



Struktur Data

Pendidikan Teknik Informatika

Universitas Negeri Padang

TIM DOSEN ©2022

JOBSHEET (JS-11)

Sorting (2)

| Fakultas | Proram Studi | Kode MK | Waktu |
|----------|-------------------------------|--------------|--------------|
| Teknik | Pendidikan Teknik Informatika | TIK1.61.2317 | 4 x 50 Menit |

TUJUAN PRAKTIKUM

1. Mahasiswa mampu mengaplikasikan konsep Merge Sort dalam menyelesaikan kasus.
2. Mahasiswa mampu mengaplikasikan konsep Quick Sort dalam menyelesaikan kasus.
3. Mahasiswa mampu mengaplikasikan konsep Shell Sort dalam menyelesaikan kasus.

HARDWARE & SOFTWARE

1. Personal Computer
2. IDE: Dev C++

TEORI SINGKAT

A. Merge Sort

Merge sort adalah teknik pengurutan yang mengikuti pendekatan membagi dan menaklukkan. Artikel ini akan sangat membantu dan menarik bagi siswa karena mereka mungkin menghadapi semacam pertanyaan dalam ujian mereka. Dalam pengkodean atau wawancara teknis untuk insinyur perangkat lunak, algoritma pengurutan banyak ditanyakan. Jadi, penting untuk membahas topik.

Merge sort mirip dengan algoritma quick sort karena menggunakan pendekatan bagi dan taklukkan untuk mengurutkan elemen. Ini adalah salah satu algoritma pengurutan yang paling populer dan efisien. Ini membagi daftar yang diberikan menjadi dua bagian yang sama, memanggil dirinya sendiri untuk dua bagian dan kemudian

menggabungkan dua bagian yang diurutkan. Kita harus mendefinisikan fungsi **merge()** untuk melakukan penggabungan.

Sub-daftar dibagi lagi dan lagi menjadi dua sampai daftar tidak dapat dibagi lagi. Kemudian kami menggabungkan pasangan dari daftar satu elemen ke dalam daftar dua elemen, mengurutkannya dalam proses. Pasangan dua elemen yang diurutkan digabungkan ke dalam daftar empat elemen, dan seterusnya sampai kita mendapatkan daftar yang diurutkan.

Untuk memahami cara kerja algoritma merge sort, mari kita ambil array yang tidak disortir. Akan lebih mudah untuk memahami jenis gabungan melalui contoh.

Misalkan elemen array adalah :

| | | | | | | | |
|----|----|----|---|----|----|----|----|
| 12 | 31 | 25 | 8 | 32 | 17 | 40 | 42 |
|----|----|----|---|----|----|----|----|

Menurut jenis gabungan, pertama-tama bagilah array yang diberikan menjadi dua bagian yang sama. Merge sort terus membagi daftar menjadi bagian yang sama hingga tidak dapat dibagi lagi. Karena ada delapan elemen dalam larik yang diberikan, maka larik tersebut dibagi menjadi dua larik berukuran 4.

| | | | | | | | | |
|---------------|----|----|----|---|----|----|----|----|
| divide | 12 | 31 | 25 | 8 | 32 | 17 | 40 | 42 |
|---------------|----|----|----|---|----|----|----|----|

Sekarang, bagi lagi kedua array ini menjadi dua. Karena ukurannya 4, maka bagilah menjadi array baru berukuran 2.

| | | | | | | | | |
|---------------|----|----|----|---|----|----|----|----|
| divide | 12 | 31 | 25 | 8 | 32 | 17 | 40 | 42 |
|---------------|----|----|----|---|----|----|----|----|

Sekarang, bagi lagi array ini untuk mendapatkan nilai atom yang tidak dapat dibagi lagi.

| | | | | | | | | |
|---------------|----|----|----|---|----|----|----|----|
| divide | 12 | 31 | 25 | 8 | 32 | 17 | 40 | 42 |
|---------------|----|----|----|---|----|----|----|----|

Sekarang, gabungkan mereka dengan cara yang sama seperti saat mereka dibagi. Dalam menggabungkan, pertama-tama bandingkan elemen dari setiap larik dan kemudian gabungkan ke dalam larik lain dalam urutan yang diurutkan.

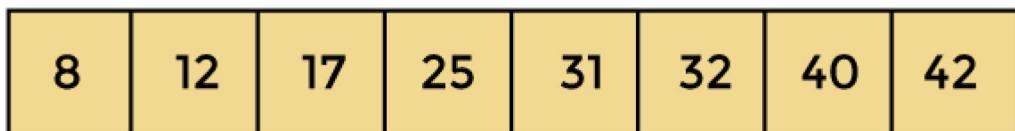
Jadi, bandingkan dulu 12 dan 31, keduanya dalam posisi terurut. Kemudian bandingkan 25 dan 8, dan dalam daftar dua nilai, letakkan 8 terlebih dahulu diikuti dengan 25. Kemudian bandingkan 32 dan 17, urutkan dan letakkan 17 terlebih dahulu diikuti dengan 32. Setelah itu, bandingkan 40 dan 42, dan tempatkan secara berurutan.



Dalam iterasi penggabungan berikutnya, sekarang bandingkan larik dengan dua nilai data dan gabungkan ke dalam larik nilai yang ditemukan dalam urutan yang diurutkan.



Sekarang, ada penggabungan terakhir dari array. Setelah penggabungan terakhir dari array di atas, array akan terlihat seperti :



Sekarang, array benar-benar diurutkan.

B. Quick Sort

Quicksort adalah algoritma pengurutan yang banyak digunakan yang membuat $n \log n$ perbandingan dalam kasus rata-rata untuk menyortir sebuah array dari n elemen. Ini adalah algoritma pengurutan yang lebih cepat dan sangat efisien. Algoritma ini mengikuti pendekatan membagi dan menaklukkan. Divide and Conquer adalah teknik memecah algoritma menjadi subproblem, kemudian menyelesaikan subproblem, dan menggabungkan kembali hasilnya untuk menyelesaikan masalah semula.

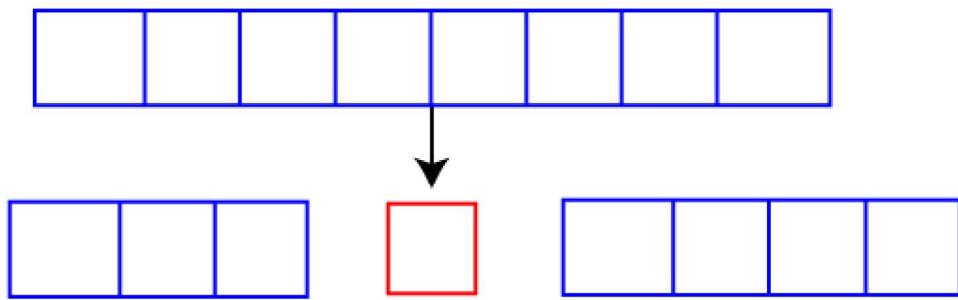
- Divide:** Di Divide, pertama-tama pilih elemen pivot. Setelah itu, partisi atau atur ulang array menjadi dua sub-array sehingga setiap elemen pada sub-array kiri lebih kecil atau sama dengan elemen pivot dan setiap elemen pada sub-array kanan lebih besar dari elemen pivot.
- Conquer:** Secara rekursif, urutkan dua subarray dengan Quicksort.

3. **Combine:** Menggabungkan array yang sudah diurutkan.

Quicksort memilih elemen sebagai pivot, lalu mempartisi array yang diberikan di sekitar elemen pivot yang dipilih. Dalam pengurutan cepat, larik besar dibagi menjadi dua larik di mana satu menyimpan nilai yang lebih kecil dari nilai yang ditentukan (Pivot), dan larik lain menyimpan nilai yang lebih besar dari pivot.

Setelah itu, sub-array kiri dan kanan juga dipartisi menggunakan pendekatan yang sama. Ini akan berlanjut sampai elemen tunggal tetap berada di sub-array.

Quick Sort



Pivot

Memilih pivot yang baik diperlukan untuk implementasi quicksort yang cepat. Namun, itu adalah tipikal untuk menentukan poros yang baik. Beberapa cara memilih pivot adalah sebagai berikut :

1. Pivot bisa acak, yaitu memilih pivot acak dari array yang diberikan.
2. Pivot bisa menjadi elemen paling kanan dari elemen paling kiri dari larik yang diberikan.
3. Pilih median sebagai elemen pivot.

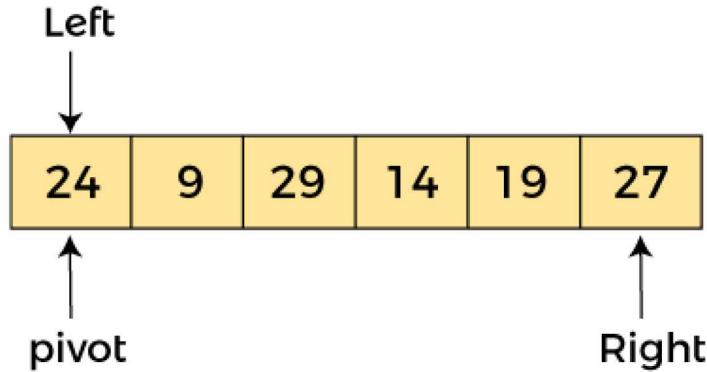
Untuk memahami cara kerja quick sort, mari kita ambil array yang tidak disortir. Ini akan membuat konsep lebih jelas dan mudah dipahami.

Misalkan elemen array adalah :

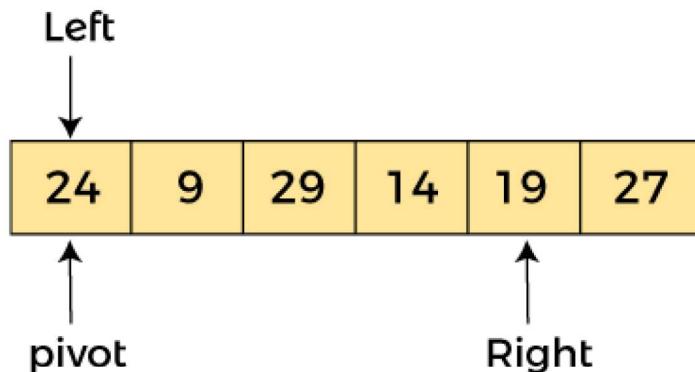
| | | | | | |
|----|---|----|----|----|----|
| 24 | 9 | 29 | 14 | 19 | 27 |
|----|---|----|----|----|----|

Dalam larik yang diberikan, kami menganggap elemen paling kiri sebagai pivot. Jadi, dalam hal ini, $a[\text{kiri}] = 24$, $a[\text{kanan}] = 27$ dan $a[\text{poros}] = 24$. Karena pivot

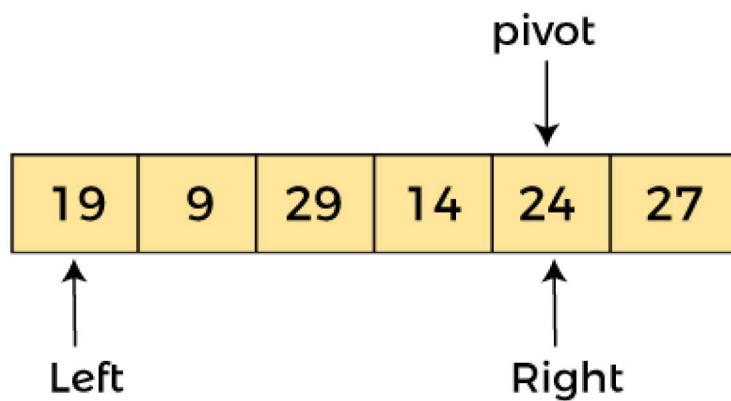
berada di kiri, maka algoritma dimulai dari kanan dan bergerak ke kiri.



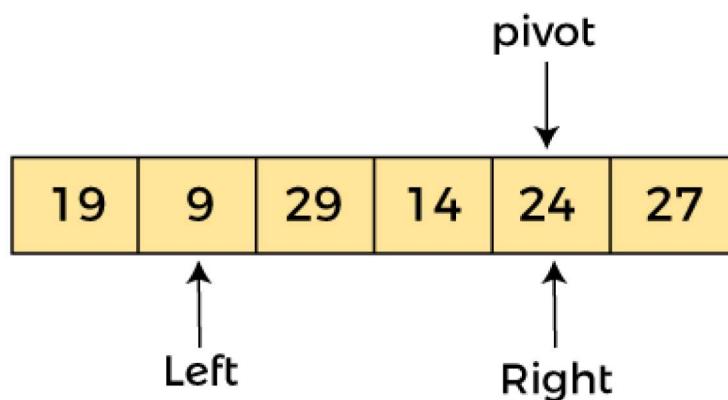
Sekarang, $a[\text{pivot}] < a[\text{kanan}]$, jadi algoritma bergerak maju satu posisi ke kiri, yaitu :



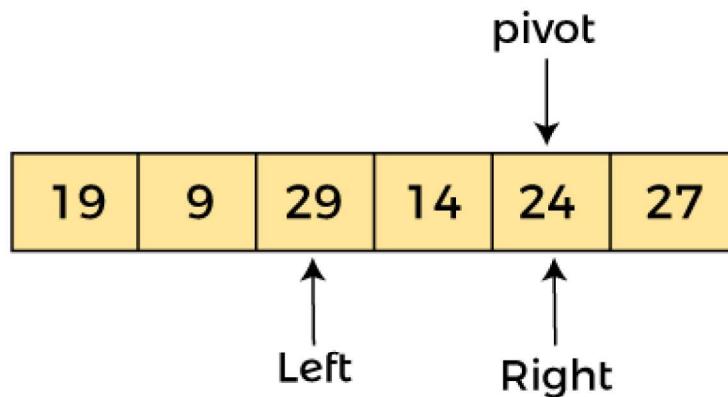
Sekarang, $a[\text{kiri}] = 24$, $a[\text{kanan}] = 19$, dan $a[\text{poros}] = 24$. Karena, $a[\text{pivot}] > a[\text{kanan}]$, maka algoritma akan menukar $a[\text{pivot}]$ dengan $a[\text{kanan}]$, dan pivot bergerak ke kanan, sebagai berikut :



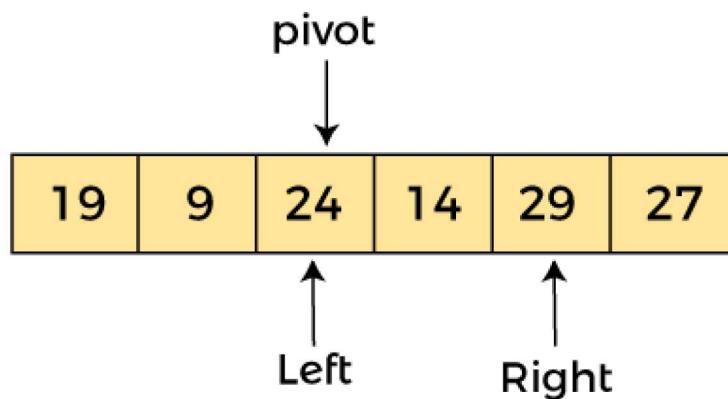
Sekarang, $a[\text{left}] = 19$, $a[\text{right}] = 24$, dan $a[\text{pivot}] = 24$. Karena pivot berada di kanan, maka algoritma dimulai dari kiri dan bergerak ke kanan. Sebagai $a[\text{pivot}] > a[\text{kanan}]$, maka algoritma bergerak satu posisi ke kanan sebagai -



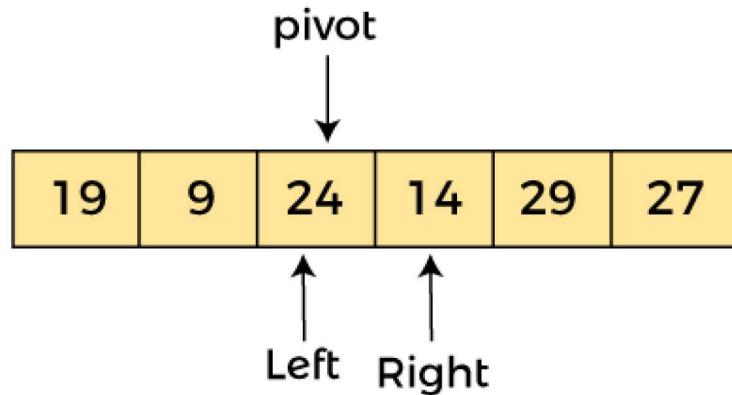
Sekarang, $a[\text{kiri}] = 9$, $a[\text{kanan}] = 24$, dan $a[\text{pivot}] = 24$. Sebagai $a[\text{pivot}] > a[\text{kiri}]$, maka algoritma bergerak satu posisi ke kanan sebagai berikut :



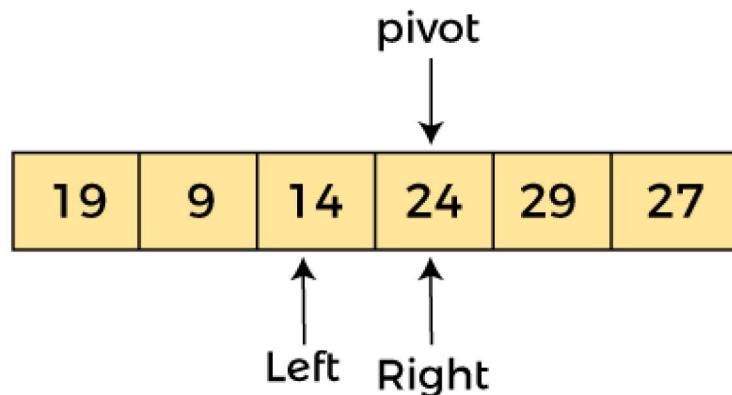
Sekarang, $a[\text{kiri}] = 29$, $a[\text{kanan}] = 24$, dan $a[\text{pivot}] = 24$. Sebagai $a[\text{pivot}] < a[\text{kiri}]$, jadi, tukar $a[\text{pivot}]$ dan $a[\text{kiri}]$, sekarang pivot ada di sebelah kiri, yaitu :



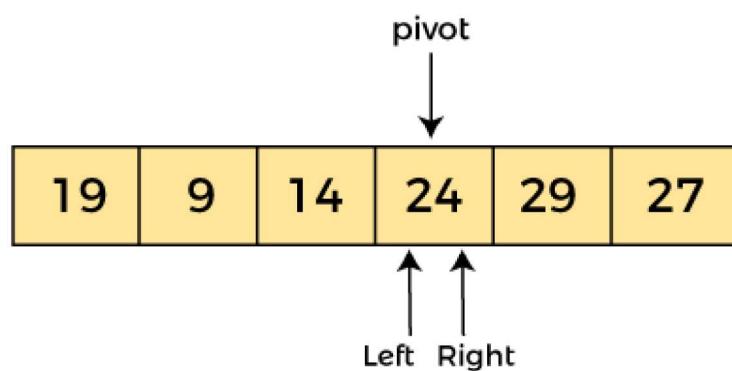
Karena pivot berada di kiri, maka algoritma dimulai dari kanan, dan bergerak ke kiri. Sekarang, $a[\text{kiri}] = 24$, $a[\text{kanan}] = 29$, dan $a[\text{pivot}] = 24$. Sebagai $a[\text{pivot}] < a[\text{kanan}]$, maka algoritma bergerak satu posisi ke kiri, sebagai berikut :



Sekarang, $a[\text{pivot}] = 24$, $a[\text{kiri}] = 24$, dan $a[\text{kanan}] = 14$. Sebagai $a[\text{pivot}] > a[\text{kanan}]$, jadi, tukar $a[\text{pivot}]$ dan $a[\text{kanan}]$, sekarang pivot di sebelah kanan, yaitu :

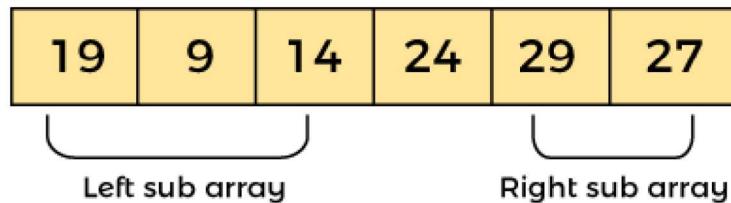


Sekarang, $a[\text{pivot}] = 24$, $a[\text{kiri}] = 14$, dan $a[\text{kanan}] = 24$. Pivot ada di kanan, jadi algoritme dimulai dari kiri dan bergerak ke kanan.

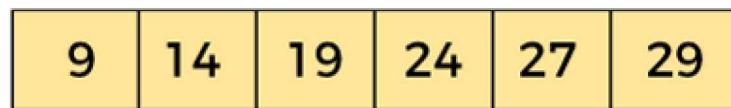


Sekarang, $a[\text{pivot}] = 24$, $a[\text{kiri}] = 24$, dan $a[\text{kanan}] = 24$. Jadi, pivot, kiri dan kanan menunjuk elemen yang sama. Ini mewakili penghentian prosedur. Elemen 24, yang merupakan elemen pivot ditempatkan pada posisi yang tepat. Unsur yang berada di

sebelah kanan unsur 24 lebih besar darinya, dan unsur yang berada di sebelah kiri unsur 24 lebih kecil darinya.



Sekarang, dengan cara yang sama, algoritma quick sort diterapkan secara terpisah ke sub-array kiri dan kanan. Setelah penyortiran selesai, array akan menjadi -



C. Shell Sort

Shell sort adalah generalisasi dari insertion sort, yang mengatasi kelemahan insertion sort dengan membandingkan elemen yang dipisahkan oleh gap beberapa posisi. Ini adalah algoritma pengurutan yang merupakan versi lanjutan dari insertion sort. Shell sort telah meningkatkan kompleksitas waktu rata-rata pengurutan penyisipan. Mirip dengan insertion sort, ini adalah algoritma sortir berbasis perbandingan dan di tempat. Shell sort efisien untuk kumpulan data berukuran sedang.

Dalam insertion sort, pada suatu waktu, elemen dapat dipindahkan ke depan hanya dengan satu posisi. Untuk memindahkan elemen ke posisi yang jauh, diperlukan banyak gerakan yang meningkatkan waktu eksekusi algoritme. Tapi shell sort mengatasi kelemahan insertion sort ini. Ini memungkinkan pergerakan dan pertukaran elemen yang jauh juga. Algoritma ini pertama-tama mengurutkan elemen yang berjauhan satu sama lain, kemudian mengurangi jarak di antara mereka. Kesenjangan ini disebut sebagai **interval**.

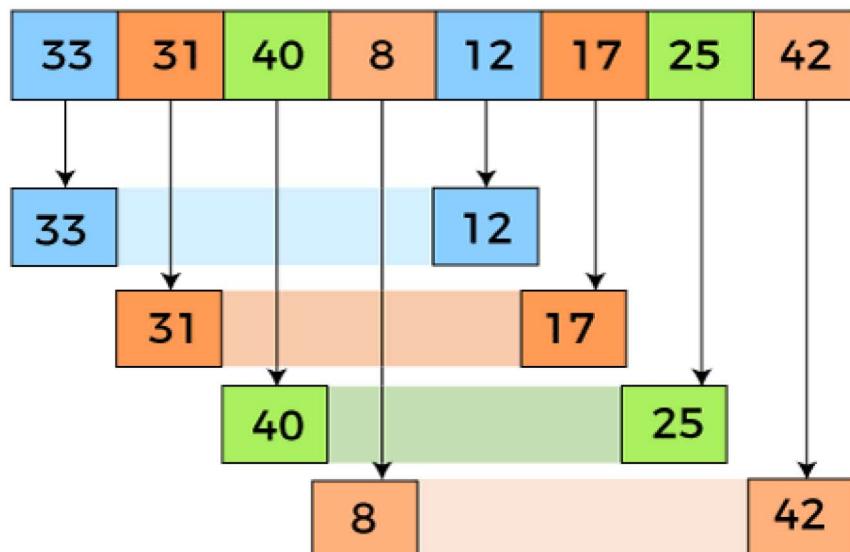
Untuk memahami cara kerja algoritma pengurutan shell, mari kita ambil array yang tidak disortir. Akan lebih mudah untuk memahami jenis shell melalui sebuah contoh.

Misalkan elemen array adalah :

| | | | | | | | |
|----|----|----|---|----|----|----|----|
| 33 | 31 | 40 | 8 | 12 | 17 | 25 | 42 |
|----|----|----|---|----|----|----|----|

Kita akan menggunakan urutan asli dari shell sort, yaitu, $N/2$, $N/4$, ..., 1 sebagai interval. Pada loop pertama, n sama dengan 8 (ukuran larik), jadi elemen-elemennya terletak pada interval 4 ($n/2 = 4$). Elemen akan dibandingkan dan ditukar jika tidak berurutan. Di sini, pada loop pertama, elemen pada posisi ke-0 akan dibandingkan dengan elemen pada posisi ke- 4 . Jika elemen ke-0 lebih besar, maka akan ditukar dengan elemen pada posisi ke- 4 . Jika tidak, itu tetap sama. Proses ini akan berlanjut untuk elemen yang tersisa.

Pada interval 4, sublist adalah {33, 12}, {31, 17}, {40, 25}, {8, 42}.

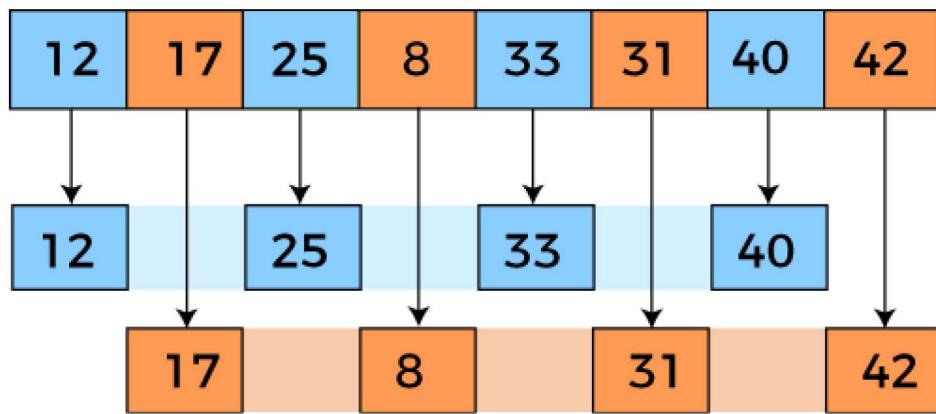


Sekarang, kita harus membandingkan nilai di setiap sub-daftar. Setelah membandingkan, kita harus menukarnya jika diperlukan dalam array asli. Setelah membandingkan dan menukar, array yang diperbarui akan terlihat sebagai berikut :

| | | | | | | | |
|----|----|----|---|----|----|----|----|
| 12 | 17 | 25 | 8 | 33 | 31 | 40 | 42 |
|----|----|----|---|----|----|----|----|

Pada loop kedua, elemen-elemen terletak pada interval 2 ($n/4 = 2$), di mana $n = 8$.

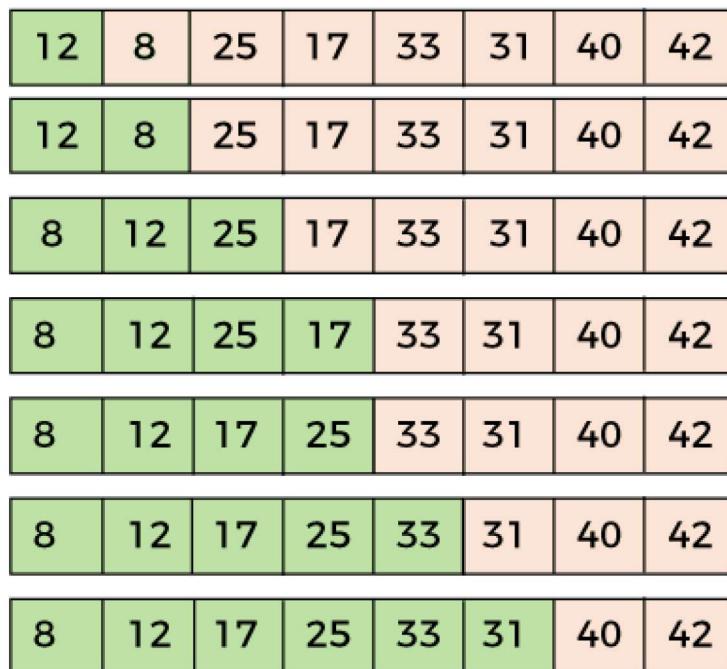
Sekarang, kita mengambil interval 2 untuk mengurutkan sisa array. Dengan interval 2, dua sublist akan dihasilkan - {12, 25, 33, 40}, dan {17, 8, 31, 42}.

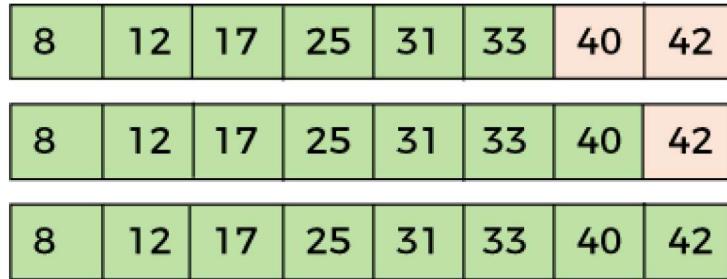


Sekarang, kita kembali harus membandingkan nilai di setiap sub-daftar. Setelah membandingkan, kita harus menukarnya jika diperlukan dalam array asli. Setelah membandingkan dan menukar, array yang diperbarui akan terlihat sebagai berikut :



Pada loop ketiga, elemen terletak pada interval 1 ($n/8 = 1$), di mana $n = 8$. Akhirnya, kita menggunakan interval nilai 1 untuk mengurutkan elemen array lainnya. Pada langkah ini, shell sort menggunakan insertion sort untuk mengurutkan elemen array.





Sekarang, array diurutkan dalam urutan menaik.

LATIHAN

1. Merge Sort secara ascending

```
/* Nama File : merge sort ascending
Pembuat      : tuliskan nama dan NIM anda
Tgl pembuatan : tuliskan tanggal hari ini*/

// Merge Sort secara ascending

#include <stdio.h>

/* Function to merge the subarrays of a[] */
void merge(int a[], int beg, int mid, int end)
{
    int i, j, k;
    int n1 = mid - beg + 1;
    int n2 = end - mid;

    int LeftArray[n1], RightArray[n2]; //temporary arrays

    /* copy data to temp arrays */
    for (int i = 0; i < n1; i++)
        LeftArray[i] = a[beg + i];
    for (int j = 0; j < n2; j++)
        RightArray[j] = a[mid + 1 + j];
```

```

i = 0; /* initial index of first sub-array */
j = 0; /* initial index of second sub-array */
k = beg; /* initial index of merged sub-array */

while (i < n1 && j < n2)
{
    if(LeftArray[i] <= RightArray[j])
    {
        a[k] = LeftArray[i];
        i++;
    }
    else
    {
        a[k] = RightArray[j];
        j++;
    }
    k++;
}

while (i<n1)
{
    a[k] = LeftArray[i];
    i++;
    k++;
}

while (j<n2)
{
    a[k] = RightArray[j];
    j++;
    k++;
}

void mergeSort(int a[], int beg, int end)
{

```

```
if (beg < end)
{
    int mid = (beg + end) / 2;
    mergeSort(a, beg, mid);
    mergeSort(a, mid + 1, end);
    merge(a, beg, mid, end);
}

/* Function to print the array */
void printArray(int a[], int n)
{
    int i;
    for (i = 0; i < n; i++)
        printf("%d ", a[i]);
    printf("\n");
}

int main()
{
    int a[] = { 12, 31, 25, 8, 32, 17, 40, 42 };
    int n = sizeof(a) / sizeof(a[0]);
    printf("Before sorting array elements are - \n");
    printArray(a, n);
    mergeSort(a, 0, n - 1);
    printf("After sorting array elements are - \n");
    printArray(a, n);
    return 0;
}
```

2. Quick Sort secara ascending

```
/* Nama File : quick sort ascending
Pembuat : tuliskan nama dan NIM anda
Tgl pembuatan : tuliskan tanggal hari ini*/
```

```

// Quick Sort secara ascending

#include <stdio.h>
/* function that consider last element as pivot,
place the pivot at its exact position, and place
smaller elements to left of pivot and greater
elements to right of pivot. */
int partition (int a[], int start, int end)
{
    int pivot = a[end]; // pivot element
    int i = (start - 1);

    for (int j = start; j <= end - 1; j++)
    {
        // If current element is smaller than the pivot
        if (a[j] < pivot)
        {
            i++; // increment index of smaller element
            int t = a[i];
            a[i] = a[j];
            a[j] = t;
        }
    }
    int t = a[i+1];
    a[i+1] = a[end];
    a[end] = t;
    return (i + 1);
}

/* function to implement quick sort */
void quick(int a[], int start, int end) /* a[] = array to
be sorted, start = Starting index, end = Ending index */
{
    if (start < end)
    {
        int p = partition(a, start, end); //p is the

```

```
partitioning index
    quick(a, start, p - 1);
    quick(a, p + 1, end);
}
}

/* function to print an array */
void printArr(int a[], int n)
{
    int i;
    for (i = 0; i < n; i++)
        printf("%d ", a[i]);
}
int main()
{
    int a[] = { 24, 9, 29, 14, 19, 27 };
    int n = sizeof(a) / sizeof(a[0]);
    printf("Before sorting array elements are - \n");
    printArr(a, n);
    quick(a, 0, n - 1);
    printf("\nAfter sorting array elements are - \n");
    printArr(a, n);

    return 0;
}
```

3. Shell Sort secara ascending

```
/* Nama File : shell sort ascending
Pembuat : tuliskan nama dan NIM anda
Tgl pembuatan : tuliskan tanggal hari ini*/
```

```
// Shell Sort secara ascending

#include <stdio.h>

/* function to implement shellSort */
```

```

int shell(int a[], int n)
{
    /* Rearrange the array elements at n/2, n/4, ..., 1
intervals */
    for (int interval = n/2; interval > 0; interval /= 2)
    {
        for (int i = interval; i < n; i += 1)
        {
            /* store a[i] to the variable temp and make the
ith position empty */
            int temp = a[i];
            int j;
            for (j = i; j >= interval && a[j - interval] >
temp; j -= interval)
                a[j] = a[j - interval];

            // put temp (the original a[i]) in its correct
position
            a[j] = temp;
        }
    }
    return 0;
}

void printArr(int a[], int n) /* function to print the array
elements */
{
    int i;
    for (i = 0; i < n; i++)
        printf("%d ", a[i]);
}

int main()
{
    int a[] = { 33, 31, 40, 8, 12, 17, 25, 42 };
    int n = sizeof(a) / sizeof(a[0]);
    printf("Before sorting array elements are - \n");
    printArr(a, n);
}

```

```
    shell(a, n);
    printf("\nAfter applying shell sort, the array elements
are - \n");
    printArr(a, n);
    return 0;
}
```

TUGAS

1. Pahami dan pelajari program Bubble Sort pada Latihan, setelah itu editlah program tersebut sehingga menjadi program **Merge Sort secara descending**.
2. Pahami dan pelajari program Insertion Sort pada Latihan, setelah itu editlah program tersebut sehingga menjadi program **Quick Sort secara descending**.
3. Pahami dan pelajari program Selection Sort pada Latihan, setelah itu editlah program tersebut sehingga menjadi program **Shell Sort secara descending**.

DAFTAR PUSTAKA

- Kelley, Al and Pohl, Ira. 2003. *C by Dissection: The Essentials of C Programming*. Addison-Weasley.
- Khannedy, Eko Kurniawan. 2007. *Diktat Pemrograman C*. Bandung : Unikom.
- Liem, Ingriani. 2003. *Catatan Singkat Bahasa C*. Jurusan Teknik Informatika. Institut Teknologi Bandung.
- Reema Thareja. 2014. *Data Structures Using C*. India : Oxford University Press.
- Solichin, Achmad. 2003. *Pemrograman Bahasa C dengan Turbo C*. Kuliah Berseri ilmukomputer.com