12/8/24
C-42

DATE:

* Array :-

i) An array is a group of similar elements of data items of the same type collected at contiguous memory locations.

ii) It indexing starts from 0.

iii) Array can store only the same type of data items.

iv) Array is a linear data structure where all elements are arranged sequentially.

v) Data items stored in an array are known as elements.

vi) Location of next index depends on the data type we use.

Representation of an Array :-

Array can be represented in several ways, depending on the different languages.

Type ↑ Name ↑ Elements

```
int array [10] = {35, 33, 42, 10, 14, 19, 27, 44, 26, 31}
        ↓
       Size
```

Contiguous Memory locations

200	204	208	212	216	220	224	228	232	234
35	33	42	10	14	19	27	44	26	31
0	1	2	3	4	5	6	7	8	9

Index

DATE:

Initialization of an Array :-

If an array is described inside a function, the elements will have garbage value. And in case an array is static or global, its elements will be initialized automatically to 0.

Syntax :-

datatype Array-Name[size] = {value1, value2, value3, ... valueN};

Types of Arrays :-

i) one-Dimensional Arrays :-

A one-dimensional array is a kind of linear array. It involves single sub-scripting. The [] (brackets) is used for the subscript of the array and to declare and access the elements from the array.

Syntax :-

DataType ArrayName[size];

eg :- int a[10];

ii) Two-Dimensional Arrays :-

An array involving two subscripts [][] is known as two-dimensional arrays. They are also known as the array of the array. Two-dimensional arrays are divided into rows and columns and are able to handle the data of the table.

DATE:

Syntax :-

DataType ArrayName[row size][column size];

eg :- int a[5][5];

iii) Three - Dimensional Arrays :-

When we require to create two or more tables of the elements to declare the array elements, then in such a situation we use three-dimensional Arrays.

Syntax :-

DataType ArrayName[size1][size2][size3];

eg :- int a[5][5][5];

Advantages of Array :-

i) It is a better version of storing the data of the same size and same type.

ii) It enables us to collect the number of elements in it.

iii) Arrays have a safer cache positioning that improves performance.

iv) Arrays can represent multiple data items of the same type using a single name.

DATE:

Disadvantages of Array :-

i) In an array, it is essential to identify the number of elements to be stored.

ii) It is a static structure. It means that in an array, the memory size is fixed.

iii) When it comes to insertion and deletion, it is a bit difficult because the elements are stored sequentially and the shifting operation is expensive.

* Sorting :-

Sorting means reording of all the elements either in ascending or in descending order.

Types of Sorting :-

1. Bubble Sort Algorithm :-

Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in the wrong order. This algorithm is not suitable for large data sets as its average and worst-case time complexity is quite high.

DATE: _____

Working of Bubble Sort :-

5 4 3 2 1

i) First Pass :-

5 4 3 2 1
4 5 3 2 1
4 3 5 2 1
4 3 2 5 1
4 3 2 1 5 → Sorted
↳ unsorted

ii) Second Pass :-

4 3 2 1 5
3 4 2 1 5
3 2 4 1 5
3 2 1 4 5
↳ unsorted ↳ Sorted

iii) Third Pass :-

3 2 1 4 5
2 3 1 4 5
2 1 3 4 5
↳ unsorted ↳ Sorted

iv) Fourth Pass :-

2 1 3 4 5
1 2 3 4 5
↓
Sorted

Q 13 32, 26, 35 10

i) First Pass :-

13 32 26 35 10
13 32 26 35 10
13 26 32 35 10
13 26 32 35 10
13 26 32 10 35
unsorted ↳ Sorted

ii) Second Pass :-

13 26 32 10 35
13 26 32 10 35
13 26 10 32 35
↳ unsorted ↳ Sorted

iii) Third Pass :-

13 26 10 32 35
13 26 10 32 35
13 10 26 32 35
↳ unsorted ↳ Sorted

DATE:

iv) Fourth Pass :-

13 10 26 32 35
10 13 26 32 35
Sorted

Time Complexity of Bubble Sort :-

Best Case $O(n)$
Average Case $O(n^2)$
Worst Case $O(n^2)$

16/8/24

2. Selection Sort :-

Selection Sort is a simple and efficient sorting algorithm that works by repeatedly selecting the smallest or largest element from the unsorted portion of the list and ~~moving~~ this process is repeated for the remaining unsorted portion until the entire list is sorted.

Working of Selection Sort :-

5 4 3 2 1

i) First Pass :-

Max Min
5 4 3 2 1
5 4 3 2 1
5 4 3 2 1
5 4 3 2 1
Unsorted → 1 Sorted

DATE:

ii) Second Pass :-

Max Min

1 4 3 2 5
1 4 3 2 5
1 4 3 2 5
1 4 3 2 5
1 2 3 4 5 → Sorted

unsorted | ↳ Sorted

~~iii) Third Pass :-~~

~~1 2 3 4 5~~

Time Complexity of Selection Sort :-

Best Case $O(n^2)$
Average Case $O(n^2)$
Worst Case $O(n^2)$

Q 22 24 7 9 13 18 15

→ Max Min
22 24 7 9 13 18 15

Max Min
7 24 22 9 13 18 15

Max Min
7 9 22 24 13 18 15

Max Min
7 9 13 24 22 18 15

Max Min
7 9 13 15 22 18 24

DATE:

4 9 13 15 18 22 24 → Sorted.

22/8/24

3. Insertion Sort :-

Insertion Sort is a simple sorting algorithm that works by iteratively inserting each element of an unsorted list into its correct position in a sorted portion of the list. It is a stable sorting algorithm, meaning that elements with equal values maintain their relative order in the sorted output.

Working of Insertion Sort :-

5 4 3 2 1

i) First Pass :-

5	4	3	2	1
4	5	3	2	1
4	3	5	2	1
4	3	2	5	1
4	3	2	1	5

unsorted | Sorted

5	4	3	2	1
4	5	3	2	1

Sorted | Unsorted

ii) Second Pass :-

4	3	2	1	5
3	4	2	1	5
3	2	4	1	5
3	2	1	4	5

4	5	3	2	1
4	3	5	2	1
3	4	5	2	1

Sorted | Unsorted

DATE:

iii) Third Pass :—

3 4 5 2 1
3 4 5 2 1
3 4 2 5 1
3 2 4 5 1
2 3 4 5 1

Sorted | Unsorted

iv) Fourth Pass :—

2 3 4 5 1
2 3 4 1 5
2 3 1 4 5
2 1 3 4 5
1 2 3 4 5

Sorted

Q 1 2 3 4 5

→ 1 / 2 3 4 5

1 2 / 3 4 5

1 2 3 / 4 5

1 2 3 4 / 5

1 2 3 4 5

Sorted

DATE:

Q 62 68 42 47 38 71 72 81

> ~~Final Path :~~

62 68 42 47 38 71 72 81

62 / 68 42 47 38 71 72 81

62 68 | 42 47 38 71 72 81

62 42 68 / 47 38 71 72 81

42 62 68 / 47 38 71 72 81

42 62 47 68 / 38 71 72 81

42 47 62 68 / 38 71 72 81

42 47 62 68 38 / 71 72 81

42 47 62 38 68 / 71 72 81

42 47 38 62 68 | 71 72 81

42 38 47 62 68 | 71 72 81

38 42 47 62 68 / 71 72 81

38 42 47 62 68 71 72 81

Solved

DATE:

Time complexity of Insertion Sort :-

Best Case	$O(n)$
Average Case	$O(n^2)$
Worst Case	$O(n^2)$

23/8/24

* Asymptotic Notations :-

The notation we used to describe asymptotic running time algorithm are defined in terms of functions :-

i) Big oh Notation (O)
ii) Sigma Notation (Σ)
iii) Theta Notation (Θ)
iv) Small oh Notation (o)
v) omega Notation (ω)

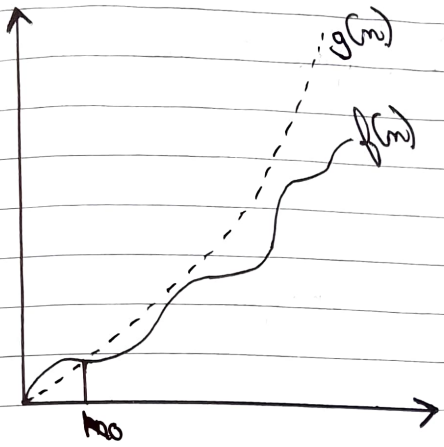
i) Big oh Notation (O) :-

It is the formal way to express the upper boundary of an algorithm running time. It measures the worst case of time complexity or the longest amount of time, algorithm takes to complete their operation.

$f(n)$ is $O(g(n))$, if there exist positive constant c and n_0 such that :-

DATE: _____

$F(n) \leq cg(n)$ $\forall$ $n \geq n_0$.



Q $f(n) = 2n^2 + 5n + 6$.

Soln $f(n) \leq C \cdot g(n)$

$2n^2 + 5n + 6 \leq C \cdot n^2$

$2(4)^2 + 5 \times 4 + 6 \leq 4 \times 4^2$

$2 \times 16 + 20 + 6 \leq 4 \times 16$

$32 + 20 + 6 \leq 64$

$58 \leq 64$

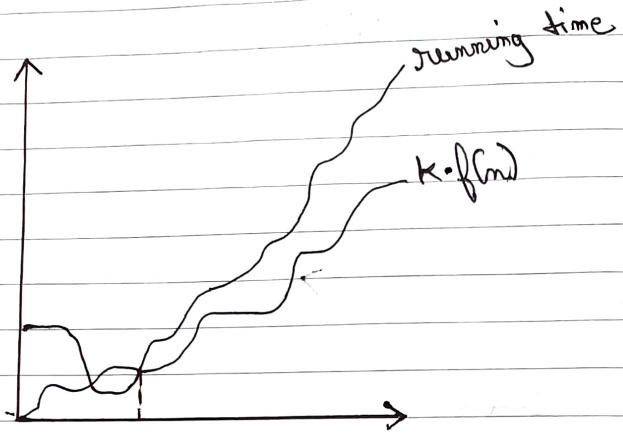
ii) Sigma Notation (Ω) :-

It is the formal way to represent the lower bound of an algorithm's running time. It measures the best amount of time an algorithm can possible take to complete or the best case time complexity.

DATE:

$f(n)$ is $\Omega(g(n))$, if there exist a positive constant C and no. such that :-

$$0 \leq C \cdot g(n) \leq f(n) \quad \forall \quad n \geq n_0.$$



Q $f(n) = 2n^2 + 5n + 6$

Soln

$$0 \leq C \cdot g(n) \leq f(n)$$
$$0 \leq C \cdot n^2 \leq 2n^2 + 5n + 6$$
$$0 \leq 3 \cdot (4)^2 \leq 2(4)^2 + 5 \times 4 + 6$$
$$0 \leq 3 \cdot 16 \leq 2 \cdot 16 + 20 + 6$$
$$0 \leq 48 \leq 32 + 20 + 6$$
$$0 \leq 48 \leq 58$$

29/8/24

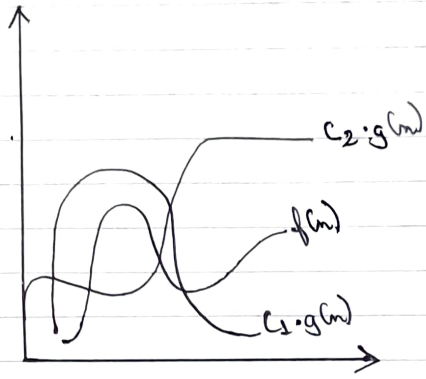
iii) Theta Notation (θ) :-

DATE:

Θ - notations (Asymptotically tightly bound)

$f(n) = \Theta(g(n))$ if there exist a positive constant, c_1 and c_2, n_0. such that :-

$$0 \leq c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n) \quad \forall\, n \geq n_0.$$



iv) Small oh Notation (o) :-

Small oh Notation (o) is a upper bound but not asymptotically tight.

$f(n) = o(g(n))$, if there exist positive constant c, n_0. such that :-

$$0 \leq f(n) < c \cdot g(n) \quad \forall\, n \geq n_0.$$

DATE: _____

iv) Omega Notation (ω) :-

Omega notation (ω) is a lower bound but not asymptotically tight.

$f(n) = \omega(g(n))$

$0 \leq c \cdot g(n) < f(n) \quad \forall \; n \geq n_0$.

* $2n^2 + 5n + 200$
$\omega(n^1), \omega(n^0) \; ---- \infty$

★ $2n^2 + 5n + 200$:-

$o(n^2), o(n^3), o(n^4) \; --- \infty$
$\Omega(n^2), \Omega(n^1), \Omega(n^0) \; --- \infty$
$\Theta(n^2)$
$o(n^3), o(n^4) \; --- \infty$
$\omega(n), \omega(n^0) \; --- \; -\infty$

$O \cap \Omega = \Theta$
$O \cap \omega = \emptyset$

30/8/24
Class

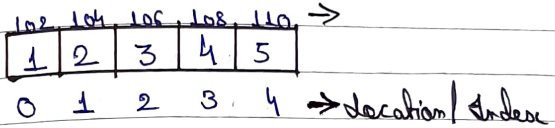
DATE:

* Array :-

Array is a collection of similar datatype.

Syntax :-

Datatype arrayname[Size] = {value}.
int a[5] = {1, 2, 3, 4, 5}



a[0] = 1
a[1] = 2
a[2] = 3
a[3] = 4
a[4] = 5

a = 102

~~* Pointer :-~~

Single Dimensional Array :-

Single dimensional array as a row where elements are stored one after another.

Multi- Dimensional Array :-

It is an array that has more than one dimensional.

DATE:

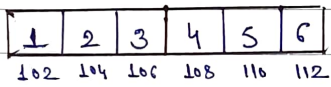
Syntax of 2D Array :-

datatype arrayname[row][column] = {values};

int a[2][3] = {1, 2, 3, 4, 5, 6};

$$a = \begin{matrix} R_1 \\ R_2 \end{matrix} \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ C_1 & C_2 & C_3 \end{bmatrix} = \{\{1,2,3\}, \{4,5,6\}\};$$

a[0][0] = 1 a[1][0] = 4
a[0][1] = 2 a[1][1] = 5
a[0][2] = 3 a[1][2] = 6



a = 102
**a;

5/9/24

* Parameters :-

Actual Parameters :-

The parameter passed to a function.

Formal Parameters :-

The parameter received by a function.

DATE:

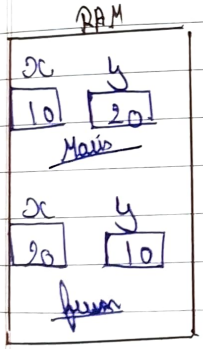
Actual Parameter.

```
int a;
a = add(m, n)

int add(int a, int b)
{                    → Formal Parameter.
    return(a+b);
}
```

Call by Value :-

Here value of actual parameters will be copied to formal parameters - and these two different parameters store values in different location.




```
main()
{
    int x=10, y=20;
    fun(x, y);
    printf("%d%d", x, y);
}
```

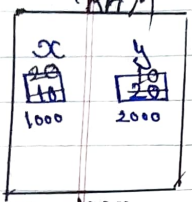
```
int fun(int x, int y)
{
    x = 20;
    y = 10;
    printf("%d%d", x, y);
}
```

Call by Reference :-

Here both actual and formal parameters refers to same memory location. Therefore, any changes made to the formal parameters will get reflected to the actual parameters.

DATE:

RAM



&x = 1000
&y = 2000

```
main
{
    int x = 10, y = 20;
    fun(&x, &y);
}  printf(x, y)
```

20, 10

```
int fun(int *Ptr1, int *Ptr2)
{
    *Ptr1 = 20;
    *Ptr2 = 10;
}  Printf(x, y);
```

Ptr1 = 1000
Ptr2 = 2000

20, 10