**Bubble Sort**

|  |
| --- |
| def bubbleSort(alist):  for x in range(len(alist)-1,0,-1):  for i in range(x):  if alist[i] > alist[i + 1]:  temp = alist[i]  alist[i] = alist[i + 1]  alist[i + 1] = temp |

**Counting Sort**

|  |
| --- |
| def countingsort( A, k ):  counter = [0] \* ( k + 1 )  for i in A:  counter[i] += 1    index = 0;  for i in range( len( counter ) ):  while 0 < counter[i]:  A[index] = i  index += 1  counter[i] -= 1 |

**Insertion Sort**

|  |
| --- |
| def insertionsort(list) :  for x in range (1,len(list)) :  value = list[x]  while x > 0 and list[x - 1] > value:  list[x] = list[x - 1]  x = x - 1  list[x] = value |

**Selection Sort**

|  |
| --- |
| def findmax(a,i):  m = 0  for i in range(1,i+1):  if a[i] > a[m]:  m = i  return m  def selectionsort(a):  for i in range(len(a)-1,-1,-1):  j = findmax(a,i)  a[i],a[j] = a[j],a[i] |

**Quick Sort**

|  |
| --- |
| def partition(A, p, r):  x = A[r]  i = p - 1  for j in range(p, r):  if A[j] <= x:  i += 1  A[i], A[j] = A[j], A[i]  A[i + 1], A[r] = A[r], A[i + 1]  return i + 1    def quicksort(A, p, r):  if p < r:  q = partition(A, p, r)  quicksort(A, p, q - 1)  quicksort(A, q + 1, r) |

**Radix Sort**

|  |
| --- |
| def radixsort( aList ):  RADIX = 10  maxLength = False  tmp , placement = -1, 1  while not maxLength:  maxLength = True  buckets = [list() for \_ in range( RADIX )]  for i in aList:  tmp = i // placement  buckets[tmp % RADIX].append( i )  if maxLength and tmp > 0:  maxLength = False  a = 0  for b in range( RADIX ):  buck = buckets[b]  for i in buck:  aList[a] = i  a += 1  placement \*= RADIX  return aList |

**Merge Sort**

|  |
| --- |
| def merge(A, p, q, r):  n1 = q - p + 1  n2 = r - q  L = []  R = []  for i in range(0, n1):  L.append(A[p + i])  for j in range(0, n2):  R.append(A[q + j + 1])  L.append(float('inf'))  R.append(float('inf'))  i = 0  j = 0  for k in range(p,r + 1):  if L[i] <= R[j]:  A[k] = L[i]  i = i + 1  else:  A[k] = R[j]  j = j + 1        def merge\_sort(A, p, r):  if p < r :  q = int((p + r) / 2)  merge\_sort(A, p, q)  merge\_sort(A, q + 1, r)  merge(A, p, q, r) |

**Heap Sort**

|  |
| --- |
| heapsize = 0  def parent(i):  return int(i/2)  def left(i):  return int(2\*i)  def right(i):  return int(2\*i)+1  def maxheapify(a,i):  global heapsize  l = left(i)  r = right(i)  if l < heapsize and a[l] > a[i]:  largest = l  else:  largest = i  if r < heapsize and a[r] > a[largest]:  largest = r  if largest != i:  a[i],a[largest] = a[largest],a[i]  maxheapify(a,largest)    def build\_max\_heap(a):  global heapsize  heapsize = len(a)  for i in range(int(len(a)/2)-1,-1,-1):  maxheapify(a,i)    def heapsort(a):  global heapsize  build\_max\_heap(a)  for i in range (len(a)-1,0,-1):  a[0],a[i] = a[i], a[0]  heapsize -= 1  maxheapify(a,0) |

**Bucket Sort**

|  |
| --- |
| import math  def insertionsort(list) :  for x in range (1,len(list)) :  value = list[x]  while x > 0 and list[x - 1] > value:  list[x] = list[x - 1]  x = x - 1  list[x] = value  def bucketsort( A ):  code = hashing( A )  buckets = [list() for \_ in range( code[1] )]  for i in A:  x = re\_hashing( i, code )  buck = buckets[x]  buck.append( i )  for bucket in buckets:  insertionsort( bucket )  index = 0  for b in range( len( buckets ) ):  for v in buckets[b]:  A[index] = v  index += 1    def hashing( A ):  m = A[0]  for i in range( 1, len( A ) ):  if ( m < A[i] ):  m = A[i]  result = [m, int( math.sqrt( len( A ) ) )]  return result      def re\_hashing( i, code ):  return int( i / code[0] \* ( code[1] - 1 ) ) |