Cara Kerja:

1. Masukkan sebuah array bilangan
2. Hitung jumlah data dari array tersebut
3. Hitung nilai terbesar dari array tersebut
4. Mulai radix sort dengan menghitung digit terbesar dari array tersebut yang didapatkan dari pembagian array dengan digit dimana digit utama paling kecil pada sebuah bilangan bulat adalah 1. Digit ini akan menjadi acuan urutan pada angka dan acuan pada counting sort.
5. Mulai perulangan untuk melakukan fungsi counting sort pada setiap digit.
6. Counting sort dimulai dengan menambah 2 variabel array. Yang satu diperuntukkan untuk menyimpan hasil sementara di counting sort, misal hasil[], dan yang kedua diperuntukkan untuk menghitung perulangan bilangan pada array, misal hitung[].
7. Memulai perulangan untuk mengisi array hitung[].
8. Menentukan posisi bilangan pada array menggunakan hasil dari array hitung[].
9. Memulai menyusun bilangan menggunakan posisi yang sudah ditentukan pada langkah sebelumnya pada array hasil[].
10. Menyalin array hasil[] pada array bilangan yang utama.
11. Mengulang langkah 6-10 hingga semua digit telah di susun.
12. Program selesai

Notasi Algoritma:

Judul: Radix Sorting

Kamus

radix\_arr: array

n, i: integer

Prosedur radixSort(radix\_arr: array, n: integer)

{melakukan pengulangan counting sort ditambah dengan penyesuaian digit pada array}

Prosedur countingSort(radix\_arr: array, n: integer, digit: integer)

{fungs sorting terhadap array menggunakan metode counting sort}

Fungsi nilaiMax(radix\_arr: array, n: integer) 🡨 integer

{mendapatkan nilai terbesar dari sebuah fungsi}

Algoritma

Input(n)

i 🡨0

for i until n-1 do

input(radix\_arr[i])

endfor

i🡨0

for i until n-1 do

output(radix\_arr[i])

endfor

radixSort(radix\_arr, n)

i🡨0

for i until n-1 do

output(radix\_arr[i])

endfor

Fungsi nilaiMax(radix\_arr: array, n: integer) 🡨 integer

{mendapatkan nilai terbesar dari sebuah fungsi}

{Kamus Lokal}

max: integer 🡨radix\_arr[0]

i: integer

{Algoritma}

i 🡨0

for i until n - 1 do

if radix\_arr[i] > max then

max 🡸 radix\_arr[i]

endif

endfor

🡪 max

Prosedur radixSort(radix\_arr: array, n: integer)

{melakukan pengulangan counting sort ditambah dengan penyesuaian digit pada array}

{Kamus Lokal}

max: integer

digit: integer

Prosedur countingSort(radix\_arr: array, n: integer, digit: integer)

{fungs sorting terhadap array menggunakan metode counting sort}

Fungsi nilaiMax(radix\_arr: array, n: integer) 🡨 integer

{mendapatkan nilai terbesar dari sebuah fungsi}

{Algoritma}

nilaiMax(radix\_arr, n)

digit 🡨 1

for digit until max/digit < 1 do

countingSort(radix\_arr, n, digit)

endfor

Prosedur countingSort(radix\_arr: array, n: integer, digit: integer)

{fungs sorting terhadap array menggunakan metode counting sort}

{Kamus Lokal}

hasil: array

count: array 🡨{0}

i: integer

{Algoritma}

i 🡨 0

for i until n-1 do

count[(radix\_arr[i]/digit) mod 10] 🡨 count[(radix\_arr[i]/digit) mod 10] + 1

endfor

i 🡨 1

for i until 9 do

count[i] 🡨 count[i] + count[i - 1]

endfor

i 🡨 n - 1

For i until 0 do

hasil[count[(radix\_arr[i]/digit) mod 10] - 1] 🡨 radix\_arr[i]

count[(radix\_arr[i]/digit) mod 10] = count[(radix\_arr[i]/digit) mod 10] - 1

endfor

i🡨0

for i until n-1 do

radix\_arr[i] 🡨 hasil[i]

endfor

Ilustrasi :

Asli = 3 13 4 6 4 17 6

Radix Sort:

Digit : 1

Counting sort:

0 = tidak ada

1 = tidak ada

2 = tidak ada

3 = 3, 13

4 = 4, 4

5 = tidak ada

6 = 6, 6

7 = 17

8 = tidak ada

9 = tidak ada

Hasil = 3, 13, 4, 4, 6, 6, 17

Digit : 10

Counting sort:

0 = 3, 4, 4, 6, 6

1 = 13, 17

2 = tidak ada

3 = tidak ada

4 = tidak ada

5 = tidak ada

6 = tidak ada

7 = tidak ada

8 = tidak ada

9 = tidak ada

Hasil = 3, 4, 4, 6, 6, 13, 17

Coret- coretan pengujian step by step:

3 1 6 4 6

(Bilangan[0]/digit)%10

(3/1)%10

3

Count[3] ++

0 ++

0 1 2 3 4 5 6 7 8 9 10

Count[10] = {0, 0, 0, 1 , 0 ,0 , 0, 0, 0, 0, 0}

(Bilangan[1]/digit)%10

(1/1)%10

1

Count[1] ++

0 ++

0 1 2 3 4 5 6 7 8 9 10

Count[10] = {0, 1, 0, 1 , 0 ,0 , 0, 0, 0, 0, 0}

(Bilangan[2]/digit)%10

(6/1)%10

6

Count[6] ++

0 ++

0 1 2 3 4 5 6 7 8 9 10

Count[10] = {0, 1, 0, 1 , 0 ,0 , 1, 0, 0, 0, 0}

(Bilangan[3]/digit)%10

(4/1)%10

4

Count[4] ++

0 ++

0 1 2 3 4 5 6 7 8 9 10

Count[10] = {0, 1, 0, 1 , 1 ,0 , 1, 0, 0, 0, 0}

(Bilangan[4]/digit)%10

(6/1)%10

6

Count[6] ++

1 ++

0 1 2 3 4 5 6 7 8 9 10

Count[10] = {0, 1, 0, 1 , 1 ,0 , 2, 0, 0, 0, 0}

Count[1] = count[1] + count[1 -1 =0]

Count[1] = 1 + 0

Count[1] = 1

Count[2] = count[2] + count[2 -1 = 1]

Count[2] = 0 + 1

Count[2] = 1

Count[3] = count[3] + count[3 -1 =2]

Count[3] = 1 + 1

Count[3] = 2

Count[4] = count[4] + count[4 -1 =3]

Count[4] = 1 + 2

Count[4] = 3

Count[5] = count[5] + count[5 -1 =4]

Count[5] = 0 + 3

Count[5] = 3

Count[6] = count[6] + count[6 -1 =5]

Count[6] = 2 + 3

Count[6] = 5

Count[10] = {0,1,1,2,3,3,5,5,5,5}

Bilangan = { 3 1 6 4 6}

I=banyakBilangan -1 = 5 - 1 = 4

Hasil[count[(bilangan[i]/digit)%10]-1] = bilangan[i]

count[(bilangan[i]/digit)%10]—

hasil sama Cuma dibalik saja dari kecil ke besar:

bilangan[0] = 3

hasil[count[(3/1)%10]-1] = 3

hasil[count[3] -1 ] = 3

hasil[2-1] = 3

hasil[1] = 3

bilangan[1] = 1

hasil[count[(1/1)%10]-1] = 1

hasil[count[1] -1 ] = 1

hasil[1-1] = 1

hasil[0] = 1

bilangan[2] = 6

hasil[count[(6/1)%10]-1] = 6

hasil[count[6] -1 ] = 6

hasil[5-1] = 6

hasil[4] = 6

bilangan[3] = 4

hasil[count[(4/1)%10]-1] = 4

hasil[count[4] -1 ] = 4

hasil[3-1] = 4

hasil[2] = 4

bilangan[4] = 6

hasil[count[(6/1)%10]-1] = 1

hasil[count[6] -1 ] = 1

hasil[4-1] = 1

hasil[3] = 1