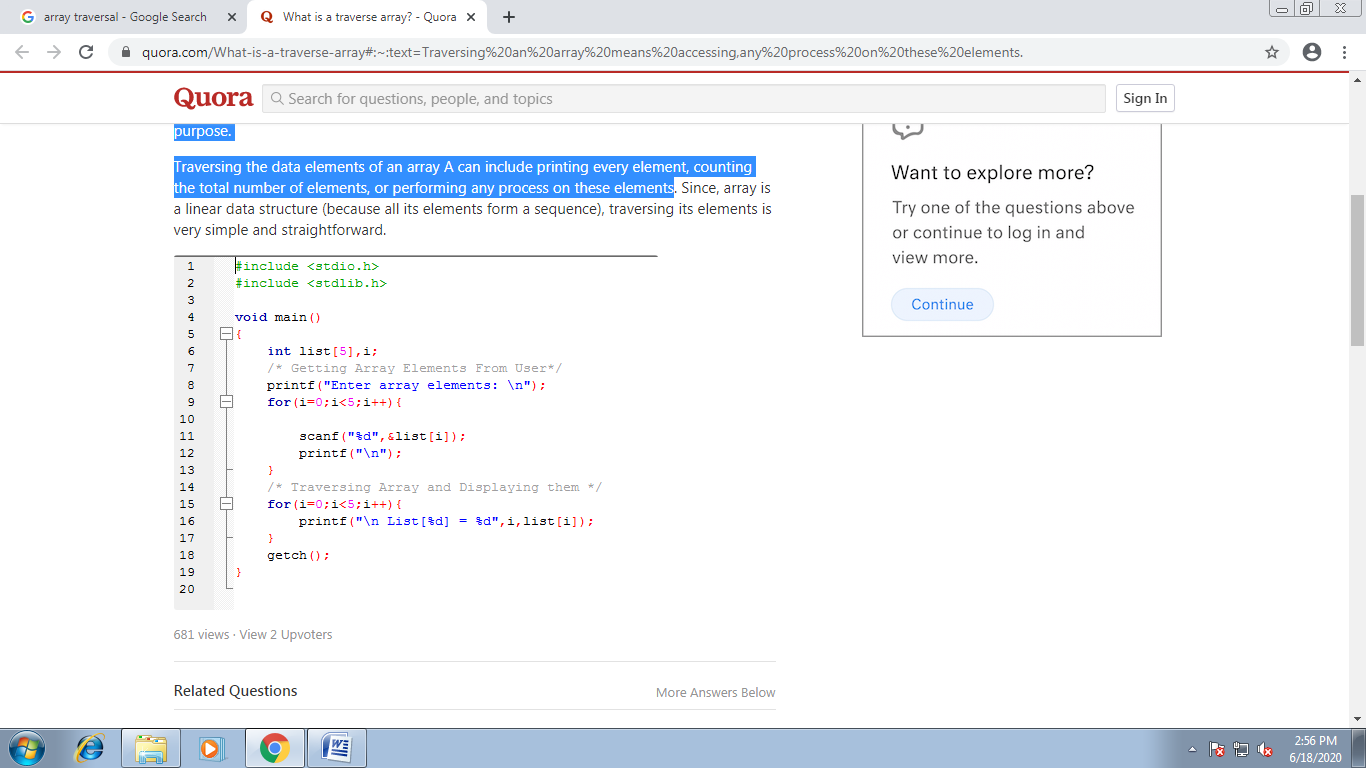
**Basic Operations of Data Structure**

The data in the data structures are processed by certain operations. The particular data structure chosen largely depends on the frequency of the operation that needs to be performed on the data structure.

1. **Traversing-** Traversing an array means accessing each and every element of the array for a specific purpose.

Traversing the data elements of an array A can include printing every element, counting the total number of elements, or performing any process on these elements



**2. Searching –**

* Searching is the process of finding a given value position in a list of values.
* It decides whether a search key is present in the data or not.
* It is the algorithmic process of finding a particular item in a collection of items

Types of searching-

Linear/Sequential Search

Binary Search

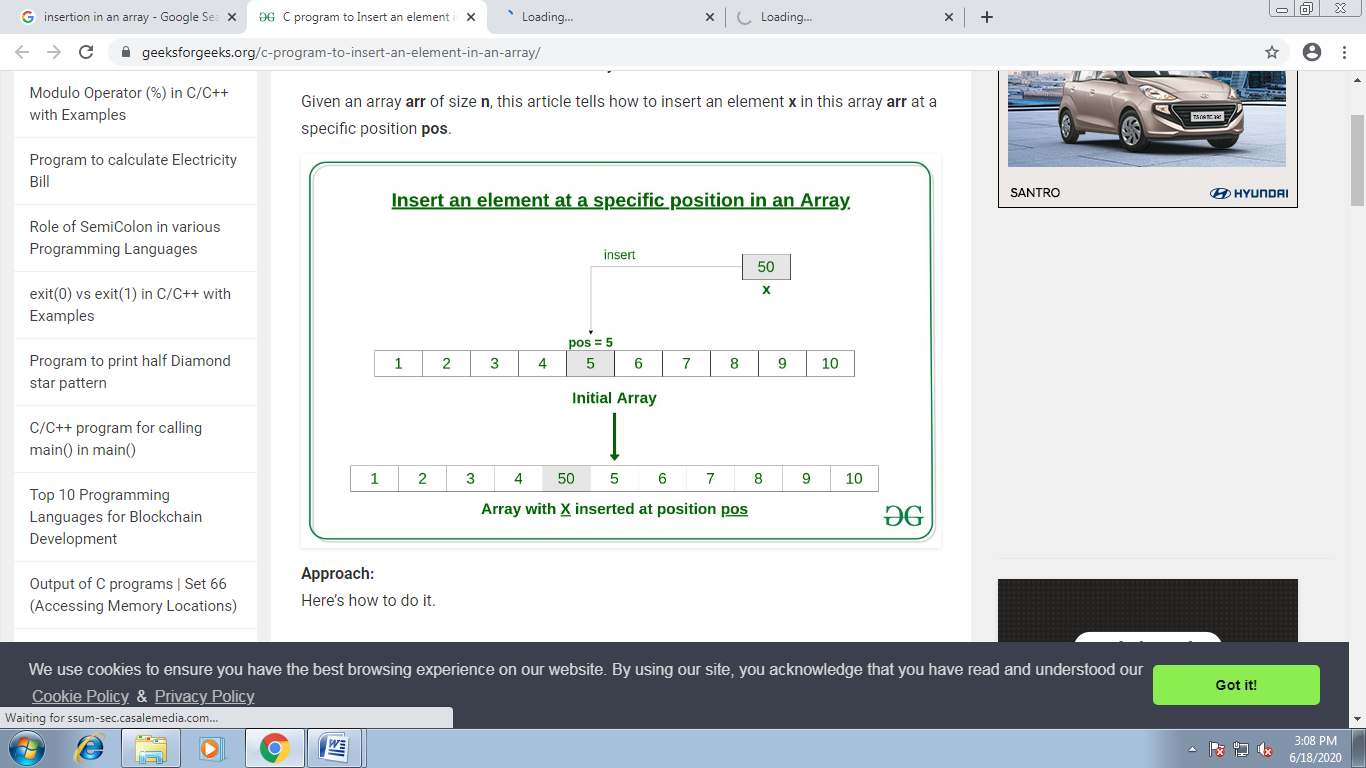
Hashing

Linear/Sequential Search

If you want to find the position in an unsorted array of an integers that stores a particular value, you cannot really do better than simply looking through the array from the beginning and move toward the end until you find what you are looking for. This algorithm is called [*sequential search*](https://opendsa-server.cs.vt.edu/ODSA/Books/Everything/html/Glossary.html#term-sequential-search). If you do find it, we call this a [*successful search*](https://opendsa-server.cs.vt.edu/ODSA/Books/Everything/html/Glossary.html#term-successful-search). If the value is not in the array, eventually you will reach the end. We will call this an [*unsuccessful search*](https://opendsa-server.cs.vt.edu/ODSA/Books/Everything/html/Glossary.html#term-unsuccessful-search).

**3. Insertion**

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.



Here, we see a practical implementation of insertion operation, where we add data at the end of the array –

#include <stdio.h>

main() {

int LA[] = {1,3,5,7,8};

int item = 10, k = 3, n = 5;

int i = 0, j = n;

printf("The original array elements are :\n");

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

printf("LA[%d] = %d \n", i, LA[i]);

}

n = n + 1;

while( j >= k) {

LA[j+1] = LA[j];

j = j - 1;

}

LA[k] = item;

printf("The array elements after insertion :\n");

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

printf("LA[%d] = %d \n", i, LA[i]);

}

}

**Program to insert a number in its specific position in a sorted array.**

#include<stdio.h>

#include<conio.h>

int insert(int A[], int N, int K, int item);

void main()

{

int A[10]={10,12,15,20,58,65,08,12};

int I, item, loc, size=8;

printf(“\n\t enter number to be inserted”);

scanf(“%d”,&item);

printf(“\n\t enter the position”);

scanf(“%d”,&loc);

if(loc<=size)

size= insert(A, size, loc, item);

else

{

printf(“\n\t location is out of bound”);

getch();

}

printf(“new array\n”);

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

printf(“%d”, A[i]);

getch();

}

int insert(int A[], int N, int K, int item)

{

int J=N;

while(J>=K-1)

{

A[J+1]=A[J];

J--;

}

A[k-1]=item;

return(N+1);

}

**4. Deletion**

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 Kth 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

#include <stdio.h>

void main() {

int LA[] = {1,3,5,7,8};

int k = 3, n = 5;

int i, j;

printf("The original array elements are :\n");

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

printf("LA[%d] = %d \n", i, LA[i]);

}

j = k;

while( j < n) {

LA[j-1] = LA[j];

j = j + 1;

}

n = n -1;

printf("The array elements after deletion :\n");

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

printf("LA[%d] = %d \n", i, LA[i]);

}

}

**Program to delete a number from an array.**

#include<stdio.h>

#include<conio.h>

int delete(int A[], int N, int K);

void main()

{

int A[10]={10,20,11,12,15,18,52,68};

int i, loc, size=8;

printf(“\n\t enter the location from where you want to delete the number”);

scanf(“%d”,&loc);

if(loc<=size)

size= delete(A, size, loc);

else

{

printf(“\n\t location is out of bound”);

getch();

exit();

}

printf(“new array\n”);

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

printf(“%d”, A[i]);

getch();

}

int delete(int A[], int N, int K)

{

int J,item;

item=A[K-1];

printf(“deleted element is %d”, item);

for(J=K-1; J<N-1;J++)

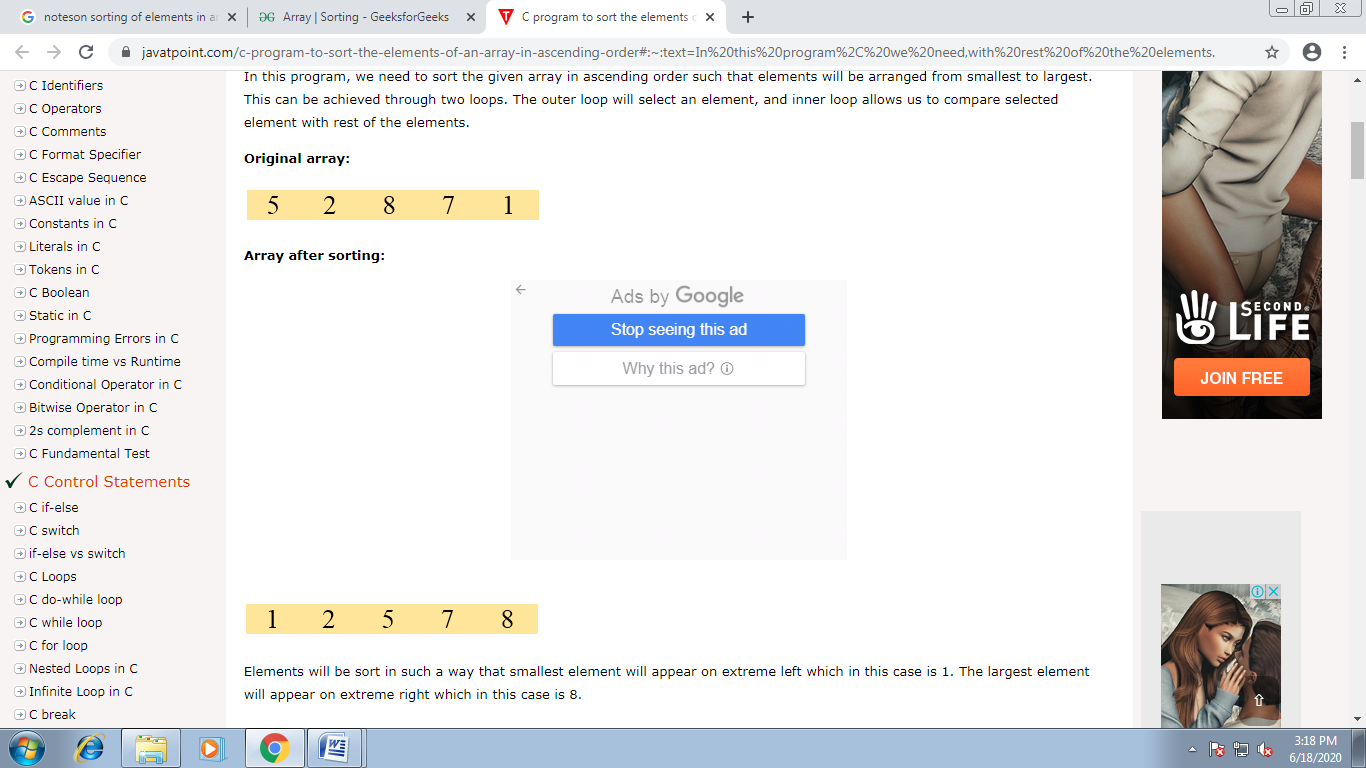
A[J]=a[J+1];

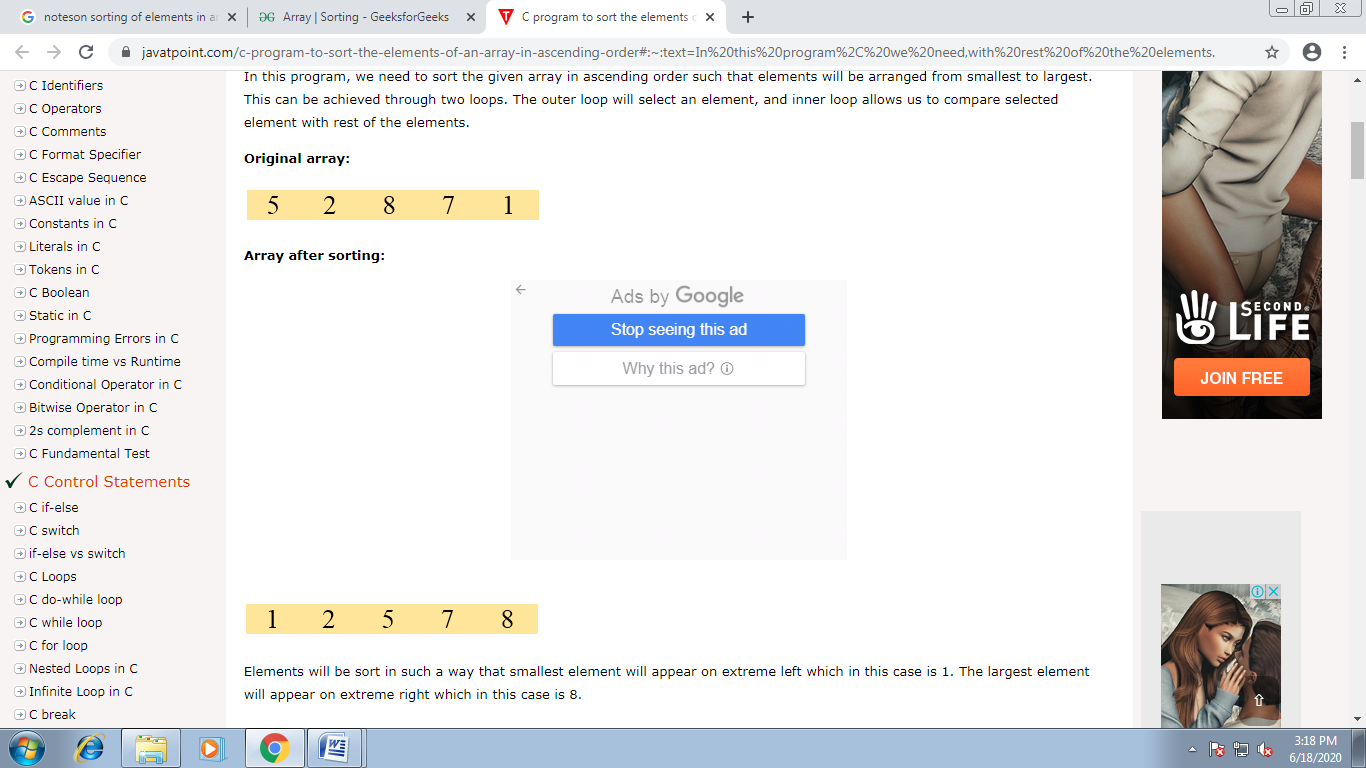
return(N-1);

}

**5. Sorting-**

Sorting is nothing but the arrangement of elements in an order.(Either in increasing or Decreasing).





Different Types of Sorting are-

Bubble Sort

Selection Sort

Insertion Sort

Quick Sort

Merge Sort

Heap Sort

Radix Sort

**6. Merging-** It is the process of combining two arrays.

A. Read the 1st array size and store it into the variable n1. Read the entered elements and store the elements into the 1st array a[] using scanf() statement, the for loop for(i=0;i<n1;i++).

B. Read the 2nd array size and store the value into the variable n2. Read the entered elements into the 2nd array b[] using scanf() statement, the for loop for(i=0;i<n2;i++).

C. Merge the two arrays a[],b[] into c[] as,

for loop iterates from i=0 to i<size of the 1st array+size of 2nd array

a) if i<size of 1st array then c[i]=a[i].Otherwise c[i]=b[i-n1].

b) After all iterations of for loop the two arraays will be merged into the array c[] and the size of c is size of 1st array+size of 2nd array.

D. Print the first array by calling print(a,n1) and print 2nd array by calling print(b,n2) and print the c[] by calling print(c,n1+n2).

