|  |  |
| --- | --- |
| **void bucketSort(int A[], int n**) {  **// Create and empty array of size BUCKET\_SIZE (empty buckets)**  bucket = (Listptr\*) malloc(sizeof(Listptr) \* BUCKET\_SIZE);    for( i = 0; i < BUCKET\_SIZE; i++)  bucket[i]=NULL;    **// Loop through the original array and put**  **each array element in a bucket**  for( i = 0; i < n; i++) {  pos = arr[i] / rangePerBucket;    **//Insert First Linkedlist**  temp = (Listptr) malloc(sizeof(List));  temp->data = arr[i];  temp->link = bucket[pos];  bucket[pos] = temp;  }    **// Sort each of the non-empty buckets using**  **sorting algorithm**  for( i = 0; i < BUCKET\_SIZE; i++)  bucket[i] = InsertionSort(bucket[i]);    **// Visit the buckets in order and put all**  **elements back into the original array**  for( i = 0, j= 0; i < BUCKET\_SIZE; i++) {  trav = &bucket[i];  while(\*trav != NULL) {  temp = \*trav;  arr[j++] = temp->data;  \*trav = temp->link;  free(temp);  }  }  } | |
| **void countsort(int arr[], int n, int k)** {  **// create an integer array of size**  **`n` to store the sorted array**  int output[n];  **// create an integer array of size**  **`k + 1`, initialized by all zero**  int freq[k + 1];  memset(freq, 0, sizeof(freq));  **// 1. Using the value of each item**  **in the input array as an index,**  **// store each integer's count in**  **`freq[]`**  for (int i = 0; i < n; i++)  freq[arr[i]]++;  **// 2. Calculate the starting index**  **for each integer**  int total = 0;  for (int i = 0; i < k + 1; i++) {  int oldCount = freq[i];  freq[i] = total;  total += oldCount;  }  **// 3. Copy to the output array,**  **preserving the order of inputs**  **with equal keys**  for (int i = 0; i < n; i++) {  output[freq[arr[i]]] = arr[i];  freq[arr[i]]++;  }  **// copy the output array back to**  **the input array**  for (int i = 0; i < n; i++)  arr[i] = output[i];  } | **void countingSort(int arr[], int size, int place)** {  int output[size + 1];  int max = (arr[0] / place) % 10;  for (int i = 1; i < size; i++) {  if (((arr [i] / place) % 10) > max)  max = arr [i];  }  int count[max + 1];  for (int i = 0; i < max; ++i)  count[i] = 0;  **// Calculate count of elements**  for (int i = 0; i < size; i++)  count[(arr [i] / place) % 10]++;    **// Calculate cumulative count**  for (int i = 1; i < 10; i++)  count[i] += count[i - 1];  **// Place the elements in sorted order**  for (int i = size - 1; i >= 0; i--) {  output[count[(arr [i] / place) % 10] - 1] = arr [i];  count[(arr [i] / place) % 10]--;  }  for (int i = 0; i < size; i++)  arr [i] = output[i];  }  **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);  } |
| **void gnomeSort(int \*array, int size)**{  int i, tmp;  for(i = 1; i < size; ){  if(array[i - 1] <= array[i])  ++i;  else {  tmp = array[i];  array[i] = array[i-1];  array[i - 1] = tmp;  --i;  if(i == 0)  i = 1;  }  }  } | **void shellSort(int arr[], int n)** {  **// Rearrange elements at each n/2, n/4,**  **n/8, ... intervals**  for (int inter = n / 2; inter > 0; inter /= 2) {  for (int i = inter; i < n; i += 1) {  int temp = arr[i];  int j;  for (j = i; j >= inter && arr [j - inter] > temp; j -= inter) {  arr [j] = arr [j - inter];  }  arr [j] = temp;  }  }  } |