mad-prac-4-linear-time-sort-1

June 28, 2024

[1]:
```python
# PRACTICAL 4.1 - COUNTING SORT
def counting_sort(A):
  M=max(A)
  print('Max element =',M)
  n=len(A)
  print('length of A =',n)
  print('size of auxillary array =',M+1)
  # create the auxillary array of size M+1 intialised by 0
  C=[0]*(M+1)
  print('Auxillary Array =',C)
  # Generate the Freq array C
  for i in A:
    C[i]=C[i]+1
  print('Freq Array =',C)
  # Generate the Cumu freq array C
  for i in range(1,M+1):
    C[i]=C[i]+C[i-1]
  print('Cumulative Freq Array =',C)
  # Generate the output array B of size n=len(A)
  B=[0]*(n)
  # Fill in the array B strating from last index which is n - 1
  for i in range(n-1,-1,-1):
      B[C[A[i]] - 1] = A[i]
      C[A[i]] = C[A[i]]-1
  return B
```

[2]:
```python
# Take an array with elements between 0 to 9
A=[2, 5, 3, 0, 2, 3, 0, 3]
print('Orginal Array :',A)
B=counting_sort(A)
print('Sorted Array :',B)
```

```
Orginal Array : [2, 5, 3, 0, 2, 3, 0, 3]
Max element = 5
length of A = 8
size of auxillary array = 6
Auxillary Array = [0, 0, 0, 0, 0, 0]
Freq Array = [2, 0, 2, 3, 0, 1]
```

```
Cumulative Freq Array = [2, 2, 4, 7, 7, 8]
Sorted Array : [0, 0, 2, 2, 3, 3, 3, 5]
```

[3]:
```python
# Take an array with elements between 0 to 9
A=[1, 3, 6, 2, 2, 0, 0, 4, 6, 5, 1]
print('Orginal Array :',A)
B=counting_sort(A)
print('Sorted Array :',B)
```

```
Orginal Array : [1, 3, 6, 2, 2, 0, 0, 4, 6, 5, 1]
Max element = 6
length of A = 11
size of auxillary array = 7
Auxillary Array = [0, 0, 0, 0, 0, 0, 0]
Freq Array = [2, 2, 2, 1, 1, 1, 2]
Cumulative Freq Array = [2, 4, 6, 7, 8, 9, 11]
Sorted Array : [0, 0, 1, 1, 2, 2, 3, 4, 5, 6, 6]
```

[4]:
```python
# PRACTICAL 4.2 RADIX SORT
# Using counting sort to sort the elements on the basis of significant(decimal) places
def counting_sort(A,place): # Counting_sort has addiitonal argument of place value
  n = len(A)
  B = [0] * n
  C = [0] * 10
# Calculate count of elements using Auxillary Array of size 10 taking digits from 0 to 9
  for i in range(0,n):
    index = A[i] // place # divide by place value
    C[index % 10] += 1
# Calculate cumulative count for the Auxillary Array
  for i in range(1, 10):
    C[i] += C[i - 1]
    # Place the elements in sorted order as per the digits at place value
    i = n - 1
  while i >= 0:
    index = A[i] // place # Divide A[i] by plac value to retain the quotients
    B[C[index % 10] - 1] = A[i]
    C[index % 10] -= 1
    i -= 1
  for i in range(0,n):
    A[i] = B[i]
  print(A)
```

[5]:
```python
# Main function to implement radix sort
def radix_sort(A):
```

```
  # Get maximum element
  max_element = max(A)
  # Apply counting sort to sort elements based on place value.
  place = 1
  print('place =',place)
  while max_element // place > 0:
    counting_sort(A, place)
    # Call for counting sort for d times if d = no.of digits in the array element
    place *= 10 # moving to digits from units to tens to hundreds.... place
    print('place =',place)
```

[6]:
```
# Test Code for checking radix sort
A = [121, 432, 564, 23, 1, 45, 788]
radix_sort(A)
print(A)
```

```
place = 1
[121, 1, 432, 23, 564, 45, 788]
place = 10
[1, 121, 23, 432, 45, 564, 788]
place = 100
[1, 23, 45, 121, 432, 564, 788]
place = 1000
[1, 23, 45, 121, 432, 564, 788]
```

[7]:
```
# Another test case
A = [181,289,390,121,145,736,514,212]
radix_sort(A)
print(A)
```

```
place = 1
[390, 181, 121, 212, 514, 145, 736, 289]
place = 10
[212, 514, 121, 736, 145, 181, 289, 390]
place = 100
[121, 145, 181, 212, 289, 390, 514, 736]
place = 1000
[121, 145, 181, 212, 289, 390, 514, 736]
```

[8]:
```
# Ascending array
A = [11,28,39,121,145,363,514,612]
radix_sort(A)
print(A)
```

```
place = 1
[11, 121, 612, 363, 514, 145, 28, 39]
place = 10
```

```
[11, 612, 514, 121, 28, 39, 145, 363]
place = 100
[11, 28, 39, 121, 145, 363, 514, 612]
place = 1000
[11, 28, 39, 121, 145, 363, 514, 612]
```

[9]:
```
# descending array
A = [181,89,39,36,21,14,10,2]
radix_sort(A)
print(A)
```

```
place = 1
[10, 181, 21, 2, 14, 36, 89, 39]
place = 10
[2, 10, 14, 21, 36, 39, 181, 89]
place = 100
[2, 10, 14, 21, 36, 39, 89, 181]
place = 1000
[2, 10, 14, 21, 36, 39, 89, 181]
```

0.1 Radix Sort - Worst Case ,Average Case ,Best Case Time Complexity

[10]:
```
# random order array - average time complexity
print("Radix Sort of Average Time Complexity")
A = [181,89,232,36,121,14,410,2]
radix_sort(A)
print(A)
```

```
Radix Sort of Average Time Complexity
place = 1
[410, 181, 121, 232, 2, 14, 36, 89]
place = 10
[2, 410, 14, 121, 232, 36, 181, 89]
place = 100
[2, 14, 36, 89, 121, 181, 232, 410]
place = 1000
[2, 14, 36, 89, 121, 181, 232, 410]
```

[11]:
```
# Test case Worst case time complexity
print("Radix Sort of Worst Time Complexity")
A = [181,892,232,136,121,614,410,2525147]
radix_sort(A)
print(A)
```

```
Radix Sort of Worst Time Complexity
place = 1
[410, 181, 121, 892, 232, 614, 136, 2525147]
place = 10
```

```
[410, 614, 121, 232, 136, 2525147, 181, 892]
place = 100
[121, 136, 2525147, 181, 232, 410, 614, 892]
place = 1000
[121, 136, 181, 232, 410, 614, 892, 2525147]
place = 10000
[121, 136, 181, 232, 410, 614, 892, 2525147]
place = 100000
[121, 136, 181, 232, 410, 614, 892, 2525147]
place = 1000000
[121, 136, 181, 232, 410, 614, 892, 2525147]
place = 10000000
[121, 136, 181, 232, 410, 614, 892, 2525147]
```

[12]:
```
# Best Case - all the elements are one digited.
print("Radix Sort of Best Time Complexity")
A = [8,9,2,6,0,1,4,7,3]
radix_sort(A)
print(A)
```

```
Radix Sort of Best Time Complexity
place = 1
[0, 1, 2, 3, 4, 6, 7, 8, 9]
place = 10
[0, 1, 2, 3, 4, 6, 7, 8, 9]
```

[13]:
```
# PRACTICAL 4.3 BUCKET SORT
def bucket_sort(array):
  bucket = [] # Bucket Auxillary array
  # Create empty buckets
  for i in range(len(array)):
    bucket.append([]) # append the empty bucket to accomodate the elements of␣
 ↪array
  print('Bucket Auxillary array :',bucket)
    # Insert elements into their respective buckets
  for j in array:
    index_b = int(10 * j) # for example j = 0.78 then int[10*0.78] = int [7.8]␣
 ↪= 7
    print('bucket index  for ',j,' is ',index_b)
    bucket[index_b].append(j)
  print('Bucket array after insertion:',bucket)
    # Sort the elements of each bucket
  for i in range(len(array)):
    bucket[i] = sorted(bucket[i])
    # Get the sorted elements
    k = 0
    for i in range(len(array)):
```

```
      for j in range(len(bucket[i])):
        array[k] = bucket[i][j]
        k += 1
    print('Bucket array with each bucket sorted :',bucket)
  return array
```

[14]:
```
# Test Code for checking bucket sort
array=[0.78,0.17,0.39,0.26,0.1,0.2,0.09,0.72,0.94,0.21,0.12,0.23,0.68,0.45,0.
 ↪53,0.8]
print('Orginal array :',array)
B=bucket_sort(array)
print('Sorted array :',array)
```

```
Orginal array : [0.78, 0.17, 0.39, 0.26, 0.1, 0.2, 0.09, 0.72, 0.94, 0.21, 0.12,
0.23, 0.68, 0.45, 0.53, 0.8]
Bucket Auxillary array : [[], [], [], [], [], [], [], [], [], [], [], [], [],
[], [], []]
bucket index  for  0.78  is  7
bucket index  for  0.17  is  1
bucket index  for  0.39  is  3
bucket index  for  0.26  is  2
bucket index  for  0.1  is  1
bucket index  for  0.2  is  2
bucket index  for  0.09  is  0
bucket index  for  0.72  is  7
bucket index  for  0.94  is  9
bucket index  for  0.21  is  2
bucket index  for  0.12  is  1
bucket index  for  0.23  is  2
bucket index  for  0.68  is  6
bucket index  for  0.45  is  4
bucket index  for  0.53  is  5
bucket index  for  0.8  is  8
Bucket array after insertion: [[0.09], [0.17, 0.1, 0.12], [0.26, 0.2, 0.21,
0.23], [0.39], [0.45], [0.53], [0.68], [0.78, 0.72], [0.8], [0.94], [], [], [],
[], [], []]
Bucket array with each bucket sorted : [[0.09], [0.17, 0.1, 0.12], [0.26, 0.2,
0.21, 0.23], [0.39], [0.45], [0.53], [0.68], [0.78, 0.72], [0.8], [0.94], [],
[], [], [], [], []]
Bucket array with each bucket sorted : [[0.09], [0.1, 0.12, 0.17], [0.26, 0.2,
0.21, 0.23], [0.39], [0.45], [0.53], [0.68], [0.78, 0.72], [0.8], [0.94], [],
[], [], [], [], []]
Bucket array with each bucket sorted : [[0.09], [0.1, 0.12, 0.17], [0.2, 0.21,
0.23, 0.26], [0.39], [0.45], [0.53], [0.68], [0.78, 0.72], [0.8], [0.94], [],
[], [], [], [], []]
Bucket array with each bucket sorted : [[0.09], [0.1, 0.12, 0.17], [0.2, 0.21,
0.23, 0.26], [0.39], [0.45], [0.53], [0.68], [0.78, 0.72], [0.8], [0.94], [],
```

```
[], [], [], [], []]
Bucket array with each bucket sorted : [[0.09], [0.1, 0.12, 0.17], [0.2, 0.21,
0.23, 0.26], [0.39], [0.45], [0.53], [0.68], [0.78, 0.72], [0.8], [0.94], [],
[], [], [], [], []]
Bucket array with each bucket sorted : [[0.09], [0.1, 0.12, 0.17], [0.2, 0.21,
0.23, 0.26], [0.39], [0.45], [0.53], [0.68], [0.78, 0.72], [0.8], [0.94], [],
[], [], [], [], []]
Bucket array with each bucket sorted : [[0.09], [0.1, 0.12, 0.17], [0.2, 0.21,
0.23, 0.26], [0.39], [0.45], [0.53], [0.68], [0.78, 0.72], [0.8], [0.94], [],
[], [], [], [], []]
Bucket array with each bucket sorted : [[0.09], [0.1, 0.12, 0.17], [0.2, 0.21,
0.23, 0.26], [0.39], [0.45], [0.53], [0.68], [0.72, 0.78], [0.8], [0.94], [],
[], [], [], [], []]
Bucket array with each bucket sorted : [[0.09], [0.1, 0.12, 0.17], [0.2, 0.21,
0.23, 0.26], [0.39], [0.45], [0.53], [0.68], [0.72, 0.78], [0.8], [0.94], [],
[], [], [], [], []]
Bucket array with each bucket sorted : [[0.09], [0.1, 0.12, 0.17], [0.2, 0.21,
0.23, 0.26], [0.39], [0.45], [0.53], [0.68], [0.72, 0.78], [0.8], [0.94], [],
[], [], [], [], []]
Bucket array with each bucket sorted : [[0.09], [0.1, 0.12, 0.17], [0.2, 0.21,
0.23, 0.26], [0.39], [0.45], [0.53], [0.68], [0.72, 0.78], [0.8], [0.94], [],
[], [], [], [], []]
Bucket array with each bucket sorted : [[0.09], [0.1, 0.12, 0.17], [0.2, 0.21,
0.23, 0.26], [0.39], [0.45], [0.53], [0.68], [0.72, 0.78], [0.8], [0.94], [],
[], [], [], [], []]
Bucket array with each bucket sorted : [[0.09], [0.1, 0.12, 0.17], [0.2, 0.21,
0.23, 0.26], [0.39], [0.45], [0.53], [0.68], [0.72, 0.78], [0.8], [0.94], [],
[], [], [], [], []]
Bucket array with each bucket sorted : [[0.09], [0.1, 0.12, 0.17], [0.2, 0.21,
0.23, 0.26], [0.39], [0.45], [0.53], [0.68], [0.72, 0.78], [0.8], [0.94], [],
[], [], [], [], []]
Bucket array with each bucket sorted : [[0.09], [0.1, 0.12, 0.17], [0.2, 0.21,
0.23, 0.26], [0.39], [0.45], [0.53], [0.68], [0.72, 0.78], [0.8], [0.94], [],
[], [], [], [], []]
Bucket array with each bucket sorted : [[0.09], [0.1, 0.12, 0.17], [0.2, 0.21,
0.23, 0.26], [0.39], [0.45], [0.53], [0.68], [0.72, 0.78], [0.8], [0.94], [],
[], [], [], [], []]
Sorted array : [0.09, 0.1, 0.12, 0.17, 0.2, 0.21, 0.23, 0.26, 0.39, 0.45, 0.53,
0.68, 0.72, 0.78, 0.8, 0.94]
```

[ ]:

```
```