RGPVNOTES.IN

Program : **B.Tech**

Subject Name: **Data Structure**

Subject Code: **IT-303**

Semester: **3rd**

Subject Notes
IT 302- Data Structure

Unit-2

2.1 Arrays & Structure Introduction

The simplest type of data structure is a linear (or one dimensional) array.

Array is set of homogenous data items which are stored in contiguous memory locations under a common name and with sequence of indices starting from 0.An array size is fixed and therefore requires a fixed number of memory locations. It is a list of finite number n of similar data referenced respectively by a set of n consecutive numbers, usually 1, 2, 3 . . . . . n. if we choose the name A for the array, then the elements of A are denoted by subscript notation

A 1, A 2, A 3 . . . . A n

or by the parenthesis notation

A (1), A (2), A (3) . . . . . A (n)

or by the bracket notation

A [1], A [2], A [3] . . . . . A [n]

Structures

Structure is a collection of variables of different data types under a single name. It is similar to a class in that, both holds a collecion of data of different data types. For example**:** You want to store some information about a person: his/her name, citizenship number and salary. You can easily create different variables name, citNo, salary to store these information separately.

The **struct** keyword defines a structure type followed by an identifier (name of the structure). Then inside the curly braces, you can declare one or more members (declare variables inside curly braces) of that structure. For example:

```
struct Person
{
  char name[50];
  int age;
  float salary;
};
```

Here a structure person is defined which has three members: name, age and salary.

Once you declare a structure person as above. You can define a structure variable as: **Person bill;** Here, a structure variable bill is defined which is of type structure Person.The members of structure variable is accessed using a **dot (.)** operator. Suppose, you want to access age of structure variable bill and assign it 50 to it. You can perform this task by using following code below:

```
bill.age = 50;
```

2.2 Declaration of array

A linear array int A[8] consisting of numbers is pictured in following figure.
int is datatype , A is arrayname, 8 is the size of array.

Fig 2.1 Array Declaration

Suppose that A is a variable that refers to an array. Then the element at index k in A is referred to as A[k]. The first element is A[0], the second is A[1], and so forth. Each element is of same size. Elements are stored contiguously, with the first element stored at the smallest memory address (called the base address).

2.3 One-dimensional Array or linear array requires only one index to access an element. Ex: int A[10]

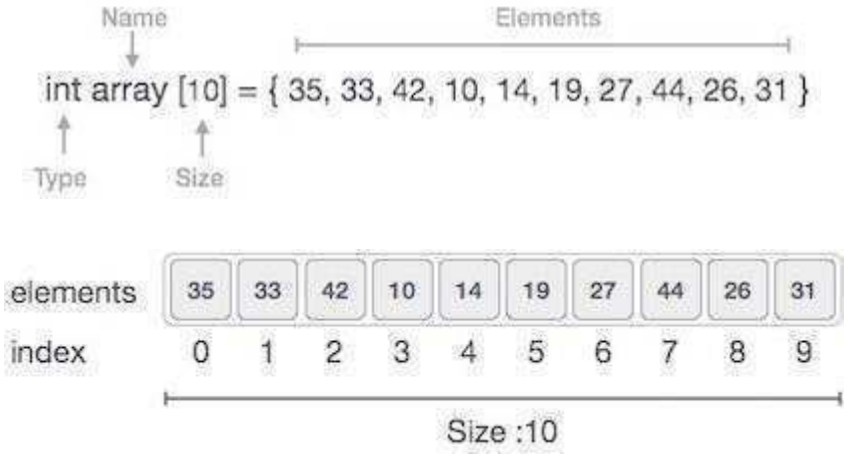


Fig 2.2 1-D Array

2.4 Address calculation in one-dimensional Array

Since array elements are stored in contiguous memory locations, the computer needs to not to know the address of every element but the address of only first element. The address of first element is called base address of array. Given the address of first element, address of any other element is calculated using the formula:-

A [I]=Base(A)+ w (I – LB)

Base(A)- address of first element I = index

w=word size **LB = Lower Bound.**

Example:

One Dimension Array

Given the base address of an array B[1300.....1900] as 1020 and size of each element is 2 bytes in the memory. Find the address of B[1700].

Solution:

The given values are: B = 1020, LB = 1300, W = 2, I = 1700

Address of A [I] = B + W * (I – LB)

= 1020 + 2 * (1700 – 1300)

= 1020 + 2 * 400

= 1020 + 800

= 1820 [Ans]

2.5 Two-Dimensional Array

It requires two indices to access an element. **A[m][n]** Here, the first index is the row number; the second index the number within the row. Array of Arrays is 2D array. ex. Matrix in mathematics and tables in business applications.

2.6 2D Array Representation in Memory

- Column-major order
- Row-major order

Arrays may be represented in Row-major form or Column-major form.
In Row-major form, all the elements of the first row are stored, then the elements of the second row and so on up to the last row.

In Column-major form, all the elements of the first column are stored, then the elements of the second column and so on up to the last column.

Ex:
8	6	5	4
2	1	9	7
3	6	4	2

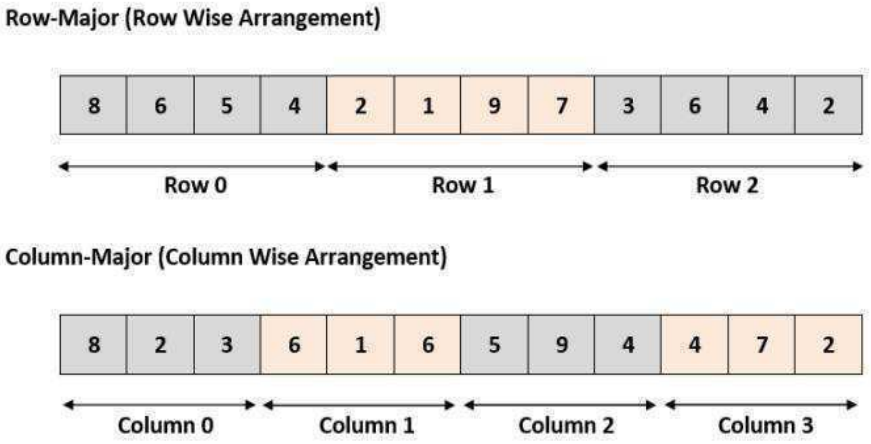


Fig 2.3 Memory representation

2.7 Address calculation in two-dimensional Array

(i)Row major order:- A[i][j]=base(A) +w[n(I - L1) +(j – L2)]
Where array is m x n matrix, L1 is the lower bound of row; L2 is the lower bound of column,w is word size, n is no of columns, base(A) is base address,i is row index, j is column index

(ii)Column major order:- A[i][j]=base(A)+ w[m(j- L2) +(i – L1)]
Where array is m x n matrix, L1 is the lower bound of row; L2 is the lower bound of column,w is word size, n is no of rows, base(A) is base address , i is row index, j is column index

Examples:
Q 1. An array X [-15.....10, 15.....40] requires one byte of storage. If beginning location is 1500 determine the location of X [15][20].
Solution:
As you see here the number of rows and columns are not given in the question. So they are calculated as:
Number or rows say M = (Ur – Lr) + 1 = [10 - (- 15)] +1 = 26
Number or rows say N = (Uc – Lc) + 1 = [40 - 15)] +1 = 26

(i) Column Major Wise Calculation of above equation
The given values are: B = 1500, W = 1 byte, I = 15, J = 20, Lr = -15, Lc = 15,
M = 26

Address of A [I][J] = base(A)+ w[m(Ij- L2) +(i – L1)]

= 1500 + 1 * [26 * (20 - 15)+(15 - (-15))]
= 1500 + 1 * [26 * 5 + 30]
= 1500 + 1 * [160]
= 1660 [Ans]

(ii) Row Major Wise Calculation of above equation
The given values are: B = 1500, W = 1 byte, I = 15, J = 20, Lr = -15, Lc = 15,
N = 26

Address of A [I][J] = base(A) +w[n(I - L1) +(j – L2)]
= 1500 + 1* [26 * (20 – 15) + (15 - (-15)))]
= 1500 + 1 * [26 *5 + 30]
= 1500 + 1 * [780 + 5]
= 1500 + 785
= 2285 [Ans]

2.8 Operations performed on the array:

a) Traversing: means to visit all the elements of the array in an operation is called traversing.
b) Insertion: means to put values into an array
c) Deletion: to delete a value from an array.
d) Sorting: Re-arrangement of values in an array in a specific order (Ascending or Descending) is called sorting.
e) Searching: The process of finding the location of a particular element in an array is called searching.
f) Merging : - Merging means joining two lists or two arrays in one single array.

a) Traversing in Linear Array:

It means processing or visiting each element in the array exactly once; Let **'A'** is an array stored in the computer's memory. If we want to display the contents of **'A'**, it has to be traversed i.e. by accessing and processing each element of **'A'** exactly once.

Algorithm: (Traverse a Linear Array) Here **LA** is a Linear array with lower boundary **LB** and upper boundary **UB**. This algorithm traverses **LA** applying an operation Process to each element of **LA**.

1. [Initialize counter.] Set K=LB.
2. Repeat Steps 3 and 4 while K≤UB.
3. [Visit element.] Apply PROCESS to LA[K].
4. [Increase counter.] Set k=K+1.
   [End of Step 2 loop.]
5. Exit.

The alternate algorithm for traversing (using for loop) is :

Algorithm: (Traverse a Linear Array) This algorithm traverse a linear array **LA** with lower bound **LB** and upper bound **UB**.

1. Repeat for K=LB to UB
   Apply PROCESS to LA[K].
   [End of loop].
2. Exit.

b) 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.

Bubble Sort:

The technique we use is called "Bubble Sort" because the bigger value gradually bubbles their way up to the top of array like air bubble rising in water, while the small values sink to the bottom of array. This technique is to make several passes through the array. On each pass, successive pairs of elements are compared. If a pair is in increasing order (or the values are identical), we leave the values as they are. If a pair is in decreasing order, their values are swapped in the array.

Pass = 1	Pass = 2	Pass = 3	Pass=4
2 1 5 7 4 3	**1 2** 5 4 3 7	**1 2** 4 3 5 7	**1 2** 3 4 5 7
1 **2 5** 7 4 3	1 **2 5** 4 3 7	1 **2 4** 3 5 7	1 **2 3** 4 5 7
1 2 **5 7** 4 3	1 2 **5 4** 3 7	1 2 **4 3** 5 7	1 2 3 4 5 7
1 2 5 **7 4** 3	1 2 4 **5 3** 7	1 2 3 4 5 7	
1 2 5 4 **7 3**	1 2 4 3 5 7		
1 2 5 4 3 7			

- Underlined pairs show the comparisons. For each pass there are size-1 comparisons.
- Total number of comparisons= (size-1)²

Algorithm: (Bubble Sort) BUBBLE (DATA, N)
Here DATA is an Array with N elements. This algorithm sorts the elements in DATA.

1. for pass=1 to N-1.
2. for (i=0; i<= N-Pass; i++)
3. If DATA[i]>DATA[i+1], then:
   Interchange DATA[i] and DATA[i+1].
   [End of If Structure.]
   [End of inner loop.]
   [End of Step 1 outer loop.]
4. Exit.

c).Searching in Linear Array:

You can perform a search for an array element based on its value or its index.

Algorithm

Consider **LA** is a linear array with **N** elements and **K** is a positive integer such that **K<=N**. Following is the algorithm to find an element with a value of ITEM using sequential search.

1. Start
2. Set J = 0
3. Repeat steps 4 and 5 while J < N
4. IF LA[J] is equal ITEM THEN GOTO STEP 6
5. Set J = J +1
6. PRINT J, ITEM
7. Stop

d).Insertion in Linear Array:

Insert operation is to insert one or more data elements into an array. Based on the requirement, a new element can be added at the beginning, end, or any given index of array.

Let **LA** be a Linear Array (unordered) with **N** elements and **K** is a positive integer such that **K<=N**. Following is the algorithm where ITEM is inserted into the K^{th} position of LA .

1. Start
2. Set J = N
3. Set N = N+1
4. Repeat steps 5 and 6 while J >= K
5. Set LA[J+1] = LA[J]
6. Set J = J-1
7. Set LA[K] = ITEM
8. Stop

e)Deletion in Linear Array:

Deletion refers to removing an existing element from the array and re-organizing all elements of an array.

Algorithm

Consider **LA** is a linear array with **N** elements and **K** is a positive integer such that **K<=N**. Following is the algorithm to delete an element available at the K^{th} position of LA.

1. Start
2. Set J = K

3. Repeat steps 4 and 5 while J < N
4. Set LA[J] = LA[J + 1]
5. Set J = J+1
6. Set N = N-1
7. Stop

2.9 Polynomial representation & Evaluation

A polynomial p(x) is the expression in variable x which is in the form ($ax^n + bx^{n-1} + ... + jx + k$), where a, b, c …., k fall in the category of real numbers and 'n' is non negative integer, which is called the degree of polynomial.

Polynomial expression consists of two parts:

- one is the coefficient
- other is the exponent

$10x^2 + 26x$, here 10 and 26 are coefficients and 2, 1 are its exponential value.

Polynomial can be represented in the various ways. These are:

- By the use of arrays
- By the use of Linked List

2.9.1 Representation of a Polynomial: A polynomial is an expression that contains more than two terms. A term is made up of coefficient and exponent. An example of polynomial is P(x) = $4x^3+6x^2+7x+9$

1. A polynomial thus may be represented using arrays or linked lists. Array representation assumes that the exponents of the given expression are arranged from 0 to the highest value (degree), which is represented by the subscript of the array beginning with 0. The coefficients of the respective exponent are placed at an appropriate index in the array. The array representation for the above polynomial expression is given below:

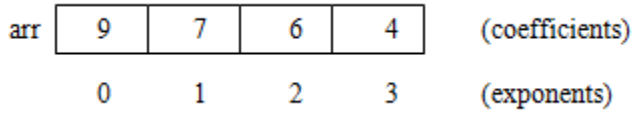


Fig 2.4 array representation

```
struct polynomial
{
int coefficient;
int exponent;
struct polynomial *next;
};
```

2. A polynomial may also be represented using a linked list. A structure may be defined such that it contains two parts- one is the coefficient and second is the corresponding exponent. The structure definition may be given as shown below:

```
struct polynode {
      float coeff;
      int exp;
      polynode* link;
  } * p;
```

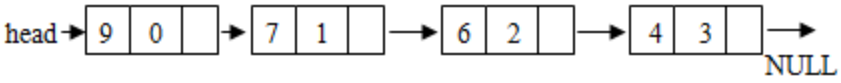


Fig 2.5 Linked representation

2.10 Addition of two Polynomials:

For adding two polynomials using arrays is straightforward method, since both the arrays may be added up element wise beginning from 0 to n-1, resulting in addition of two polynomials. Addition of two polynomials using linked list requires comparing the exponents, and wherever the exponents are found to be same, the coefficients are added up. For terms with different exponents, the complete term is simply added to the result thereby making it a part of addition result. The complete program to add two polynomials is given in subsequent section.

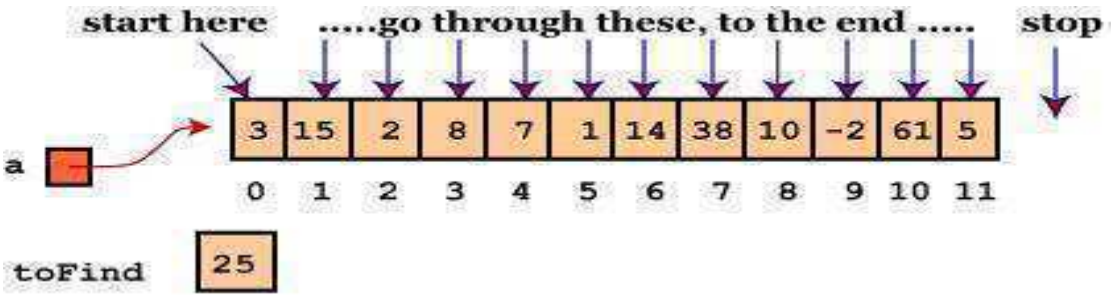
2.11 Searching

Linear Search

Linear search or sequential search is a method for finding a particular value in a list that consists of checking every one of its elements, one at a time and in sequence, until the desired one is found. Linear search is the simplest search algorithm.

How Linear Search works

Linear search in an array is usually programmed by stepping up an index variable until it reaches the last index. This normally requires two comparisons for each list item: one to check whether the index has reached the end of the array, and another one to check whether the item has the desired value.



Linear Search Algorithm

1.Repeat For J = 1 to N

2.If (ITEM == A[J]) Then

3.Print: ITEM found at location J

4, Return [End of If]

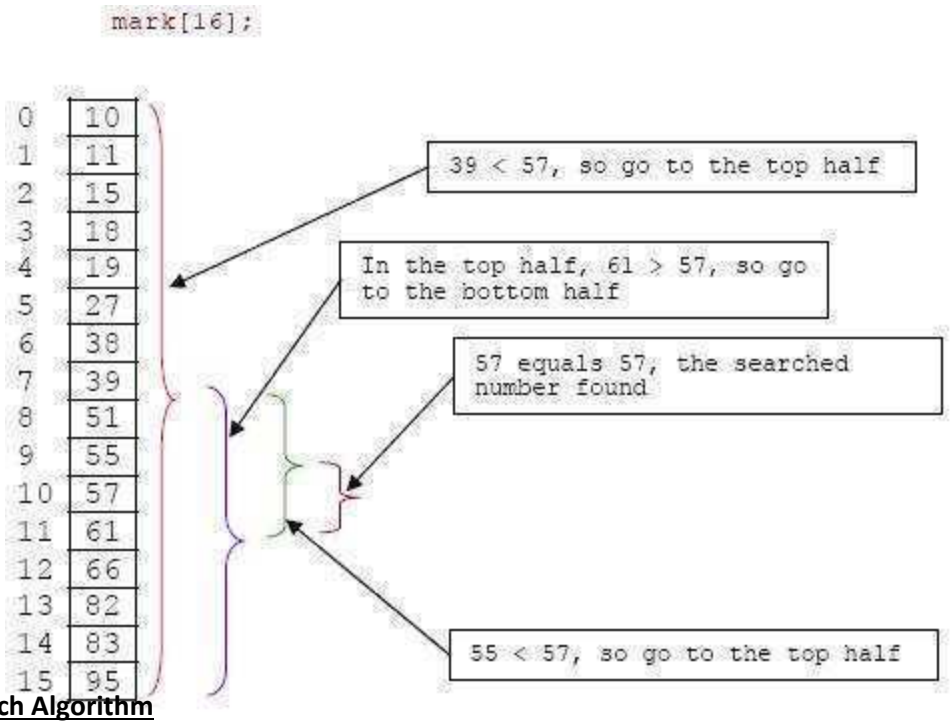
[End of For Loop]

5. If (J > N) Then
Print: ITEM doesn't exist
[End of If]

6.Exit

Binary Search A binary search or half-interval search algorithm finds the position of a specified input value (the search "key") within an array sorted by key value. For binary search, the array should be arranged in ascending or descending order. In each step, the algorithm compares the search key value with the key value of the middle element of the array. If the keys match, then a matching element has been found and its index is returned. Otherwise, if the search key is less than the middle element's key, then the algorithm repeats its action on the sub-array to the left of the middle element or, if the search key is greater, on the sub-array to the right. If the remaining array to be searched is empty, then the key cannot be found in the array and a special "not found" indication is returned.

How Binary Search Works



Binary Search Algorithm

1.Set BEG = 1 and END = N

2.Set MID = (BEG + END) / 2

3.Repeat step 4 to 8 While (BEG <= END) and (A[MID] ≠ ITEM)

4.If (ITEM < A[MID]) Then

5.Set END = MID – 1

6.Else if (ITEM > A[MID])

7.Set BEG = MID + 1

8.Else If (A[MID] == ITEM) Then

9.Print: ITEM exists at location MID

10.Else

11.Print: ITEM doesn't exist

[End of If]

13. Exit

2.12 Sorting

Sorting is nothing but storage of data in sorted order, it can be in ascending or descending order. There are so many things in our real life that we need to search, like a particular record in database, roll numbers in merit list, a particular telephone number, any particular page in a book etc.

TYPES OF SORTING

1. An internal sort is any data sorting process that takes place entirely within the main memory of a computer. This is possible whenever the data to be sorted is small enough to all be held in the main memory.

2. External sorting is a term for a class of sorting algorithms that can handle massive amounts of data. External sorting is required when the data being sorted do not fit into the main memory of a computing device (usually RAM) and instead they must reside in the slower external memory (usually a hard drive). External sorting typically uses a hybrid sort-merge strategy. In the sorting phase, chunks of data small enough to fit in main memory are read, sorted, and written out to a temporary file. In the merge phase, the sorted sub files are combined into a single larger file.

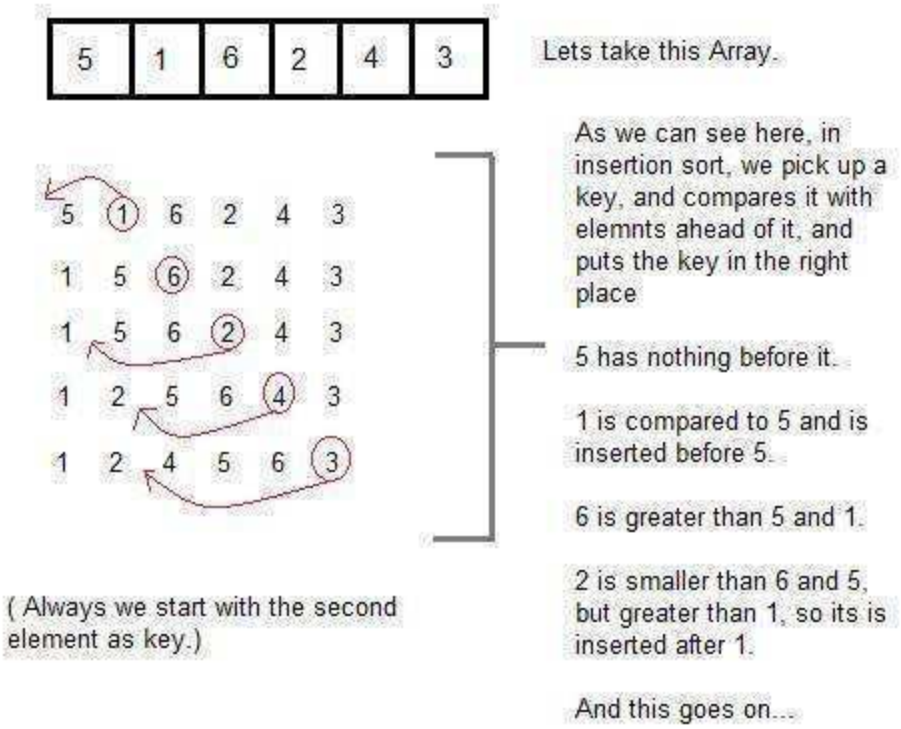
2.12.1. Insertion sort

It is a simple sorting algorithm that builds the final sorted array (or list) one item at a time.

Simple implementation

1. Efficient for small data sets
2. Stable; i.e., does not change the relative order of elements with equal keys
3. In-place; i.e., only requires a constant amount O(1) of additional memory space.

How Insertion Sort Works



Insertion Sort Algorithm

This algorithm sorts the array A with N elements.

1. Set A[0]=-12345(infinity i.e. Any large no)

2.Repeat step 3 to 5 for k=2 to n

3.Set key=A[k] And j=k-1

4.Repeat while key<A[j]

A) Set A[j+1]=A[j]

1. j=j-1

5.Set A[j+1]=key

6.Return

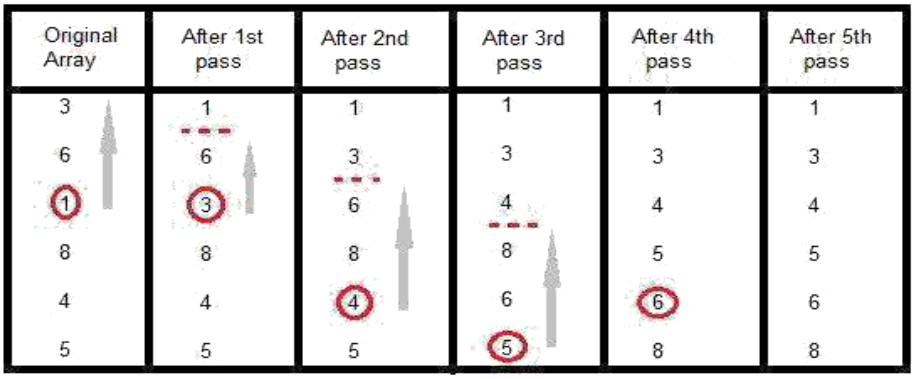
Algorithm	Worst Case	Average Case	Best Case
Insertion Sort	$n(n-1)/2 = O(n^2)$	$n(n-1)/4 = O(n^2)$	O(n)

2.12.2.Selection Sort

Selection sorting is conceptually the simplest sorting algorithm. This algorithm first finds the smallest element in the array and exchanges it with the element in the first position, then find the second smallest element and exchange it with the element in the second position, and continues in this way until the entire array is sorted

How Selection Sort works

In the first pass, the smallest element found is 1, so it is placed at the first position, then leaving first element, smallest element is searched from the rest of the elements, 3 is the smallest, so it is then placed at the second position. Then we leave 1 and 3, from the rest of the elements, we search for the smallest and put it at third position and keep doing this, until array is sorted



Selection Sort Algorithm

1. Repeat For J = 0 to N-1

2. Set MIN = J
3. Repeat For K = J+1 to N
4. If (A[K] < A[MIN]) Then
5. Set MIN = K

[End of If]
[End of Step 3 For Loop]
(i) Interchange A[J] and A[MIN] [End of Step 1 For Loop]
6.Exit

Complexity of Selection Sort Algorithm

The number of comparison in the selection sort algorithm is independent of the original order of the element. That is comparison during PASS 1 to find the smallest element, there are n-2 comparisons during PASS 2 to find the second sma and so on. Accordingly

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

Algorithm	Worst Case	Average Case	Best Case
Selection Sort	$n(n-1)/2 = O(n^2)$	$n(n-1)/2 = O(n^2)$	$O(n^2)$

2.12.3.Bubble Sort

Bubble Sort is an algorithm which is used to sort N elements that are given in a memory for eg: an Array with N elements. Bubble Sort compares all the element one by one and sort them based on their values. It is called Bubble sor with each iteration the smaller element in the list bubbles up towards the first place, just like a water bubble rises up to surface.

Sorting takes place by stepping through all the data items one-by-one in pairs and comparing adjacent data items and each pair that is out of order.

How Bubble Sort Works

Let us take the array of numbers "5 1 4 2 8", and sort the array from lowest number to greatest number using bubble so step, elements written in bold are being compared. Three passes will be required.

First Pass:

(5 1 4 2 8) → (1 5 4 2 8), Here, algorithm compares the first two elements, and swaps since 5 > 1.

(1 5 4 2 8) → (1 4 5 2 8), Swap since 5 > 4 (

1 4 5 2 8) → (1 4 2 5 8), Swap since 5 > 2

(1 4 2 5 8) → (1 4 2 5 8), Now, since these elements are already in order (8 > 5), algorithm does not swap them.

Second Pass:

(1 4 2 5 8) → (1 4 2 5 8)

(1 4 2 5 8) → (1 2 4 5 8), Swap since 4 > 2 (

1 2 4 5 8) → (1 2 4 5 8)

(1 2 4 5 8) → (1 2 4 5 8)

Now, the array is already sorted, but our algorithm does not know if it is completed. The algorithm needs one whole pa any swap to know it is sorted.

Third Pass:

(1 2 4 5 8) → (1 2 4 5 8) (1 2 4 5 8) → (1 2 4 5 8) (1 2 4 5 8) → (1 2 4 5 8) (1 2 4 5 8) → (1 2 4 5 8) (1 2 4 5 8) → (1 2 4 5 8)

Bubble Sort Algorithm

1.Repeat Step 2 and 3 for k=1 to n

2.Set ptr=1

3.Repeat while ptr<n-k

1. If (A[ptr] > A[ptr+1]) Then Interchange A[ptr] and A[ptr+1] [End of If]
2. ptr=ptr+1

[end of step 3 loop]

[end of step 1 loop]

4. Exit

Complexity of Bubble Sort Algorithm

In Bubble Sort, n-1 comparisons will be done in 1st pass, n-2 in 2nd pass, n-3 in 3rd pass and so on. So the total comparisons will be

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

2.12.4.Quick Sort

Quick Sort, as the name suggests, sorts any list very quickly. Quick sort is not stable search, but it is very fast and requires very less additional space. It is based on the rule of Divide and Conquer (also called partition-exchange sort). This algorithm divides the list into three main parts

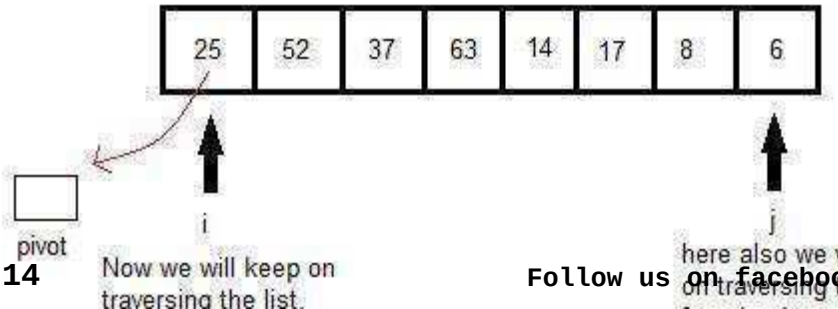
Elements less than the Pivot element Pivot element
Elements greater than the pivot element

How Quick Sort Works

In the list of elements, mentioned in below example, we have taken 25 as pivot. So after the first pass, the list will be changed like this.

6 8 17 14 **25** 63 37 52

Hence after the first pass, pivot will be set at its position, with all the elements smaller to it on its left and all the elements larger than it on the right. Now 6 8 17 14 and 63 37 52 are considered as two separate lists, and same logic is applied on them, and we keep doing this until the complete list is sorted.



Quick Sort Algorithm

Quick Sort (A, BEG, END):
Desciption: Here A is an unsorted array having N elements. BEG is the lower bound and END is the upper
bound.
1.
If (BEG < END) then
2. Find the element that divides the array into two parts using subfunction Partition( ).
3. Quick Sort (Left Half)
4. Quick Sort (Right Half)
[End of If]
5.Exit

Partition ( ):
1. Set LEFT = BEG, RIGHT = END and LOC = BEG
2. Beginning with the element pointed by RIGHT, scan the array from right to left, comparing each element with the element pointed by LOC until:
3.(a) Element smaller than the element pointed by LOC is found.
(b) Interchange elements pointed by LOC and RIGHT.
(c) If RIGHT becomes equal to LOC, terminate the subfunction Partition ( ).
Beginning with the element pointed by LEFT, scan the array from left to right, comparing each element with the element pointed by LOC until:
4.(a) Element greater than the element pointed by LOC is found.
(b) Interchange elements pointed by LOC and LEFT.
(c) If LEFT becomes equal to LOC, terminate the subfunction Partition ( ).
5.Exit

Complexity of Quick Sort Algorithm

The Worst Case occurs when the list is sorted. Then the first element will require n comparisons to recognize that it remains in the first position. Furthermore, the first sublist will be empty, but the second sublist will have n-1 elements. Accordingly the second element require n-1 comparisons to recognize that it remains in the second position and so on.

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

Algorithm	Worst Case	Average Case	Best Case
Quick Sort	$n(n+1)/2 = O(n^2)$	O(n logn)	O(n logn)

5. Merge Sort

Merge Sort follows the rule of Divide and Conquer. But it doesn't divide the list into two halves. In merge sort the unsorted list is divided into N sub lists, each having one element, because a list of one element is considered sorted. Then, it repeatedly merge these sub lists, to produce new sorted sub lists, and at lasts one sorted list is produced. Merge Sort is quite fast, and has a time complexity of O(n log n). It is also a stable sort, which means the equal elements are ordered in the same order in the sorted list.

How Merge Sort Works

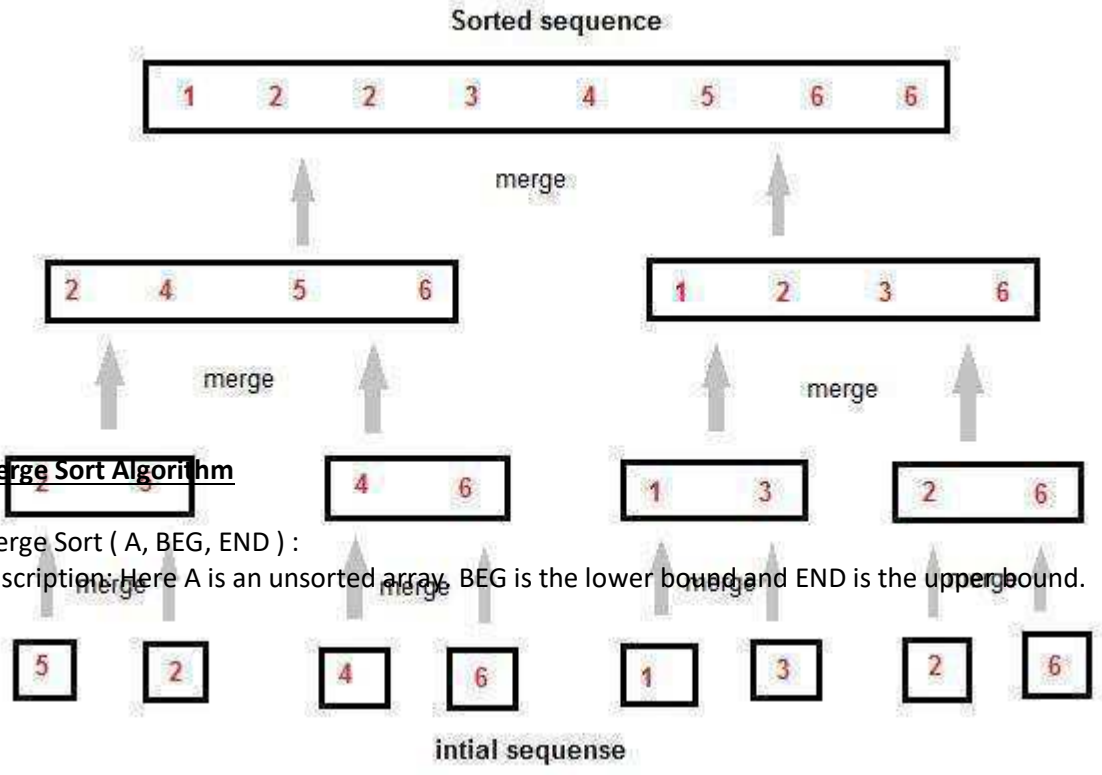
Suppose the array A contains 8 elements, each pass of the merge-sort algorithm will start at the beginning of the array A and merge pairs of sorted subarrays as follows.

PASS 1. Merge each pair of elements to obtain the list of sorted pairs.

PASS 2. Merge each pair of pairs to obtain the list of sorted quadruplets.

PASS 3. Merge each pair of sorted quadruplets to obtain the two sorted subarrays.

PASS 4. Merge the two sorted subarrays to obtain the single sorted array.



Merge Sort Algorithm

Merge Sort (A, BEG, END) :

Description: Here A is an unsorted array. BEG is the lower bound and END is the upper bound.

1. If (BEG < END) Then
2. Set MID = (BEG + END) / 2
3. Call Merge Sort (A, BEG, MID)
4. Call Merge Sort (A, MID + 1, END)
5. Call Merge Array (A, BEG, MID, END)
6. [End of If]

Merge Array (A, BEG, MID, END)
Description: Here A is an unsorted array. BEG is the lower bound, END is the upper bound and MID is the
middle value of array. B is an empty array.

1. Repeat For I = BEG to END
2. Set B[I] = A[I]
3. [End of For Loop]
4. Set I = BEG, J = MID + 1, K = BEG
5. Repeat While (I <= MID) and (J <=
6. If (B[I] <= B[J]) Then
7. Set A[K] = B[I]
8. Set I = I + 1 and K = K
9. Else
10. Set A[K] = B[J]
11. Set J = J + 1 and K = K
12. [End of If]
13. [End of While Loop]
14. If (I <= MID) Then
15. Repeat While (I <= MID)
16. Set A[K] = B[I]
17. Set I = I + 1 and K = K
18. [End of While Loop]
19. Else
20. Repeat While (J <= END)
21. Set A[K] = B[J]
22. Set J = J + 1 and K = K
23. [End of While Loop]
24. [End of If]
25. Exit

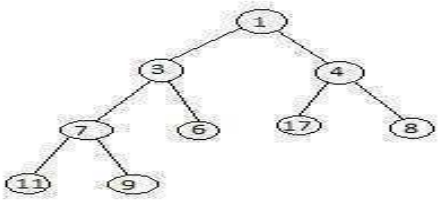
Complexity of Merge Sort Algorithm

Let f(n) denote the number of comparisons needed to sort an n-element array A using merge-sort algorithm. The algorithm requires at most logn passes. Each pass merges a total of n elements and each pass require at most n comparisons. Thus for both the worst and average case F(n) ≤ n logn. Thus the time complexity of Merge Sort is O(n Log n) in all 3 cases (worst, average and best) as merge sort always divides the array in two halves and take linear time to merge two halves.

Algorithm	Worst Case	Average Case	Best Case
Merge Sort	O(n logn)	O(n logn)	O(n logn)

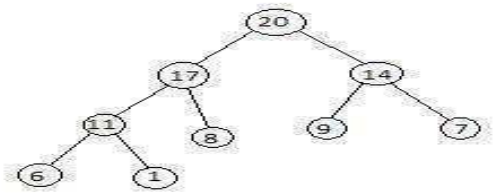
6.Heap Sort

Heap Sort is one of the best sorting methods being in -place and with no quadratic worst-case scenarios. Heap sort algorithm is divided into two basic parts
Creating a Heap of the unsorted list.
Then a sorted array is created by repeatedly removing the largest/smallest element from the heap, and inserting it into the array. The heap is reconstructed after each removal.
What is a Heap?
Heap is a special tree-based data structure that satisfies the following special heap properties **Shape Property**: Heap data structure is always a Complete Binary Tree, which means all levels of the tree are fully filled.
Heap Property: All nodes are either greater than or equal to or less than or equal to each of its children. If the parent nodes are greater than their children, heap is called a Max-Heap, and if the parent nodes are smaller than their child nodes, heap is called Min-Heap.



Min-Heap

In min-heap, first element is the smallest. So when we want to sort a list in ascending order, we create a Min-heap from that list, and picks the first element, as it is the smallest, then we repeat the process with remaining elements.



Max-Heap

In max-heap, the first element is the largest, hence it is used when we need to sort a list in descending order.

Heap Sort Algorithm

Step 1 − Create a new node at the end of heap.

Step 2 − Assign new value to the node.

Step 3 − Compare the value of this child node with its parent.

Step 4 − If value of parent is less than child, then swap them.

Step 5 − Repeat step 3 & 4 until Heap property holds.

Complexity of Heap Sort Algorithm

The heap sort algorithm is applied to an array A with n elements. The algorithm has two phases, and we analyze the complexity of each phae separately.

Phase 1. Suppose H is a heap. The number of comparisons to find the appropriate place of a new element item in H cannot exceed the depth of H. Since H is complete tree, its depth is bounded by $\log_2 m$ where m is the number of elements in H. Accordingly, the total number g(n) of comparisons to insert the n elements of A into H is bounded as

$$g(n) \leq n \log_2 n$$

Phase 2. If H is a complete tree with m elements, the left and right subtrees of H are heaps and L is the root of H Reheaping uses 4 comparisons to move the node L one step down the tree H. Since the depth cannot exceeds $\log_2 m$, it uses $4\log_2 m$ comparisons to find the appropriate place of L in the tree H.

$$h(n) \leq 4n\log_2 n$$

Thus each phase requires time proportional to $n\log_2 n$, the running time to sort n elements array A would be $n\log_2 n$

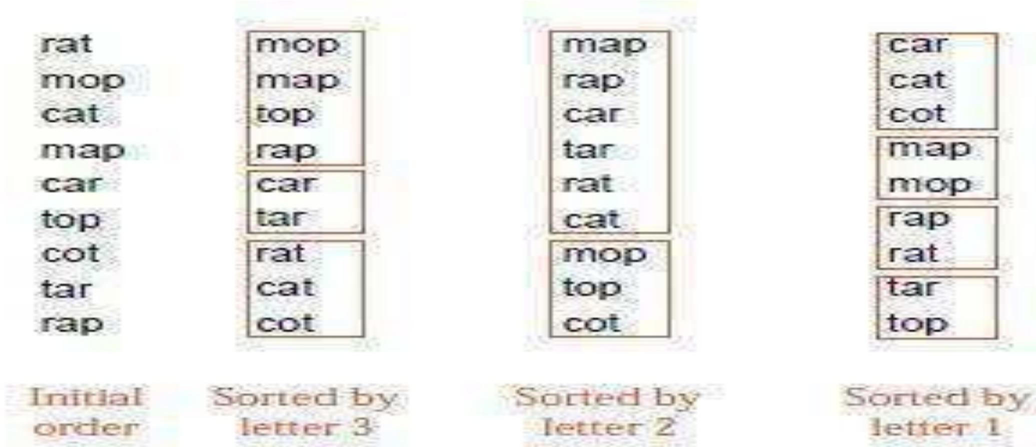
Algorithm	Worst Case	Average Case	Best Case
Heap Sort	O(n logn)	O(n logn)	O(n logn)

7. Radix Sort

The idea is to consider the key one character at a time and to divide the entries, not into two sub lists, but into as many sub lists as there are possibilities for the given character from the key. If our keys, for example, are words or other alphabetic strings, then we divide the list into 26 sub lists at each stage. That is, we set up a table of 26 lists and distribute the entries into the lists according to one of the characters in the key.

How Radix Sort Works

A person sorting words by this method might first distribute the words into 26 lists according to the initial letter (or distribute punched cards into 12 piles), then divide each of these sub lists into further sub lists according to the second letter, and so on. The following idea eliminates this multiplicity of sub lists: Partition the items into the table of sub lists first by the least significant position, not the most significant. After this first partition, the sub lists from the table are put back together as a single list, in the order given by the character in the least significant position. The list is then partitioned into the table according to the second least significant position and recombined as one list. When, after repetition of these steps, the list has been partitioned by the most significant place and recombined, it will be completely sorted. This process is illustrated by sorting the list of nine three-letter words below.



Radix Sort Algorithm

```
Algorithm: Radix-Sort (list, n)
shift = 1
for loop = 1 to keysize do
  for entry = 1 to n do
    bucketnumber = (list[entry].key / shift) mod 10
    append (bucket[bucketnumber], list[entry])
  list = combinebuckets()
  shift = shift * 10
```

Complexity of Radix Sort Algorithm

The list A of n elements A_1, A_2,……………A_n is given. Let d denote the radix(e.g d=10 for decimal digits, d=26 for letters and d=2 for bits) and each item A_i is represented by means of s of the digits:

$$A_i = d_{i1}\, d_{i2} \ldots\ldots\ldots\ldots\, d_{is}$$

The radix sort require s passes, the number of digits in each item . Pass K will compare each d_{ik} with each of the d digits. Hence

$$C(n) \leq d*s*n$$

Algorithm	Worst Case	Average Case	Best Case
Radix Sort	$O(n^2)$	d*s*n	O(n logn)

RGPVNOTES.IN

We hope you find these notes useful.

You can get previous year question papers at https://qp.rgpvnotes.in .

If you have any queries or you want to submit your study notes please write us at rgpvnotes.in@gmail.com

LIKE & FOLLOW US ON FACEBOOK
facebook.com/rgpvnotes.in