**LAPORAN PRAKTIKUM**

**SRUKTUR DATA**

**MODUL 2 - 3**

****

**DISUSUN OLEH :**

**MUHAMMAD NURIL ANWARI**

**362458302106**

**1E**

**TEKNOLOGI REKAYASA PERANGKAT LUNAK**

**POLITEKNIK NEGERI BANYUWANGI**

**TAHUN 2025**

**MODUL 2 ALGORITMA PENGURUTAN (INSERTION, SELECTION, BUBBLE DAN SHELL)**

**TUJUAN PRAKTIKUM :**

1. Memahami mengenai algoritma pengurutan insertion sort dan selection sort
2. Mampu mengimplementasikan algoritma pengurutan insertion sort dan selection sort secara ascending dan descending
3. Memahami mengenai algoritma pengurutan bubble sort dan shell sort
4. Mampu mengimplementasikan algoritma pengurutan bubble sort dan shell sort secara ascending dan descending.

**TEORI SINGKAT :**

**Algoritma Pengurutan** Algoritma pengurutan adalah metode untuk menyusun data dalam urutan tertentu, seperti dari kecil ke besar (ascending) atau sebaliknya (descending). Beberapa algoritma pengurutan yang umum digunakan adalah Insertion Sort, Selection Sort, Bubble Sort, dan Shell Sort.

* Insertion Sort bekerja dengan cara menyisipkan setiap elemen ke posisi yang benar dalam daftar yang sudah diurutkan. Algoritma ini cocok untuk data yang hampir terurut tetapi kurang efisien untuk data dalam jumlah besar.
* Selection Sort mencari elemen terkecil dalam daftar dan menukarnya dengan elemen di posisi awal. Proses ini terus berulang hingga seluruh daftar terurut. Meskipun sederhana, algoritma ini kurang efisien karena harus mencari elemen terkecil di setiap langkahnya.
* Bubble Sort membandingkan dan menukar elemen yang bersebelahan jika urutannya salah. Proses ini dilakukan berulang kali hingga tidak ada lagi pertukaran yang diperlukan. Bubble Sort mudah dipahami tetapi lambat karena banyaknya pertukaran.
* Shell Sort adalah pengembangan dari Insertion Sort yang membandingkan elemen dengan jarak tertentu sebelum menggunakan metode penyisipan. Ini membuat pengurutan lebih cepat dibandingkan Insertion Sort, terutama untuk jumlah data yang besar. Setiap algoritma memiliki kelebihan dan kekurangan, dan pemilihannya tergantung pada ukuran serta kondisi data yang akan diurutkan.

**TUGAS PENDAHULUAN :**

**1. Tuliskan algoritma pengurutan insertion sort secara ascending algoritma insertion sort:**

1. Mulai dari angka kedua dalam daftar (karena angka pertama dianggap sudah terurut).
2. Bandingkan angka ini dengan angka sebelumnya (yang ada di sebelah kiri).
3. Jika angka sebelumnya lebih besar, geser angka itu ke kanan untuk memberi ruang.
4. Ulangi langkah ini sampai angka yang sedang diproses menemukan tempat yang tepat.
5. Lanjutkan ke angka berikutnya dan ulangi proses hingga semua angka tersusun rapi dari kecil ke besar.

**2. Tuliskan algoritma pengurutan selection sort secara ascending Algoritma Selection Sort:**

1. Cari angka terkecil dalam daftar.
2. Tukar angka terkecil itu dengan angka di posisi pertama.
3. Cari angka terkecil kedua, tukar dengan angka di posisi kedua.
4. Ulangi proses ini sampai semua angka tersusun dari kecil ke besar.

**3. Tuliskan algoritma pengurutan bubble sort secara ascending Algoritma Bubble Sort:**

1. Bandingkan dua angka pertama. Jika angka pertama lebih besar, tukar posisinya.
2. Lanjutkan ke pasangan angka berikutnya, ulangi perbandingan dan tukar jika perlu.
3. Ulangi langkah ini hingga angka terbesar "menggelembung" ke posisi terakhir.
4. Lakukan proses ini lagi untuk sisa angka yang belum terurut hingga semua angka tersusun dengan benar.

**4. Tuliskan algoritma pengurutan shell sort secara ascending Algoritma Shell Sort:**

1. Pilih sebuah jarak yang cukup untuk membandingkan angka dalam daftar.
2. Bandingkan angka yang terpisah sejauh jarak tadi, tukar jika tidak berurutan.
3. Kurangi jarak dan ulangi proses sampai jaraknya menjadi 1 (seperti Insertion Sort)
4. Setelah jarak = 1, lakukan penyortiran terakhir hingga semua angka berurutan dengan benar.

**PERCOBAAN :**

**Percobaan 1 : Insertion Sort dengan cara Ascending**

void insertionSort<T extends Comparable<T>>(List<T> arr) {

for (int i = arr.length - 1; i >= 0; i--) {

for (int j = i + 1, k = i; j < arr.length; j++) {

if (arr[j].compareTo(arr[k]) > 0) {

break;

} else {

T temp = arr[k];

arr[k] = arr[j];

arr[j] = temp;

k = j;

}

}

}

}

void display<T>(List<T> data) {

for (T obj in data) {

print('$obj ');

}

print('');

}

void main() {

List<num> data = List<num>.generate(10, (index)=>(index + 1) \* 2);

data.shuffle();

print('Data Sebelum Pengurutan:');

display(data);

DateTime start = DateTime.now();

insertionSort<num>(data);

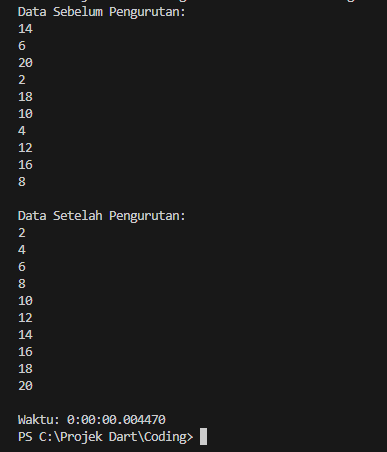
Duration elapsedTime = DateTime.now().difference(start);

print('Data Setelah Pengurutan:');

display(data);

print('Waktu: $elapsedTime');

}**Output :**

****

**Percobaan 2 : Selection Sort secara Ascending**

void selectionSort<T extends Comparable<T>>(List<T> arr) {

T temp;

for (int i = arr.length - 1; i >= 0; i--) {

int max = i;

for (int j = i - 1; j >= 0; j--) {

if (arr[j].compareTo(arr[max]) > 0) max = j;

}

temp = arr[i];

arr[i] = arr[max];

arr[max] = temp;

}

}

void display<T>(List<T> data) {

for (T objek in data) {

print('$objek ');

}

print('');

}

void main() {

List<num> data = List<num>.generate(10, (index)=>(index + 1) \* 2);

data.shuffle();

print('Data Sebelum Pengurutan:');

display(data);

DateTime start = DateTime.now();

selectionSort(data);

Duration elapsedTime = DateTime.now().difference(start);

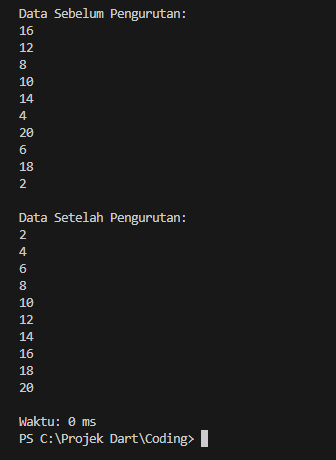
print('Data Setelah Pengurutan:');

display(data);

print('Waktu: ${elapsedTime.inMilliseconds} ms');

}

**Output :**

****

**Percobaan 3 :**

void bubbleSort<T extends Comparable<T>>(List<T> arr) {

int i, j;

T temp;

for (i = 0; i < arr.length - 1; i++) {

for (j = 0; j < arr.length - i - 1; j++) {

if (arr[j].compareTo(arr[j + 1]) > 0) {

temp = arr[j + 1];

arr[j + 1] = arr[j];

arr[j] = temp;

}

}

}

}

void display<T>(List<T> data) {

for (T objek in data) {

print('$objek ');

}

print('');

}

void main() {

List<num> data = List<num>.generate(10, (index)=>(index + 1) \* 2);

data.shuffle();

print('Data Sebelum Pengurutan:');

display(data);

DateTime start = DateTime.now();

bubbleSort(data);

Duration elapsedTime = DateTime.now().difference(start);

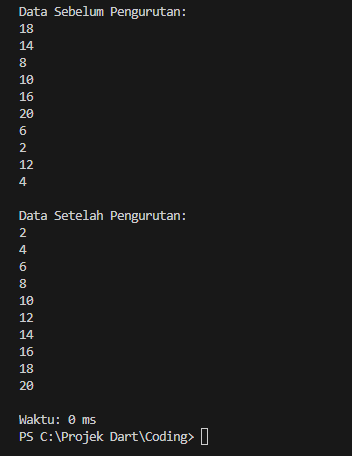
print('Data Setelah Pengurutan:');

display(data);

print('Waktu: ${elapsedTime.inMilliseconds} ms');

}

**Output :**

****

**Percobaan 4 :**

void bubbleSortFlag<T extends Comparable<T>>(List<T> arr) {

int i = 0, j;

bool didSwap = true;

T temp;

while (i < arr.length - 1 && didSwap) {

didSwap = false;

for (j = 0; j < arr.length - i - 1; j++) {

if (arr[j].compareTo(arr[j + 1]) > 0) {

temp = arr[j + 1];

arr[j + 1] = arr[j];

arr[j] = temp;

didSwap = true;

}

}

i++;

}

}

void display<T>(List<T> data) {

for (T objek in data) {

print('$objek ');

}

print('');

}

void main() {

List<num> data = List<num>.generate(10, (index)=>(index + 1) \* 2);

data.shuffle();

print('Data Sebelum Pengurutan:');

display(data);

DateTime start = DateTime.now();

bubbleSortFlag(data);

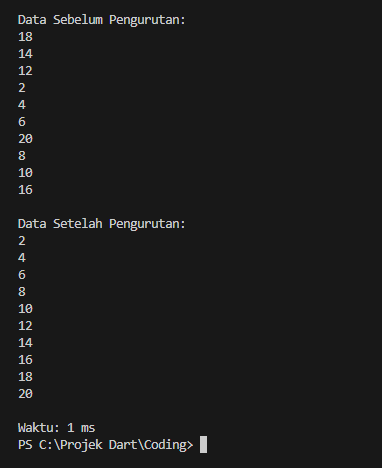
Duration elapsedTime = DateTime.now().difference(start);

print('Data Setelah Pengurutan:');

display(data);

print('Waktu: ${elapsedTime.inMilliseconds} ms');

}**Output :**

****

**Percobaan 5 :**

void shellSort<T extends Comparable<T>>(List<T> arr) {

int i, jarak;

bool didSwap = true;

T temp;

jarak = arr.length;

while (jarak > 1) {

jarak = (jarak / 2).floor();

didSwap = true;

while (didSwap) {

didSwap = false;

i = 0;

while (i < (arr.length - jarak)) {

if (arr[i].compareTo(arr[i + jarak]) > 0) {

temp = arr[i];

arr[i] = arr[i + jarak];

arr[i + jarak] = temp;

didSwap = true;

}

i++;

}

}

}

}

void display<T>(List<T> data) {

for (T objek in data) {

print('$objek ');

}

print('');

}

void main() {

List<num> data = List<num>.generate(10, (index)=>(index + 1) \* 2);

data.shuffle();

print('Data Sebelum Pengurutan:');

display(data);

DateTime start = DateTime.now();

shellSort(data);

Duration elapsedTime = DateTime.now().difference(start);

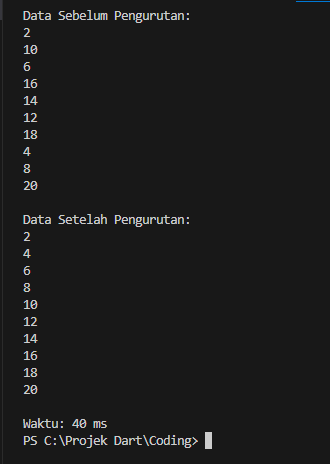
print('Data Setelah Pengurutan:');

display(data);

print('Waktu: ${elapsedTime.inMilliseconds} ms');

}

**Output :**

****

**TUGAS**

**Latihan 4 :**

void selectionSortAscending(List<int> arr) {

int n = arr.length;

for (int i = n - 1; i > 0; i--) {

int maxIdx = i;

for (int j = i - 1; j >= 0; j--) {

if (arr[j] < arr[maxIdx]) {

maxIdx = j;

}

}

// Tukar elemen

int temp = arr[i];

arr[i] = arr[maxIdx];

arr[maxIdx] = temp;

}

}

void selectionSortDescending(List<int> arr) {

int n = arr.length;

for (int i = n - 1; i > 0; i--) {

int minIdx = i;

for (int j = i - 1; j >= 0; j--) {

if (arr[j] > arr[minIdx]) {

minIdx = j;

}

}

// Tukar elemen

int temp = arr[i];

arr[i] = arr[minIdx];

arr[minIdx] = temp;

}

}

void main() {

List<int> arrAscending = [64, 25, 12, 22, 11];

List<int> arrDescending = [64, 25, 12, 22, 11];

print("Original array: $arrAscending");

// Pengurutan Ascending

selectionSortAscending(arrAscending);

print("Ascending Sort: $arrAscending");

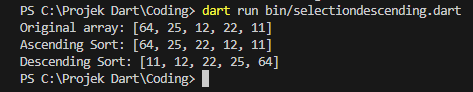
// Pengurutan Descending

selectionSortDescending(arrDescending);

print("Descending Sort: $arrDescending");

}

**Output :**

****

**Penjelasan:**

1. **selectionSortAscending:**
   * Fungsi ini mengurutkan array dalam urutan menaik (ascending) menggunakan Selection Sort.
   * Pengurutan dimulai dari sisi kanan dan mencari elemen terkecil dari bagian kiri array untuk kemudian dipindahkan ke posisi yang tepat.
2. **selectionSortDescending:**
   * Fungsi ini mengurutkan array dalam urutan menurun (descending) menggunakan Selection Sort.
   * Pengurutan dimulai dari sisi kanan dan mencari elemen terbesar dari bagian kiri array untuk kemudian dipindahkan ke posisi yang tepat.

**MODUL 3 ALGORITMA PENGURUTAN (QUICK DAN MERGE)**

**TUJUAN PRAKTIKUM :**

**Algoritma Pengurutan** (Quick Sort dan Merge Sort)

* 1. Memahami konsep dasar dari algoritma Quick Sort dan Merge Sort dalam pengurutan data.
  2. Menganalisis perbedaan kinerja antara Quick Sort dan Merge Sort dalam berbagai kasus (Best Case, Worst Case, dan Average Case).
  3. Mengimplementasikan algoritma Quick Sort dan Merge Sort dalam kode program untuk memahami cara kerja rekursi dan pembagian data.
  4. Membandingkan efisiensi algoritma berdasarkan kompleksitas waktu (Time Complexity) dan kompleksitas ruang (Space Complexity).
  5. Mengetahui kelebihan dan kekurangan dari masing-masing algoritma dalam berbagai kondisi data (data acak, data terurut, data terbalik).
  6. Meningkatkan pemahaman tentang rekursi dalam algoritma sorting, terutama dalam Merge Sort dan Quick Sort.

**TEORI SINGKAT :**

**Quick Sort**

**Quick Sort** adalah algoritma pengurutan yang bekerja dengan membagi data menjadi dua bagian berdasarkan pivot (elemen acuan).

**Langkah-langkahnya:**

1. Pilih satu elemen sebagai pivot (bisa elemen pertama, terakhir, atau tengah).
2. Pisahkan data menjadi dua bagian:
   * Kiri: elemen yang lebih kecil dari pivot.
   * Kanan: elemen yang lebih besar dari pivot.
3. Rekursif: Ulangi langkah yang sama untuk bagian kiri dan kanan sampai semua elemen terurut.

**Kelebihan:**

1. Cepat dalam rata-rata kasus (O(n log n))

2. Efisien untuk data besar

3. Tidak membutuhkan banyak memori tambahan

**Kekurangan:**

1. Bisa jadi lambat di worst case (O(n²)) jika pivot dipilih kurang baik

2. Tidak stabil (urutan elemen yang sama bisa berubah)

**Merge Sort**

**Merge Sort** adalah algoritma yang menggunakan divide and conquer dengan cara membagi data menjadi bagian kecil, lalu menggabungkannya kembali dalam keadaan terurut.

**Langkah-langkahnya:**

1. Bagi array menjadi dua bagian hingga tersisa satu elemen.

2. Urutkan setiap bagian kecil.

3. Gabungkan kembali (merge) dua bagian dengan cara membandingkan elemen satu per satu

**Kelebihan:**

1. Selalu memiliki performa O(n log n), baik dalam kasus terbaik maupun terburuk

2. Stabil (urutan elemen yang sama tetap dipertahankan)

**Kekurangan:**

1. Membutuhkan memori tambahan untuk proses penggabungan

2. Lebih lambat dibanding Quick Sort dalam beberapa kasus

**TUGAS PENDAHULUAN :**

* 1. **Tuliskan algoritma pengurutan quick sort secara ascending.**

**Apa itu Quick Sort?**

Quick Sort adalah cara mengurutkan angka dengan memilih satu angka sebagai patokan (pivot) lalu memindahkan angka-angka lain ke kiri atau kanan pivot sesuai ukurannya. Setelah itu, proses diulang pada bagian kiri dan kanan sampai semuanya rapi.

**Langkah-Langkah Algoritma Quick Sort (Ascending)**

1. Pilih pivot → Bisa elemen pertama, terakhir, atau tengah.

2. Partisi array → Pisahkan elemen yang lebih kecil ke kiri dan lebih besar ke kanan pivot.

3. Rekursif → Ulangi proses yang sama pada bagian kiri dan kanan hingga seluruh elemen terurut

* 1. **Tuliskan algoritma pengurutan merge sort secara ascending.**

Algoritma Merge Sort (Ascending Order) Merge Sort adalah algoritma pengurutan berbasis pembagian (divide and conquer). Artinya, kita akan membagi array menjadi bagian kecil, mengurutkan masing-masing bagian, lalu menggabungkannya kembali dalam urutan yang benar.

**Langkah-Langkah Merge Sort (Ascending).**

1. Bagi array menjadi dua bagian hingga tiap bagian hanya berisi satu elemen.

2. Urutkan tiap bagian secara rekursif.

3. Gabungkan dua bagian yang sudah diurutkan menjadi satu array yang terurut.

**Percobaan 1 :**

void main() {

List<num> data = List.generate(10, (index) => (index + 1)..toDouble());

printData(data);

DateTime startTime = DateTime.now();

quickSort(data, 0, data.length - 1);

Duration elapsedTime = DateTime.now().difference(startTime);

printData(data);

print('Waktu ${elapsedTime.inMilliseconds}');

}

void quickSort<T extends Comparable<T>>(List<T> arr, int p, int r) {

int q;

if (p < r) {

q = partition(arr, p, r);

quickSort(arr, p, q);

quickSort(arr, q + 1, r);

}

}

int partition<T extends Comparable<T>>(List<T> arr, int p, int r) {

int i, j;

T pivot, temp;

pivot = arr[p];

i = p;

j = r;

while (i <= j) {

while (pivot.compareTo(arr[j]) < 0) j--;

while (pivot.compareTo(arr[i]) > 0) i++;

if (i < j) {

temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

j--;

i++;

} else {

return j;

}

}

return j;

}

void printData<T>(List<T> data) {

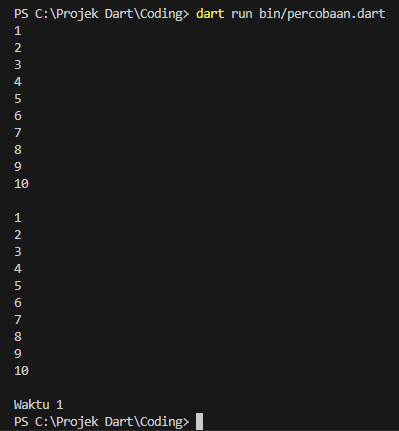
for (T objek in data) {

print('$objek ');

}

print('');

}**Output :**

****

**Percobaan 2 :**

void main() {

List<num> data = List.generate(10, (index) => (index + 1)..toDouble());

printData(data);

DateTime startTime = DateTime.now();

mergeSort(data, 0, data.length - 1);

Duration elapsedTime = DateTime.now().difference(startTime);

printData(data);

print('\nwaktu : ${elapsedTime.inMilliseconds}');

}

void mergeSort<T extends Comparable<T>>(List<T> arr, int l, int r) {

int med;

if (l < r) {

med = (l + r) ~/ 2;

mergeSort(arr, l, med);

mergeSort(arr, med + 1, r);

merge(arr, l, med, r);

}

}

void merge<T extends Comparable<T>>(

List<T> arr, int left, int median, int right) {

List<T?> temp = List.filled(arr.length, null);

int kiri1, kanan1, kiri2, kanan2, i, j;

kiri1 = left;

kanan1 = median;

kiri2 = median + 1;

kanan2 = right;

i = left;

while ((kiri1 <= kanan1) && (kiri2 <= kanan2)) {

if (arr[kiri1].compareTo(arr[kiri2]) <= 0) {

temp[i] = arr[kiri1];

kiri1++;

} else {

temp[i] = arr[kiri2];

kiri2++;

}

i++;

}

while (kiri1 <= kanan1) {

temp[i] = arr[kiri1];

kiri1++;

i++;

}

while (kiri2 <= kanan2) {

temp[i] = arr[kiri2];

kiri2++;

i++;

}

j = left;

while (j <= right) {

arr[j] = temp[j]!;

j++;

}

}

void printData<T>(List<T> data) {

for (T objek in data) {

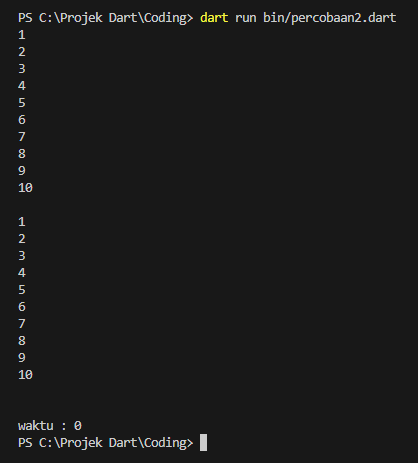
print('$objek ');

}

print('');

}

**Output :**

****

**Latihan :**

**Latihan 1 :** Dari percobaan 1 tambahkan method untuk melakukan pengurutan quick sort secara descending

void main() {

List<num> data = List.generate(10, (index) => (index + 1).toDouble());

print('Data awal:');

printData(data);

DateTime startTimeAsc = DateTime.now();

quickSort(data, 0, data.length - 1);

Duration elapsedTimeAsc = DateTime.now().difference(startTimeAsc);

print('Data setelah Quick Sort Ascending:');

printData(data);

print('Waktu Ascending: ${elapsedTimeAsc.inMilliseconds} ms\n');

List<num> dataDescending = List.from(data);

DateTime startTimeDesc = DateTime.now();

quickSortDescending(dataDescending, 0, dataDescending.length - 1);

Duration elapsedTimeDesc = DateTime.now().difference(startTimeDesc);

print('Data setelah Quick Sort Descending:');

printData(dataDescending);

print('Waktu Descending: ${elapsedTimeDesc.inMilliseconds} ms');

}

void quickSort<T extends Comparable<T>>(List<T> arr, int p, int r) {

if (p < r) {

int q = partition(arr, p, r);

quickSort(arr, p, q);

quickSort(arr, q + 1, r);

}

}

void quickSortDescending<T extends Comparable<T>>(List<T> arr, int p, int r) {

if (p < r) {

int q = partitionDescending(arr, p, r);

quickSortDescending(arr, p, q);

quickSortDescending(arr, q + 1, r);

}

}

int partition<T extends Comparable<T>>(List<T> arr, int p, int r) {

int i = p, j = r;

T pivot = arr[p];

while (i <= j) {

while (pivot.compareTo(arr[j]) < 0) j--;

while (pivot.compareTo(arr[i]) > 0) i++;

if (i < j) {

T temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

j--;

i++;

} else {

return j;

}

}

return j;

}

int partitionDescending<T extends Comparable<T>>(List<T> arr, int p, int r) {

int i = p, j = r;

T pivot = arr[p];

while (i <= j) {

while (pivot.compareTo(arr[j]) > 0) j--;

while (pivot.compareTo(arr[i]) < 0) i++;

if (i < j) {

T temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

j--;

i++;

} else {

return j;

}

}

return j;

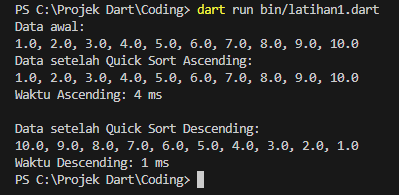
}

void printData<T>(List<T> data) {

print(data.join(', '));

}

**Output :**

****

**Kesimpulan :**

**Penjelasan:**

**1.Bagian main()**

* Membuat daftar angka dari 1.0 sampai 10.0 menggunakan List.generate().
* Menampilkan data awal sebelum diurutkan.

**2. Mengurutkan Secara Ascending**

* Menyimpan waktu mulai eksekusi sorting menggunakan DateTime.now().
* Memanggil quickSort() untuk mengurutkan data dari kecil ke besar.
* Menghitung waktu eksekusi sorting dengan difference(startTimeAsc).
* Menampilkan hasil sorting beserta waktu eksekusi.

**3.Mengurutkan Secara Descending**

* Menyalin daftar angka yang sudah diurutkan ke dataDescending.
* Memanggil quickSortDescending() untuk mengurutkan data dari besar ke kecil.
* Menghitung dan menampilkan waktu eksekusi.

**4. Fungsi quickSort()**

* Fungsi ini memecah daftar angka menjadi dua bagian berdasarkan pivot (elemen
* patokan).
* partition() memilih pivot dan menyusun angka yang lebih kecil di kiri dan lebih besar di
* kanan.
* Memanggil dirinya sendiri (rekursi) untuk mengurutkan bagian kiri dan kanan sampai

selesai.

**5.Fungsi partition()**

* Memilih pivot sebagai elemen pertama dari daftar.
* Membandingkan dan menukar angka supaya angka kecil di kiri, besar di kanan.
* Mengembalikan posisi terakhir pivot setelah pemisahan selesai.

**6.Fungsi quickSortDescending()**

* Sama seperti quickSort(), tetapi menggunakan partitionDescending().

**7. Fungsi partitionDescending()**

* Sama seperti partition(), tetapi logikanya dibalik supaya angka besar ada di kiri, angka

kecil di kanan.

**8. Fungsi printData()**

* Menampilkan angka dalam format yang lebih rapi menggunakan join(', ').

**Kesimpulan :**

1. Program ini dapat mengurutkan angka secara ascending dan descending

menggunakan Quick Sort.

1. Menggunakan rekursi untuk membagi dan mengurutkan daftar angka.
2. Menggunakan fungsi partition dan partitionDescending untuk pemisahan data.
3. Menampilkan waktu eksekusi sorting untuk melihat seberapa cepat prosesnya.

**Kesimpulan Praktikum :**

Algoritma pengurutan memiliki berbagai metode dengan kelebihan dan kekurangannya masing-masing. **Insertion Sort** bagus untuk data kecil atau yang hampir terurut karena menyisipkan elemen pada posisi yang tepat. **Selection Sort** bekerja dengan mencari nilai terkecil dan menukarnya, tetapi kurang efisien. **Bubble Sort** menukar elemen berulang kali hingga data terurut, namun butuh banyak langkah. **Shell Sort** lebih cepat dari Insertion Sort karena membandingkan elemen dengan jarak tertentu sebelum menyusun secara berurutan. Secara umum, metode ini cocok untuk data kecil, tetapi untuk data besar lebih baik menggunakan algoritma yang lebih cepat seperti Merge Sort atau Quick Sort.

**Referensi :**

**Chat Gpt :** <https://chatgpt.com/>

**Algoritma pengurutan dalam pemrogrman :** <https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2008-2009/Makalah2008/Makalah0809-022.pdf>

**GitHub :**