**void countingSort(int A[], int size){**

int \*sorted = (int\*)calloc(size, sizeof(int));

int x, max = A[0];

**//find the largest number in the array**

for(x = 0; x < size; x++){

max = (max < A[x]) ? A[x] : max;

}

**//create count array with length max + 1**

int \*count = (int\*)calloc(max + 1, sizeof(int));

**//count each occurences of number**

for(x = 0; x < size; x++){

count[A[x]]++;

}

**//store cumulative sum**

for(x = 1; x < max + 1; x++){

count[x] += count[x - 1];

}

**//place elements in sorted array**

for(x = 0; x < size; x++){

sorted[count[A[x]] - 1] = A[x];

count[A[x]]--;

}

**//transfer sorted array to original array**

for(x = 0; x < size; x++){

A[x] = sorted[x];

}

}

**void shellSort(int A[], int size){**

int interval, x, y, temp = 0;

for(interval = size / 2; interval > 0; interval /= 2){

for(x = interval; x < size; x++){

temp = A[x];

**//comparing of nodes based on interval**

for(y = x; y >= interval && A[y - interval] > temp; y-= interval){

A[y] = A[y - interval];

}

A[y] = temp;

}

}

}

**void bucketSort(float A[], int size, int no\_of\_buckets**){

float max, min, range;

int x;

max = min = A[0];

**//find max and min elements**

for(x = 0; x < size; x++){

max = (max < A[x]) ? A[x] : max;

min = (min > A[x]) ? A[x] : min;

}

range = (max - min) / no\_of\_buckets;

**//create n buckets**

nodePtr \*buckets = (nodePtr\*)calloc(no\_of\_buckets+1, sizeof(struct node));

**//scatter the array element to individual buckets**

for(x = 0; x < size; x++){

int bucketIndex = (A[x] - min) / range;

insertSortedLL(&buckets[bucketIndex], A[x]);

}

**//gather the sorted elements to original array**

int ctr = 0;

for(x = 0; x < sizeof(buckets); x++){

nodePtr ptr;

for(ptr = buckets[x]; ptr != NULL; ptr = ptr->link){

A[ctr++] = ptr->num;

}

}

}

void insertSortedLL(nodePtr \*head, float num){

nodePtr \*trav;

nodePtr temp;

for(trav = head; \*trav != NULL && (\*trav)->num < num; trav = &(\*trav)->link){}

temp = (nodePtr)calloc(1, sizeof(struct node));

temp->num = num;

temp->link = \*trav;

\*trav = temp;

}

**void gnomeSort(int A[], int size**){

int ctr;

for(ctr = 1; ctr < size;){

if(ctr == 0){

ctr++;

}

**//if node before the ctr is larger; swap and move to previous node**

if(A[ctr - 1] > A[ctr]){

swap(&A[ctr - 1], &A[ctr]);

ctr--;

}

**//if node before the ctr is smaller or equal; move to next node**

if(A[ctr - 1] <= A[ctr]){

ctr++;

}

}

}

**void countingSort(int A[], int size, int exp)**{

int divide = exp / 10;

**//find largest digit based on exp**

int max = A[0] / divide % exp;

int x;

for(x = 0; x < size; x++){

max = (max < A[x] / divide % exp) ? A[x] / divide % exp : max;

}

**//create sorted and count array**

int \*sorted = (int\*)calloc(size, sizeof(int));

int \*count = (int\*)calloc(max + 1, sizeof(int));

**//count each occurrences of number**

for(x = 0; x < size; x++){

count[A[x] / divide % exp]++;

}

**//store cumulative sum**

for(x = 1; x < max + 1; x++){

count[x] += count[x - 1];

}

**//place elements in sorted array**

for(x = size - 1; x >= 0; x--){

sorted[count[A[x] / divide % exp] - 1] = A[x];

count[A[x] / divide % exp]--;

}

**//transfer sorted array to original array**

for(x = 0; x < size; x++){

A[x] = sorted[x];

}

}

**void radixSort(int A[], int size)**{

**// find largest number of digit**

int digit = 0, exp, x;

int max = findMax(A, size);

for(;max > 0; digit++, max /= 10){}

**// perform counting sort per digit**

for(x = 0, exp = 10; x < digit; x++, exp \*= 10){

countingSort(A, size, exp);

}

}