Array

Definition:

An array can be defined mathematically as a set of (Index, Value) pair.

For example- An array A[5]= {(0,12),(1,32),(2,54),(3,67),(4,2)}

0	1	2	3	4
12	32	54	67	2

Array as a data structure-

Domain : Type of the data stored in the array.

Functions : Operations performed on the array like-

1) **Create()** – Create an array
2) **Insert()** – Insert a new element in the array.
3) **Delete()** – Delete an existing element from the array.
4) **Reverse()** – Alter the order of the elements in reverse fashion.
5) **Display()** – Display the elements of an array.
6) **Search()** – Search a specific element in an array.
7) **Sort()** – Arrange the element is ascending or descending order.
8) **Concate()** – Join two array one after another
9) **Merge()** – Join two sorted array in a such a fashion that the resultant array will also become sorted etc.

Axioms : The rules followed by the above functions like which input will be taken and what output will be returned. For example Insert() takes as input an existing array and add the new element to be inserted and outputs the array with the newly added element.

Algorithms –

Create()

Input : Take no input.

Output : Return the newly created array.

Procedure :

1. Begin
2. Allocate an Array ARR of size MAXSIZE.
3. Take the size of the array into n from user.
4. Initialize i to 0.
5. If i>=n then goto Step 9.
6. Take the value of i^{th} element from user into ARR[i].
7. Increment i by 1.
8. Goto Step 3
9. Return the array ARR of size n.

10. End of the process.

Insert()

Input : Take an array of size n and a new element t of same as the type of array.

Output : Return the array of size n+1.

Procedure :

1. Begin
2. Initialize i to n+1.
3. If i=MAXSIZE then goto Step 5 .
4. ARR[i] := t
5. Print “The Array is full”
6. Return the array ARR of size n+1.
7. End of the process.

Example-

12	32	54	67	2
0	1	2	3	4

If we add a new element 15 to the above array then the resultant array will be-

12	32	54	67	2	15
0	1	2	3	4	5

Delete()

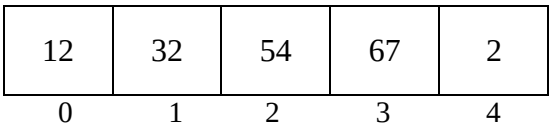
Input : Take an array of size MAXSIZE with n no. of element and the element d to be deleted.

Output : Return the array with n-1 no. of element.

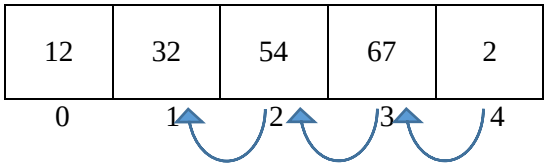
Procedure :

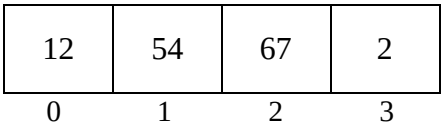
1. Begin
2. Initialize i to 0.
3. If i>=n then goto Step 13.
4. If ARR[i] = d then begin
5. Initialize j with i+1.
6. If j>=n then go to Step 10
7. ARR[j-1] :=ARR[j]
8. Increment j by 1
9. Goto Step 6
10. Print “The element” d ”is found at” i.
11. Increment i by 1.
12. Goto Step 3
13. Return the array ARR of size n-1.
14. End of the process.

Example-



If we want to delete 32 from the above array then the resultant array will be-





Display() : This is your exercise. Follow the Create procedure and try to write the algorithm.

Reverse()

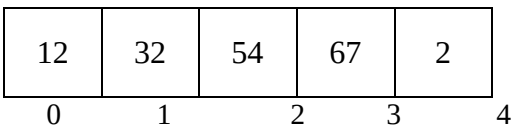
Input : An array of size n.

Output : The reversed array of size n.

Procedure :

1. Begin
2. Initialize i to 0.
3. If i>floor(n/2) then goto Step 8.
4. Temp :=ARR[i]
5. ARR[i] :=ARR[n-i-1]
6. ARR[n-i-1] :=Temp
7. Increment i by 1.
8. Goto Step 3
9. Return the array ARR of size n.
10. End of the process.

For example, consider the following Array , here n=5



Loop will start from 0 and ends in floor(5/2) =2 ,

ARR[0] and ARR[5-0-1] will be swapped

ARR[1] and ARR[5-1-1] will be swapped

ARR[2] and ARR[5-2-1] will be swapped

The resultant array will be,

2	67	54	32	12
0	1	2	3	4

Concate() : This is again your exercise. You have to do the following

A[]

12	32	54	67	2

B[]

5	76	20	17	21

C[]

12	32	54	67	2	5	76	20	17	21

Merge()

Input : Take two sorted array A[] of size n and B[] of size m.

Output : Return the merged array M[] of size (n+m)

Procedure :

1. Begin
2. Initialize i to 0 and j to 0 to the array A and B respectively
3. Initialize k to 0 as the index of array C
4. If i>=n or j>=m then goto Step 15
5. If A[i] <= B[j] then
6. C[k] := A[i] // the element of array A is lower than element of B so it will go to C
7. Index i will be incremented by 1. // As element of A goes to C so i is incremented.
8. Increment k by 1.
9. Goto Step 4
10. Else then
11. C[k] := B[j] // the element of array B is lower than element of A so it will go to C
12. Index j will be incremented by 1. // As element of B goes to C so j is incremented.
13. Increment k by 1.
14. Goto Step 4
15. If j>=m then goto Step 20 // Pushing to Case I
16. C[k] := B[j] // Case II
17. Increment j by 1.
18. Incement k by 1.
19. Goto Step 15.
20. If i>=n then goto Step 25
21. C[k] := A[i] // Case I
22. Increment i by 1.
23. Incement k by 1.
24. Goto Step 20.
25. Return the array C of size (n+m)
26. End the procedure

Case I :

A[]

12	32	54	67	91

B[]

5	16	20	37	85

M []

5	12	16	20	32	37	54	67	85	91

B will end before A

Case II:

A[]

12	32	54	67	80

B[]

5	16	20	37	85

M []

5	12	16	20	32	37	54	67	80	85

A will end before B

Searching

Searching in an array is of two types –

1) Linear Search
2) Binary Search

Linear Search :

Searching for a element from beginning to end of an array. It requires less than n no of comparisons (from 0 to n-1) if the search is successful or in case of unsuccessful search requires n no. of comparisons. Assuming the size of the array is n.

Input : An array of size n ARR[] and the element to be searched k.

Output : The position of the element in the array.

Procedure :

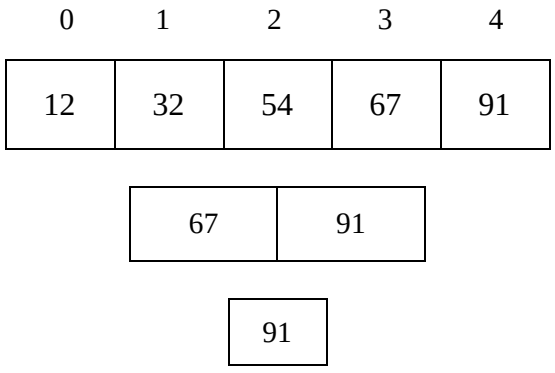
1. Begin
2. Initialize the index i to 0
3. If i>=n then goto Step 7.
4. If ARR[i] = k then return the position i
5. Increment i by 1.
6. Goto Step 3
7. Print “The element not found”
8. End of the procedure.

** Note : Upgrade the above algo to find out the frequency of k in ARR[].

Binary Search :

This approach is called Divide and conquer approach. Assume that we have a sorted array then we divide the array into two part and the element at the middle is compared with k, if found then okay. Otherwise two possibilities are there, first k is lesser than mid then it may be in the left part of the array, second k is greater than mid then it may be at the right part. One possibility will happen and the same process will be repeated for that part.

Lets see the pictorial representation,



Let us assume the key element is 91.

Here, beg = 0 and end = 4
Then first time mid = 2 but not matched and k is greater than item at mid
Hence, second time beg = 3 and end = 4 and mid = 3 again not matched and k is greater than item at mid
Then third time beg= 4 and end=4 and mid =4 and item matched and found at 4.
Input : A sorted array ARR[] with beg as lower index and end as upper index and the item to be searched i.e. k.

Output : The position of the item in the array.

Procedure :

Try to write it else I will help you.

Advantage : Advantage of Binary search over linear search, it require significant amount of lesser comparisons than linear search. As this divide and conquer process create a tree like structure the amount of comparisons comes to $\log_2 n$ which is significant betterment. But the limitation is requirement of sorted array.