsorted.
- In each pass, the first element of the unsorted part is picked up, transferred to the sorted sublist, and inserted at the appropriate place.
- A list of *n* elements will take at most *n-1* passes to sort the data.

K.RaviKanth, Ast.Prof., Dept. of CSE, IIIT- RGUKT- Basara, T.S, IND.



K.RaviKanth, Ast.Prof., Dept. of CSE, IIIT- RGUKT-Basara, T.S, IND.

```
# Source code of Insertion Sort
def insertionSort(arr):
    # Traverse through 1 to len(arr)
    for i in range(1, len(arr)):
        key = arr[i]
        j = i-1
        while j >=0 and key < arr[j] :
                arr[j+1] = arr[j]
                j -= 1
        arr[j+1] = key
    return arr

size = int(input("Enter a size of the array: "))
arr=[]
for i in range(0,size):
  element=int(input("Enter a element: "))
  arr.append(element)
print ("Entered elements into list: ",(arr))
print ("Sorted array is:",insertionSort(arr))
```

Time Complexity

- Best-case:O(n)
- Worst-case:$O(n^2)$
- Average-case:$O(n^2)$
- So, Insertion Sort is $O(n^2)$

- **Advantages:**
  - Good for less volumes of data
  - Better than Bubble & Selection Tech.s
  - Selection sort is 60% faster than Bubble sort
  - Insertion sort is 40% faster than Selection sort
- **Dis-Advantages:**
  - It is inefficient for large volumes of data

K.RaviKanth, Ast.Prof., Dept. of CSE, IIIT- RGUKT- Basara, T.S, IND.

Bubble Sort

- The bubble sort is the simplest but not very efficient of the sorts
- It compares the adjacent elements in a list of data elements and swaps them if they are not in an order
- Each pair of adj. ele.s is compared and swapped until the smallest or largest ele. Bubbles up to the top
- After that, the wall moves one element ahead, increasing the number of sorted elements and decreasing the number of unsorted ones.
- Each time an element moves from the unsorted part to the sorted part one sort pass is completed.
- Repeat this process till all the ele.s are in sorted order
- This technique is often referred to as adj. sort or Sinking sort
- Given a list of n elements, bubble sort requires up to n-1 passes to sort the data.

K.RaviKanth, Ast.Prof., Dept. of CSE, IIIT- RGUKT- Basara, T.S, IND.

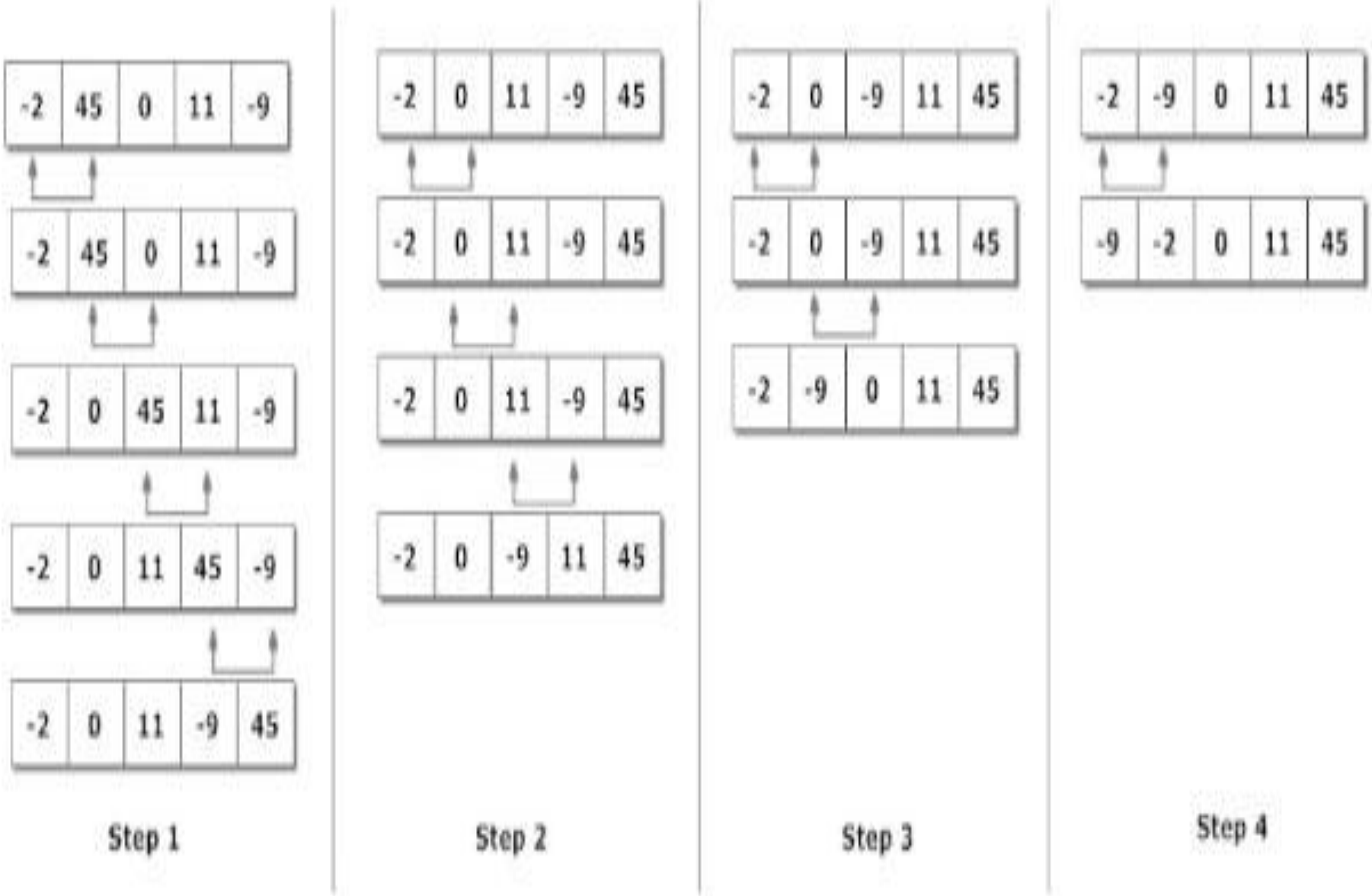


Figure: Working of Bubble sort algorithm

K.RaviKanth, Ast.Prof., Dept. of CSE, IIIT- RGUKT-Basara, T.S, IND.

```
# Source code of Bubble Sort

def bubble_sort(items):
    """ Implementation of bubble sort """
    for i in range(len(items)):
        for j in range(len(items)-1-i):
            if items[j] > items[j+1]:
                items[j], items[j+1] = items[j+1],
  items[j]      # Swap!
    return items


size = int(input("Enter a size of the array: "))
list1=[]
for i in range(0,size):
  element=int(input("Enter a element: "))
  list1.append(element)
print ("Entered elements into list: ",(list1))

print ("After doing BUBBLE SORT: ",bubble_sort(list1))
```


K.RaviKanth, Ast.Prof., Dept. of CSE, IIIT- RGUKT- Basara, T.S, IND.

Time Complexity

- Best-case:O(n)
- Worst-case:O(n^2)
- Average-case:O(n^2)
- So, Bubble Sort is O(n^2)

- In the i pass, i largest ele. Bubbles upto the i position
- Each pass places a new ele. Into its proper position, it requires (N-1) passes for N ele.s
- The i pass makes N-i comaprisions

- ***Video Link:*** **http://www.youtube.com/watch?v=lyZQPjUT5B4**

K.RaviKanth, Ast.Prof., Dept. of CSE, IIIT- RGUKT- Basara, T.S, IND.

Bubble Sort – Analysis

- **Advantages:**
  - It is fairly easy to write (about 5 LOC) LOC-Lines of code
  - It is a fairly easy to undestand
  - It is a good choice in situations where a few ele.s have to be added to an already sorted list of ele.s

- **Dis-advantage:**
  - It is relatively slow alg. taking $O(n^2)$ T.C
  - It should be avoided on large tables

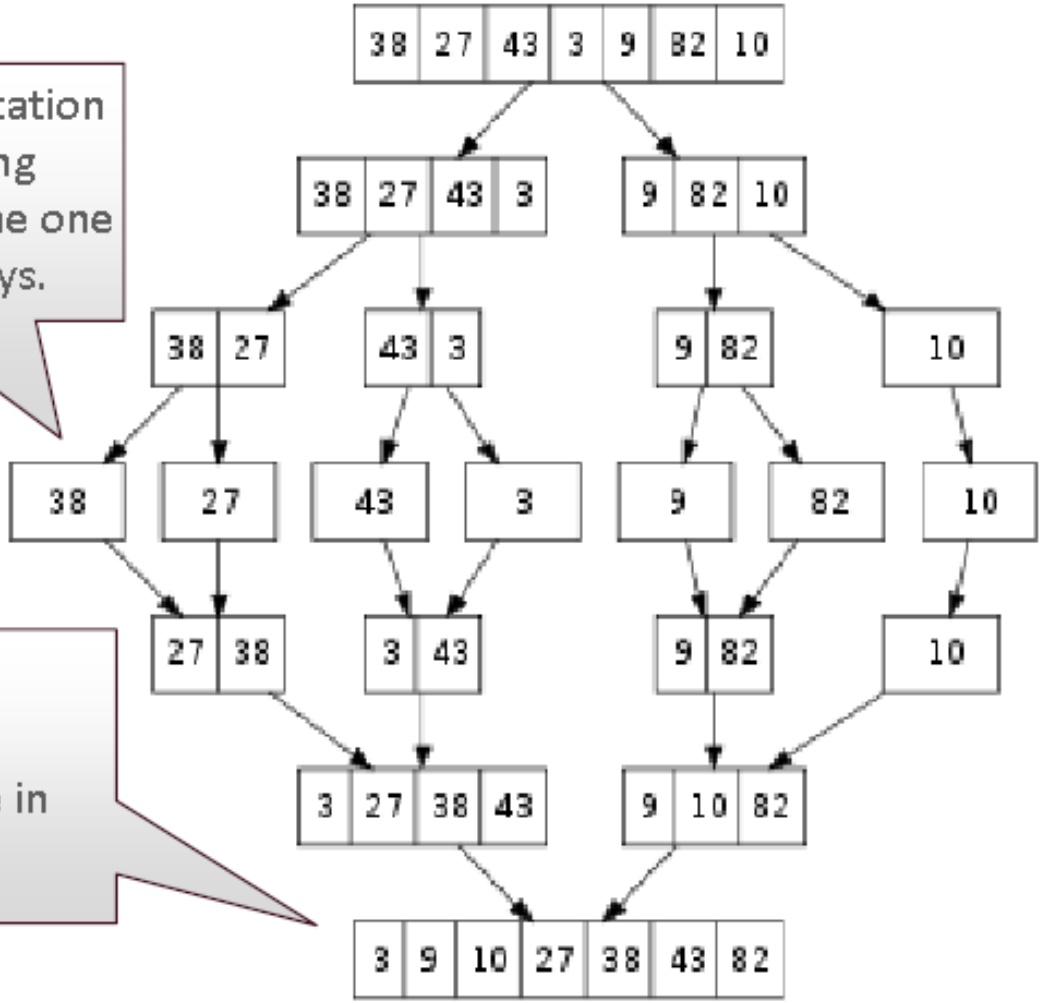
K.RaviKanth, Ast.Prof., Dept. of CSE, IIIT- RGUKT- Basara, T.S, IND.

Merge sort

- Merge sort technique sorts a given set of values by combining two sorted arrays into one larger sorted array [Divide & Conquer Tech.]
- Consider the sorted array A which contains p elements & sorted array B which contains q elements, the merge sort technique combines the elements of A & B into a single sorted array C with p+q elements
- The 1st data item of array A is compared with the 1st data item of array B
- If data item of array A < that of B, then that data item from A is moved to the new array C
- If the data item of array B < that of A, then that data item form B is moved to the new array C
- This comparison of data items continues until one of the array ends

Visual Representation of Merge Sort

This visual representation shows the array being broken down into the one element sorted arrays.



They are "merged" together again and again until they are in one sorted array.

Time Complexity

- Best case : O (n log n)
- Average case : O(n log n)
- Worst case: O(n log n)

- **Advantages:**
  - Faster as T.C is O(n log n)
  - Slightly faster than Heap sort (but requires twice the memory of Heapsort)
- **Dis-Advantages:**
  - Requires additional memory
  - In merge sort, splitting is easy but merging is a bit difficult
  - Requires twice the memory of heap sort
  - ***Video Link:*** **https://www.youtube.com/watch?v=XaqR3G_NVoo**

K.RaviKanth, Ast.Prof., Dept. of CSE, IIIT- RGUKT- Basara, T.S, IND.

IIIT BASARA

Infosys | Campus Connect

For Details Contact Me @ :

9247448766

ravikanth27787@gmail.com

python

Programming Language