```c
// C program for implementation of Normal Bubble sort
#include <stdio.h>

void swap(int *xp, int *yp)
{
    int temp = *xp;
    *xp = *yp;
    *yp = temp;
}

// A function to implement bubble sort
void bubbleSort(int arr[], int n)
{
    int i, j;
    for (i = 0; i < n-1; i++) // loop for the number of trips
    {
        for (j = 0; j < n-i-1; j++) // loop for the number of comparisons per trip
        {
            if (arr[j] > arr[j+1]) //swap if the previous element is greater than the next one(for ascending order)
            {
                swap(&arr[j], &arr[j+1]);
            }
        }

    }
}

/* Function to print an array */
void printArray(int arr[], int size)
{
    int i;
    for (i=0; i < size; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

```

```c
// Driver program to test above functions
int main()
{
    int arr[] = {64, 34, 25, 12, 22, 11, 90};
    int n = sizeof(arr)/sizeof(arr[0]);
    printf("UnSorted array: \n");
    printArray(arr, n);
    bubbleSort(arr, n);
    printf("Sorted array: \n");
    printArray(arr, n);
    return 0;
}


// Optimized implementation of Bubble sort
#include <stdio.h>

void swap(int *xp, int *yp)
{
    int temp = *xp;
    *xp = *yp;
    *yp = temp;
}

// An optimized version of Bubble Sort
void optimizedBubbleSort(int arr[], int n)
{
    int i, j,temp;
    int swapped;
    for (i = 0; i < n-1; i++) // loop for the number of trips
    {
        swapped = 0; //swapped is set to false for every trip

        for (j = 0; j < n-i-1; j++) // loop for the number of comparisons per trip
        {
            if (arr[j] > arr[j+1]) // > for ascending order and < for descending order
```

```
 73:             {
 74:                 //swap(&arr[j], &arr[j+1]);
 75:                 temp=arr[j];
 76:                 arr[j]=arr[j+1];
 77:                 arr[j+1]=temp;
 78:                 swapped = 1;// if there is some swapping,
    swapped is set to true
 79:             }
 80:             printf("\ni=%d,j=%d,swapped=%d\n",i,j,
    swapped);
 81:             printArray(arr, n);
 82:         }
 83:
 84:         // IF no two elements were swapped by inner loop,
    then break
 85:         if (swapped == 0)
 86:             break;
 87:     }
 88: }
 89:
 90: /* Function to print an array */
 91: void printArray(int arr[], int size)
 92: {
 93:     int i;
 94:     for (i=0; i < size; i++)
 95:         printf("%d ", arr[i]);
 96:     printf("\n");
 97: }
 98:
 99: // Driver program to test above functions
100: int main()
101: {
102:     int arr[] = {20,10,30,40,50};
103:     //int arr[] = {40,30,20,10};
104:     int n = sizeof(arr)/sizeof(arr[0]);
105:     printf("Unsorted array: \n");
106:     printArray(arr, n);
107:
108:     optimizedBubbleSort(arr, n);
```

```c
    printf("Sorted array: \n");
    printArray(arr, n);
    return 0;
}


// C program for implementation of selection sort
#include <stdio.h>

void swap(int *xp, int *yp)
{
    int temp = *xp;
    *xp = *yp;
    *yp = temp;
}

void selectionSort(int arr[], int n)
{
    int i, j, min_idx;

    // One by one, move boundary of unsorted subarray
    for (i = 0; i < n-1; i++)
    {
        // Find the minimum element in unsorted array
        min_idx = i;
        for (j = i+1; j < n; j++)
        {
           if (arr[j] < arr[min_idx])
                min_idx = j;
        }

        // Swap the found minimum element with the first
element
        swap(&arr[min_idx], &arr[i]);
    }
}

/* Function to print an array */
void printArray(int arr[], int size)
```

```
147: {
148:     int i;
149:     for (i=0; i < size; i++)
150:         printf("%d ", arr[i]);
151:     printf("\n");
152: }
153:
154: // Driver program to test above functions
155: int main()
156: {
157:     int arr[] = {64, 25, 12, 22, 11};
158:     int n = sizeof(arr)/sizeof(arr[0]);
159:     selectionSort(arr, n);
160:     printf("Sorted array: \n");
161:     printArray(arr, n);
162:     return 0;
163: }
164:
165:
166: //C program for insertion sort
167: #include <math.h>
168: #include <stdio.h>
169:
170: /* Function to sort an array
171: using insertion sort*/
172: void insertionSort(int arr[], int n)
173: {
174:     int i, key, j;
175:     for (i = 1; i < n; i++)
176:     {
177:         key = arr[i];
178:         j = i - 1;
179:
180:         /* Move elements of arr[0..i-1],
181:         that are greater than key,
182:         to one position ahead of
183:         their current position */
184:         while (j >= 0 && arr[j] > key)
185:         {
```

```
186:             arr[j + 1] = arr[j];
187:             j = j - 1;
188:         }
189:         arr[j + 1] = key;
190:     }
191: }
192:
193: // A utility function to print
194: // an array of size n
195: void printArray(int arr[], int n)
196: {
197:     int i;
198:     for (i = 0; i < n; i++)
199:         printf("%d ", arr[i]);
200:     printf("\n");
201: }
202:
203: // Driver code
204: int main()
205: {
206:     int arr[] = {12, 11, 13, 5, 6,11};
207:     int n = sizeof(arr) / sizeof(arr[0]);
208:
209:     insertionSort(arr, n);
210:     printArray(arr, n);
211:
212:     return 0;
213: }
214:
215:
216: //Write a program to implement merge sort.
217: #include <stdio.h>
218: #define size 10
219: void merge(int a[], int, int, int);
220: void merge_sort(int a[],int, int);
221: void main()
222: {
223:     int arr[size], i, n;
224:     printf("\n Enter the number of elements in the array
    : ");
```

```
225:     scanf("%d", &n);
226:     printf("\n Enter the elements of the array: ");
227:     for(i=0;i<n;i++)
228:     {
229:         scanf("%d", &arr[i]);
230:     }
231:     merge_sort(arr, 0, n-1);
232:     printf("\n The sorted array is: \n");
233:     for(i=0;i<n;i++)
234:         printf(" %d\t", arr[i]);
235: }
236: void merge(int arr[], int beg, int mid, int end)
237: {
238:     int i=beg, j=mid+1, index=beg, temp[size], k;
239:
240:     //comparing the elements in the two halves and
    copying the elements into temp array
241:     while((i<=mid) && (j<=end))
242:     {
243:         if(arr[i] < arr[j])
244:         {
245:             temp[index] = arr[i];
246:             i++;
247:         }
248:         else
249:         {
250:             temp[index] = arr[j];
251:             j++;
252:         }
253:          index++;
254:      }
255:
256:     //if there are some remaining elements in the first
    half
257:     for(;i<=mid;i++)
258:         {
259:             temp[index] = arr[i];
260:             index++;
261:         }
```

```
262:
263:     //if there are some remaining elements in the Second
    half
264:     for(;j<=end;j++)
265:         {
266:             temp[index] = arr[j];
267:             index++;
268:         }
269:
270:     //coping all the sorted elements back to the original
    array from temp array
271:     for(k=beg;k<index;k++)
272:         arr[k] = temp[k];
273: }
274: void merge_sort(int arr[], int beg, int end)
275: {
276:     int mid;
277:     if(beg<end)
278:     {
279:         mid = (beg+end)/2;
280:         merge_sort(arr, beg, mid);
281:         merge_sort(arr, mid+1, end);
282:         merge(arr, beg, mid, end);
283:     }
284: }
285:
286: //Implementation of Quicksort in C
287: #include <stdio.h>
288: void quicksort (int [], int, int);
289: int main()
290: {
291:     int array[25];
292:     int size, i;
293:
294:     printf("Enter the number of elements: ");
295:     scanf("%d", &size);
296:     printf("Enter the elements to be sorted:\n");
297:     for(i = 0; i < size; i++)
298:     {
```

```
299:         scanf("%d", &array[i]);
300:     }
301:     quicksort(array, 0, size - 1);
302:     printf("Ascending order list after applying quick
    sort:\n");
303:     for(i = 0; i < size; i++)
304:     {
305:         printf("%d ", array[i]);
306:     }
307:     printf("\n");
308:
309:     return 0;
310: }
311: void quicksort(int array[], int low, int high)
312: {
313:     int pivot, left, right, temp;
314:     if(low < high)
315:     {
316:         pivot = low;
317:         left = low;
318:         right = high;
319:         while (left < right)
320:         {
321:         /*Increase left until an element greater than
    pivot is found*/
322:             while (left <= high && array[left] <=
    array[pivot])
323:             {
324:                 left++;
325:             }
326:
327:         /*Decrease right until an element less than or
    equal to pivot is found*/
328:             while (right >= low && array[right] >
    array[pivot])
329:             {
330:                 right--;
331:             }
332:
```

```
333:         /*
334:         if left<right, then swap array[left] and
     array[right]
335:         */
336:             if (left < right)
337:             {
338:                 temp = array[left];
339:                 array[left] = array[right];
340:                 array[right] = temp;
341:             }
342:         }
343:         /*
344:         if left==right or left>right than swap
     array[right] and array[pivot]
345:         */
346:         temp = array[right];
347:         array[right] = array[pivot];
348:         array[pivot] = temp;
349:         quicksort(array, low, right - 1);
350:         quicksort(array, right + 1, high);
351:     }
352: }
353:
354: // Counting sort in C
355: #include<stdio.h>
356: #define MAX 100
357:
358: void countSort(int array[], int size)
359: {
360:     int output[MAX];
361:     int count[MAX];
362:     int max = array[0];
363:
364:     // Here we find the largest item in the array
365:     for (int i = 1; i < size; i++)
366:     {
367:          if (array[i] > max)
368:             max = array[i];
369:     }
```

```
370:
371:     // Initialize the count for each element in array to 0
372:     for (int i = 0; i <= max; ++i)
373:     {
374:         count[i] = 0;
375:     }
376:
377:     // For each element we store the count
378:     for (int i = 0; i < size; i++)
379:     {
380:         count[array[i]]++;
381:     }
382:
383:     // Store the cummulative count of each array
384:     for (int i = 1; i <= max; i++)
385:     {
386:         count[i] += count[i - 1];
387:     }
388:
389:     // Search the index of each element of the actual
    array in count array, and
390:     // place the elements in output array
391:     for (int i = size - 1; i >= 0; i--)
392:     {
393:         output[--count[array[i]]] = array[i]; //here
    predecrement is important
394:     }
395:
396:     // Transfer the sorted items into actual array
397:     for (int i = 0; i < size; i++)
398:     {
399:         array[i] = output[i];
400:     }
401: }
402:
403: // printing items of the array
404: void display(int array[], int size)
405: {
406:     for (int i = 0; i < size; i++)
```

```
407:         printf("%d ",array[i]);
408:     printf("\n");
409: }
410:
411: // Driver code
412: int main() {
413:     int array[] = {2, 5, 2, 8, 1, 4, 1};
414:     int n = sizeof(array) / sizeof(array[0]);
415:
416:     countSort(array, n);
417:
418:     display(array, n);
419:
420:     return 0;
421: }
422:
423: /*
424: Algo radix Sort
425: radixSort(array)
426:   d <- maximum number of digits in the largest element
427:   create d buckets of size 0-9
428:   for i <- 0 to d
429:     sort the elements according to ith place digits using
    countingSort
430:
431: countingSort(array, d)
432:   max <- find largest element among dth place elements
433:   initialize count array with all zeros
434:   for j <- 0 to size
435:     find the total count of each unique digit in dth
    place of elements and
436:     store the count at jth index in count array
437:   for i <- 1 to max
438:     find the cumulative sum and store it in count array
    itself
439:   for j <- size down to 1
440:     restore the elements to array
441:     decrease count of each element restored by 1
442: */
```

```c


// Radix Sort in C Programming

#include <stdio.h>

// Function to get the largest element from an array
int getMax(int array[], int n) {
  int max = array[0];
  for (int i = 1; i < n; i++)
    if (array[i] > max)
      max = array[i];
  return max;
}

// Using counting sort to sort the elements on the basis
of significant places
void countingSort(int array[], int size, int place) {
  int output[size];
  int max = (array[0] / place) % 10;

  for (int i = 1; i < size; i++) {
    if (((array[i] / place) % 10) > max)
      max = array[i];
  }
  int count[max];

  for (int i = 0; i < max; ++i)
    count[i] = 0;

  // Calculate count of elements
  for (int i = 0; i < size; i++)
    count[(array[i] / place) % 10]++;

  // Calculate cumulative count
  for (int i = 1; i < 10; i++)
    count[i] += count[i - 1];

  // Place the elements in sorted order
```

```c
  for (int i = size - 1; i >= 0; i--) {
    output[count[(array[i] / place) % 10] - 1] = array[i];
    count[(array[i] / place) % 10]--;
  }

  for (int i = 0; i < size; i++)
    array[i] = output[i];
}

// Main function to implement radix sort
void radixsort(int array[], int size) {
  // Get maximum element
  int max = getMax(array, size);

  // Apply counting sort to sort elements based on place value.
  for (int place = 1; max / place > 0; place *= 10)
    countingSort(array, size, place);
}

// Print an array
void printArray(int array[], int size) {
  for (int i = 0; i < size; ++i) {
    printf("%d  ", array[i]);
  }
  printf("\n");
}

// Driver code
int main() {
  int array[] = {121, 432, 564, 23, 1, 45, 788};
  int n = sizeof(array) / sizeof(array[0]);
  radixsort(array, n);
  printArray(array, n);
}

//Bucket Sort in C
#include <stdio.h>
int getMax(int a[], int n) // function to get maximum element from the given array
```

```
519: {
520:     int max = a[0];
521:     for (int i = 1; i < n; i++)
522:     if (a[i] > max)
523:     max = a[i];
524:     return max;
525: }
526: void bucket(int a[], int n) // function to implement
     bucket sort
527: {
528:     int max = getMax(a, n); //max is the maximum element
     of array
529:     int bucket[max], i;
530:     for (int i = 0; i <= max; i++)
531:     {
532:         bucket[i] = 0;
533:     }
534:     for (int i = 0; i < n; i++)
535:     {
536:         bucket[a[i]]++;
537:     }
538:     for (int i = 0, j = 0; i <= max; i++)
539:     {
540:         while (bucket[i] > 0)
541:         {
542:             a[j++] = i;bucket[i]--;
543:         }
544:     }
545: }
546: void printArr(int a[], int n) // function to print array
     elements
547: {
548:     for (int i = 0; i < n; ++i)
549:     printf("%d ", a[i]);
550: }
551: int main()
552: {
553:     int a[] = {54, 12, 84, 57, 69, 41, 9, 5};
554:     int n = sizeof(a) / sizeof(a[0]); // n is the size of
     array
```

```
555:     printf("Before sorting array elements are - \n");
556:     printArr(a, n);
557:     bucket(a, n);
558:     printf("\nAfter sorting array elements are - \n");
559:     printArr(a, n);
560: }
561:
562: // Heap Sort in C
563: #include <stdio.h>
564: // Function to swap the position of two elements
565: void swap(int* a, int* b)
566: {
567:     int temp = *a;
568:     *a = *b;
569:     *b = temp;
570: }
571: // To heapify a subtree rooted with node i
572: // which is an index in arr[].
573: // n is size of heap
574: void heapify(int arr[], int N, int i)
575: {
576:     // Find largest among root, left child and right child
577:
578:     // Initialize largest as root
579:     int largest = i;
580:
581:     // left = 2*i + 1
582:     int left = 2 * i + 1;
583:
584:     // right = 2*i + 2
585:     int right = 2 * i + 2;
586:
587:     // If left child is larger than root
588:     if (left < N && arr[left] > arr[largest])
589:
590:         largest = left;
591:
592:     // If right child is larger than largest
593:     // so far
```

```
594:     if (right < N && arr[right] > arr[largest])
595:
596:         largest = right;
597:
598:     // Swap and continue heapifying if root is not largest
599:     // If largest is not root
600:     if (largest != i) {
601:
602:         swap(&arr[i], &arr[largest]);
603:
604:         // Recursively heapify the affected
605:         // sub-tree
606:         heapify(arr, N, largest);
607:     }
608: }
609:
610: // Main function to do heap sort
611: void heapSort(int arr[], int N)
612: {
613:
614:     // Build max heap
615:     for (int i = N / 2 - 1; i >= 0; i--)
616:
617:         heapify(arr, N, i);
618:
619:     // Heap sort
620:     for (int i = N - 1; i >= 0; i--)
621:     {
622:
623:         swap(&arr[0], &arr[i]);
624:         // Heapify root element to get highest element at
625:         // root again
626:         heapify(arr, i, 0);
627:     }
628: }
629:
630: // A utility function to print array of size n
631: void printArray(int arr[], int N)
632: {
```

```
633:     for (int i = 0; i < N; i++)
634:         printf("%d ", arr[i]);
635:     printf("\n");
636: }
637:
638: // Driver's code
639: int main()
640: {
641:     int arr[] = { 12, 11, 13, 5, 6, 7,5 };
642:     int N = sizeof(arr) / sizeof(arr[0]);
643:
644:     // Function call
645:     heapSort(arr, N);
646:     printf("Sorted array is\n");
647:     printArray(arr, N);
648: }
649:
650:
```