**Insertion Sort**

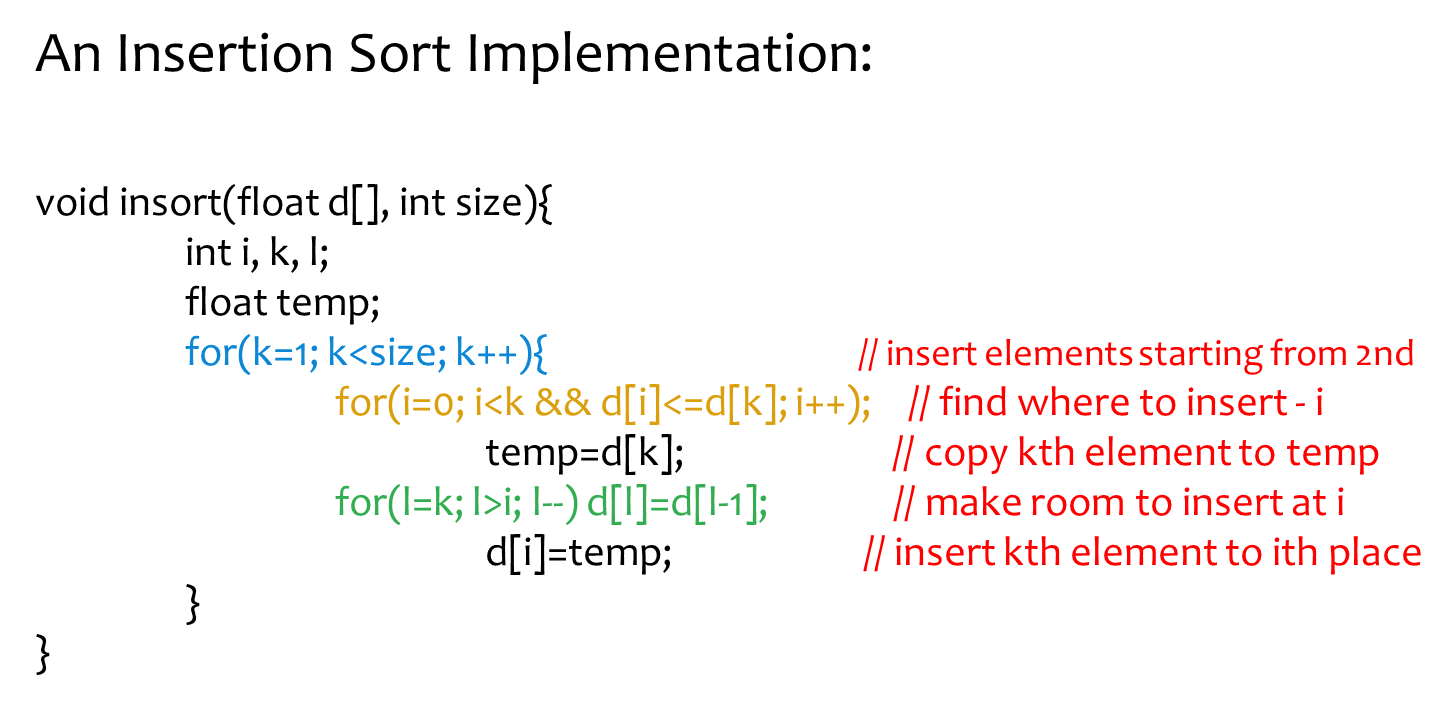
At any given time, we recognize an element in the given array with index k, and the array elements up to k-1 is sorted and the elements from k to n-1 are yet to be sorted.

K can be initialized to 1, indicating a single element sorted array portion and an n-1 element unsorted array portion, at start.

In the execution of the algorithm, at each step, we take the k th element of the array and insert into the right position within the sorted part of the array, and increment k thereby maintaining the invariant.

If the k th item need to be inserted into i th position, within the sorted array portion, then we need to push all the items from i th to (k-1)th element, one position rightwards, to make room to insert the k th item at i th position.

In each step, k will be incremented and in the last step, k will be n-1 and the last element of the array will be inserted to the right position in the sorted array portion, completing the sort operation.



Apply the above algorithm on the below array:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 23 | 78 | 45 | 8 | 32 | 56 |

**Merge Sort Implementation using recursion:**

void msort(int d[],int size){

int i,j,k, result[size], mid=size/2;

if(size==1) return; // nothing to do, already sorted array

msort(d,mid); // sort first half of d[] by a recursive call

msort(&d[mid],size-mid); // sort 2nd half of d[] by a recursive call

for(k=0,i=0,j=mid; i<mid && j<size; k++) // merge as long as i and j are

result[k]=d[i]<d[j]?d[i++]:d[j++]; //within the respective subarrays

while(i<mid) result[k++]=d[i++]; // append if any residual of 1st half

while(j<size) result[k++]=d[j++]; // append any residual of 2nd half

for(k=0;k<size;k++) d[k]=result[k]; // copy result to d[]

}

Discussion - Can you visualize how the above array given in Figure (below)will be sorted by the msort() function above? Mark each arrow in Figure with a sequence number to illustrate the order of processing.

