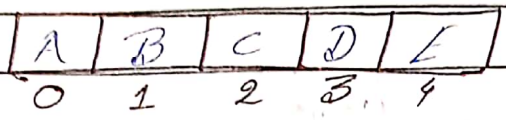
DATE: __/__/____
PAGE: _____

Array

→ It is used to store same type of data elements in the continuous manner. "Name of the array is the Base Address."



Advantages

① Random Access ⇒ Get i^{th} element in O(1) Time because we have address of the first element.

② Cache Friendliness ⇒ Cache is fastest and very close to the CPU. The cache friendliness refers to that to access the nearby set of element if you access element of array that means, if you access element A, then B, C, D also get fetched.

Array Types

① Static / Fixed Sized ⇒ Size of Array is fixed

Eg:- ① int arr[100] ② int arr[n] ③ int arr[] = {10, 20, 30, 40};

Stack Allocation

② int *arr = new int[n]

Heap Allocation.

Good Write

DATE: __/__/____
PAGE: _____

(2) Dynamic Sized Array ⇒ Array size is not fixed, it can be changed

→ Vector is used to implement the Dynamic Sized Array.

(*) Resize Automatically.

(##) Operation in Array

(###) Searching in Array

(*) For Sorted Array searching is done by Binary Search.

(**) For Unsorted Array searching is done by Linear Search.

Linear Search

```
int Search (int arr[], int n, int k)
{
    for (int i = 0; i < n; i++)
        if (arr[i] == k)
            return i;
    return -1;
}
```

Time Complexity ⇒ O(n)

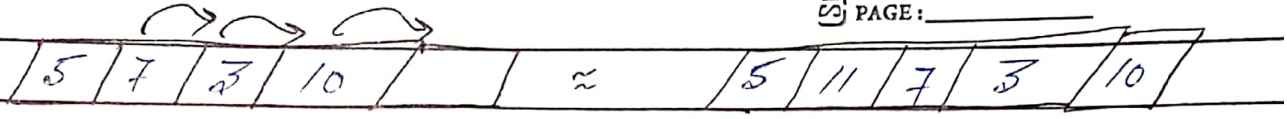
Inserting in Array

→ We can't insert if array size is fixed and it is already full.

(*) For that we need to check whether is array is full or not.

Good Write

DATE: __/__/____
PAGE: _____



pos = 2, x = 11, idx = 1.

```
int insert (int arr[], int n, int x, int cap, int pos)
{
    if(n == cap)
        return n;

    int idx = pos - 1

    for(int i = n-1; i >= idx; i--)

        arr[i+1] = arr[i];

        arr[idx] = x;

        return(n+1);
}
```

Time Complexity ⇒ O(n), At the End ⇒ O(1).

Deletion in Array

```
int delete (int arr[], int n, int x)
{
    int i;

    for(i=0; i<n; i++)

        if(arr[i] == x)  // Searching for 1st occurance
            break;
```

Good Write

DATE : __/__/____
PAGE : _____

```
if(s == n)   // Element Not found
    return n;
for(int j = s; j < n-1; j++) // If element present loop will Run
    arr[j] = arr[j+1];   // Shifting of elements
return (n-1);            // Return deleted array
}
```

(**) Summary of Operations

Insert O(n)

Search O(n) [Unsorted] O(log n) [Sorted]

Delete O(n)

Get i^{th} element O(1)

Update i^{th} element O(1).

(XX) Insert at end & Delete at end done in O(1) Time

(##) Largest Element in Array

```
int largest(int arr[], int n)
{
    int max = arr[0];
    for(s=1; s<n; s++)
```

Good Write

DATE: __/__/____
PAGE: _____

```
if(arr[i] > max)
    max = arr[i]
return max;
}
```

(**) Method for finding size of any array

```
int n = sizeof(arr) / sizeof(arr[0]);
```

(**) Problem : 1 ⇒ Reverse the Array / String

```
int reversearray(int arr[], int n)
{  void
   int start = 0; int end = n-1; int temp;
   while(arr[start] != arr[end])
   {  temp = arr[start];
      arr[start] = arr[end];
      arr[end] = temp;
      start++;
      end--;
   }
   return
```

(*) Swapping with the help of temp variable