IN244 – Strategi Algoritmik

02 – Sorting Lanjut

Metode Sorting Lanjut

- Shell Sort
- Quick Sort
- Merge Sort

SHELL SORT

SHELL SORT

- Penciptanya, Donald Shell
- Shell sort : perbaikan dari algoritma insertion sort.
- Kumpulan *N* elemen dibagi menjadi *K* segmen (sub list) , *K* digunakan sebagai *increment*.
- Komparasi nilai dilakukan untuk pasangan elemen yang berjarak *K*
- Nilai *K* akan mengecil selama jalannya program dan berakhir jika K = 1

Struktur Array

```
class Array():
#Indeks array : dari 1 sampai dengan Nmax
    def __init__(self,size):
        self.items = [None]*(size+1)
        self.Nmax = size
        self.N = 0
```

SHELL SORT

Pass	List (K=5)										Notes
1	77	62	14	9	30	21	80	25	70	55	Swap
	21	62	14	9	30	77	80	25	70	55	In order
	21	62	14	9	30	77	80	25	70	55	In order
	21	62	14	9	30	77	80	25	70	55	In order
	21	62	14	9	30	77	80	25	70	55	In order

- Nilai K diawali dengan K1 = N div 2
- Proses membandingkan 2 nilai yang berjarak *K* dan dilakukan swap jika urutannya tidak sesuai
- Setelah semua nilai dibandingkan dengan K=5, nilai K menjadi K = K div 2 utk proses berikutnya.
  - K2 = 5 div 2 = 2
  - K3 = 2 div 2 = 1

SHELL SORT

Pass	List (K=2)										Notes
2	21	62	14	9	30	77	80	25	70	55	Swap
	14	62	21	9	30	77	80	25	70	55	Swap
	14	9	21	62	30	77	80	25	70	55	In order
	14	9	21	62	30	77	80	25	70	55	In order
	14	9	21	62	30	77	80	25	70	55	In order
	14	9	21	62	30	77	80	25	70	55	Swap
	14	9	21	62	30	25	80	77	70	55	Swap
	14	9	21	25	30	62	80	77	70	55	In order
	14	9	21	25	30	62	80	77	70	55	Swap
	14	9	21	25	30	62	70	77	80	55	In order
	14	9	21	25	30	62	70	77	80	55	Swap
	14	9	21	25	30	62	70	55	80	77	Swap
	14	9	21	25	30	55	70	62	80	77	In order

SHELL SORT

Pass	List (K=1)										Notes
3	14	9	21	25	30	55	70	62	80	77	Swap
	9	14	21	25	30	55	70	62	80	77	In order
	9	14	21	25	30	55	70	62	80	77	In order
	9	14	21	25	30	55	70	62	80	77	In order
	9	14	21	25	30	55	70	62	80	77	In order
	9	14	21	25	30	55	70	62	80	77	In order
	9	14	21	25	30	55	70	62	80	77	Swap
	9	14	21	25	30	55	62	70	80	77	In order
	9	14	21	25	30	55	62	70	80	77	In order
	9	14	21	25	30	55	62	70	80	77	Swap
	9	14	21	25	30	55	62	70	77	80	In order

```
def ShellSort(self):
# current:indeks elemen yang akan diurutkan
# k : jarak utk perbandingan nilai
# i : indeks elemen yang dibandingkan
# temp:untuk pertukaran (swap)

    k = self.N // 2
    while (k > 0):
      current = k
      while (current <= self.N):
        temp = self.items[current]
        i = current-k
        while (i > 0) and (temp < self.items[i]):
            self.items[i+k] = self.items[i]
            i = i - k
        self.items[i+k] = temp
        current = current + 1
      k = k // 2   # nilai K berkurang
```

Nilai K

- Increment yang biasa digunakan
  - Shell (1, 2, 4, 8, …, N/2)
  - Hibbard (1, 3, 7, 15, …, $2^n - 1$)
  - Knuth (1, 4, 13, 40, …, $3k_{n-1} + 1$)
- Semuanya berakhir jika K = 1

QUICK SORT

QUICK SORT

- ***Quick sort*** :
  - Lebih efisien dari bubble sort, karena jumlah swap (pertukaran) hanya melibatkan nilai tertentu (tidak seluruh elemen array)
  - jumlah pertukaran data lebih sedikit
- Pada setiap iterasi, akan dipilih elemen yang disebut ***pivot***, dan membagi array menjadi 3 kelompok:
  - Kelompok dengan elemen, di mana nilainya < nilai pivot
  - Elemen Pivot
  - Kelompok dengan elemen, di mana nilainya > nilai pivot

Quick Sort

- Algoritma bersifat rekursif
- Algoritma bekerja sbb :
  - melakukan sorting untuk partisi kiri
  - melakukan sorting untuk partisi kanan
  - Gabungan ke dua partisi menghasilkan kumpulan data yang terurut

Proses di Partisi

- Tentukan elemen yang menjadi pivot
- Setelah proses perbandingan dan pertukaran, maka :
  - Semua elemen < pivot ada di partisi kiri
  - Semua elemen > pivot ada di partisi kanan
  - Pivot menempati posisi yang sudah tepat dan tidak akan berubah lagi

QUICK SORT

- Berikut ini data yang akan disort
- Pivot diambil dari elemen pada posisi ke N

1	4	8	9	0	11	5	10	7	6

QUICK SORT

- Index *left* pada elemen pertama
- Index *right* di sebelah kiri pivot

1	4	8	9	0	11	5	10	7	6
left								*right*	

- Algoritma akan memindahkan semua nilai < pivot utk berada di sebelah kiri dan semua nilai > pivot di sebelah kanan, dengan cara pertukaran data (swapping)

QUICK SORT

- Index *left* bergerak ke kanan dan akan berhenti pada nilai elemen > pivot

1	4	8	9	0	11	5	10	7	**6**
		left						*right*	

QUICK SORT

- Index right bergerak ke kiri dan akan berhenti pada nilai elemen < pivot

1	4	8	9	0	11	5	10	7	**6**
		left				*right*			

QUICK SORT

- Jika index *left* <= *right* maka elemen yang ditunjuk akan ditukar.

1	4	8	9	0	11	5	10	7	6
		left				*right*			

1	4	5	9	0	11	8	10	7	6
		left				*right*			

QUICK SORT

1	4	5	9	0	11	8	10	7	**6**
		left				*right*			

1	4	5	*9*	*0*	11	8	10	7	**6**
			left	*right*					

1	4	5	*0*	*9*	11	8	10	7	**6**
			left	*right*					

QUICK SORT

1	4	5	0	9	11	8	10	7	6
			left	*right*					

1	4	5	0	9	11	8	10	7	6
			right	*left*					

QUICK SORT

1	4	5	0	9	11	8	10	7	6
			right	*left*					

- Pada saat index *left* > *right*, pertukaran akan berhenti
- Proses partisi berakhir dengan menukar **pivot** elemen dengan elemen yang ditunjuk ***left*.**

1	4	5	0	6	11	8	10	7	9
			right	*left*					

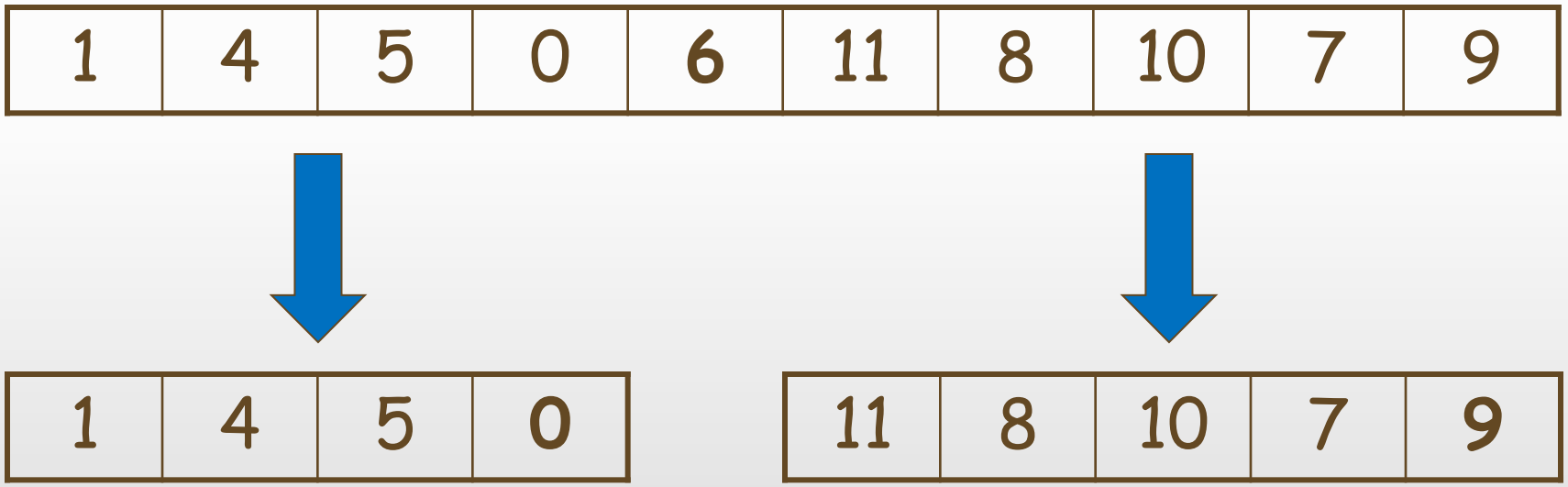
QUICK SORT

- Hasil proses partisi :
  - Semua elemen di kiri pivot, nilainya < pivot
  - Semua elemen di kanan pivot, nilainya > pivot
- Nilai pivot sudah menempati posisi yang tepat dan tidak akan berpindah lagi

1	4	5	0	**6**	11	8	10	7	9
			right	*left*					

QUICK SORT

- Proses akan dilanjutkan secara rekursif untuk merapikan urutan data di sub-array kiri dan sub- array kanan



QUICK SORT

- Merupakan divide-and-conquer algorithm
- Kecepatan algoritma ditentukan oleh PIVOT
- Jika pembagian partisi tidak seimbang, quicksort akan berjalan lambat
- Jika pembagian partisi seimbang, quicksort akan berjalan cepat
- PARTISI seimbang jika ukurannya hampir sama
  - Partisi kiri : [n/2]
  - Partisi kanan : [n/2]-1.

THE BEST PIVOT

- ***Median-of-Three Way:***
  - Pivot sebagai median dari semua elemen;
- Cara heuristik :
  - Memilih tiga elemen secara random
  - Memakai median dari ketiganya sebagai pivot.
- The worst pivot :
  - Elemen pertama array sbg pivot

Algoritma Quick Sort

```
def quickSort(self,leftMostIndex, rightMostIndex):
    #Lakukan proses untuk membuat partisi
    #Partisi kiri < pivot
    #Partisi kanan > pivot
    pivot = self.items[rightMostIndex]
    left = leftMostIndex
    right = rightMostIndex - 1
    while (left <= right):
        #Cari nilai elemen > pivot
        while (self.items[left] < pivot):
            left = left + 1
        #Cari nilai elemen < pivot
        while (right>=leftMostIndex and self.items[right]> pivot):
            right = right - 1
        #Swap keduanya
        if (left <= right):  #pastikan left <= right
            self.swap( left, right)
            left = left + 1
            right = right - 1
    #while sudah selesai
```

Algoritma Quick Sort

```
#Bagian ini setelah while selesai
#Pindahkan pivot elemen ke posisi yang tepat
self.swap(left, rightMostIndex)
#Lanjutkan proses utk partisi kiri dan kanan
#secara rekursif
if (leftMostIndex < right):
        self.quickSort(leftMostIndex, right)
if (left+1 < rightMostIndex ):
        self.quickSort(left+1, rightMostIndex)
```

MERGE SORT

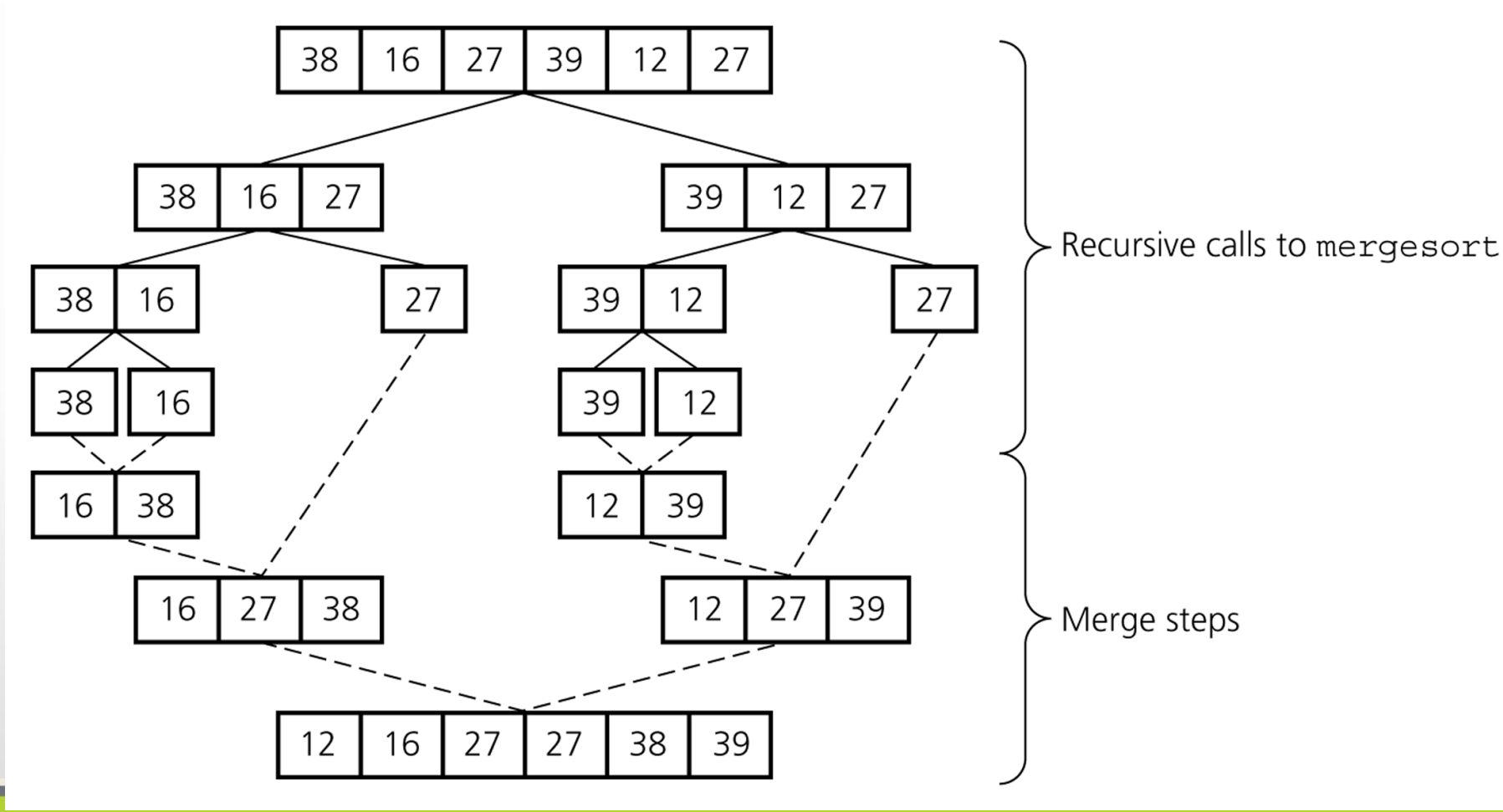
MERGE SORT

- Prinsip Merge sort :
  - Terdapat dua kumpulan data yang sudah terurut
  - Gabungkan kedua kumpulan data menjadi satu kumpulan data yang terurut
- Cara Kerja:
  - Data dalam array dibagi atas dua subarray yang belum urut, sampai ukuran tertentu
  - Urutkan data dalam setiap subarray
  - Merge (gabungkan) kedua subarray yang sudah diurutkan menjadi satu array yang terurut

MERGE SORT

- Dilakukan secara rekursif
- Array akan dibagi dua terus menerus, sampai ukuran subarray sama dengan 1
- Karena hanya satu elemen, berarti data sudah urut
- Selanjutnya dilakukan penggabungan subarray yang sudah urut.

MERGE SORT


38 16 27 39 12 27
38 16 27
39 12 27
38 16
27
39 12
27
38 16
39 12
16 38
12 39
16 27 38
12 27 39
12 16 27 27 38 39
Recursive calls to mergesort
Merge steps

MERGE SORT

- Algoritma terdiri atas:
  - 2 input arrays (arrayA and arrayB)
  - 1 ouput array (arrayC)
  - 3 variabel indeks
    - indexA : untuk array A
    - indexB : untuk array B
    - indexC : untuk array C

MERGE SORT

- Penggabungan :
  - Setiap elemen ArrayA[indexA] akan dibandingkan dengan arrayB[indexB]
  - Nilai yang lebih kecil akan disimpan ke arrayC[indexC]
  - Variabel index terkait akan ditambah 1
- Jika semua data dalam salah satu array selesai diproses, sisa data dalam array yang lain disalin ke arrayC.

Merge Sort

arrayA

1	13	24	26
indexA			

arrayB

2	15	27	38
indexB			

arrayA[indexA] dibandingkan dengan arrayB[indexB]. Nilai yang lebih kecil disimpan ke arrayC[indexC].

1 < 2, jadi arrayA[indexA] disalin ke arrayC[indexC]

arrayC

indexC							

Merge Sort

arrayA

1	13	24	26
	indexA		

arrayB

2	15	27	38
indexB			

2 < 13, jadi arrayB[indexB] Disalin ke arrayC[indexC]

arrayC

1							
	indexC						

Merge Sort

arrayA

1	13	24	26
	indexA		

arrayB

2	15	27	38
	indexB		

13 < 15 , jadi arrayA[indexA] disalin ke arrayC[indexC]

arrayC

1	2						
		indexC					

Merge Sort

arrayA

1	13	24	26
		indexA	

15 < 24, jadi arrayB[indexB] Disalin ke arrayC[indexC]

arrayB

2	15	27	38
	indexB		

arrayC

1	2	13					
			indexC				

Merge Sort

arrayA

1	13	24	26
		indexA	

arrayB

2	15	27	38
		indexB	

arrayC

1	2	13	15				
				indexC			

24 < 27, jadi arrayA[indexA] disalin ke arrayC[indexC]

Merge Sort

arrayA

1	13	24	26
			indexA

arrayB

2	15	27	38
		indexB	

26 < 27, jadi
arrayA[indexA] disalin ke
arrayC[indexC]

arrayC

1	2	13	15	24			
					indexC		

Merge Sort

arrayA

1	13	24	26

arrayB

2	15	27	38
		indexB	

arrayC

1	2	13	15	24	26		
						indexC	

Karena data arrayA sudah selesai dipr oses,
sisa data arrayB disalin ke arrayC

Hasil Merge Sort

arrayA

1	13	24	26

arrayB

2	15	27	38

arrayC

1	2	13	15	24	26	27	38

Algoritma Merge Sort (bagian divide)

```
def mergesort(self,first,last):
        if( first < last ):
             mid = (first + last)//2;
           # Urutkan bagian-1 dari array
             self.mergesort(first, mid);
           # Urutkan bagian-2 dari array
             self.mergesort(mid+1, last);
           # Merge kedua bagian
             self.merge(first, mid, last);
```

Algoritma Merge Sort (Bagian awal Merge)

```
def merge(self,first,mid,last):
#tempArray : array temporer utk tampung hasil merge
#Merge bagian-1 (indexA) dan bagian-2 (indexB)
        tempArray = [None]*(self.Nmax+1)
        indexA = first
        lastA = mid
        indexB = mid+1
        lastB = last

        indexC = indexA
        while( indexA <= lastA and indexB <= lastB ):
            if( self.items[indexA] < self.items[indexB] ):
                tempArray[indexC] = self.items[indexA]
                indexA = indexA + 1
            else:
                tempArray[indexC] = self.items[indexB]
                indexB = indexB + 1
            indexC = indexC + 1
        #while selesai
```

Algoritma Merge Sort (Bagian akhir Merge)

```
#Copy sisa array ke tempArray
#Hanya satu while saja yang dikerjakan
        while (indexA <= lastA):
            tempArray[indexC] = self.items[indexA]
            indexA = indexA + 1
            indexC = indexC + 1

        while (indexB <= lastB ):
            tempArray[indexC] = self.items[indexB]
            indexB = indexB + 1
            indexC = indexC + 1
#Salin kembali tempArray ke array semula
        indexC = first
        while (indexC <= last):
            self.items[indexC] = tempArray[indexC]
            indexC = indexC + 1
```

Efficiency Summary

Sort	Worst Case	Average Case
Insertion	$O(n^2)$	$O(n^2)$
Shell	$O(n^{1.5})$	$O(n^{1.25})$
Selection	$O(n^2)$	$O(n^2)$
Bubble	$O(n^2)$	$O(n^2)$
Quick	$O(n^2)$	$O(n\log_2 n)$
Heap	$O(n\log_2 n)$	$O(n\log_2 n)$
Merge	$O(n\log_2 n)$	$O(n\log_2 n)$