Quick Sort, Merge Sort

Nama: Oscar Javier Abdullah

Kelas: Gt-A11

Percobaan 1

Input:

```
#include <iostream>
#include <cstdlib>
#include <ctime>

const int MAX = 10;
const int MaxStack = 10;

int Data[MAX];

// Function to swap two elements
void swap(int& a, int& b) {
    int temp = a;
    a = b;
    b = temp;
}

// Non-recursive Quick Sort function
void quickSortNonRecursive() {
    struct Stack {
        int left;
        int right;
    } stack[MaxStack];
    int top = 1;
    stack[1].left = 0;
    stack[1].right = MAX - 1;

    while (top!= 0) {
```

```
        int left = stack[top].left;
        int right = stack[top].right;
        top--;

        while (right > left) {
            int i = left;
            int j = right;
            int pivot = Data[(left + right) / 2];

            while (i <= j) {
                while (Data[i] < pivot)
                    i++;
                while (pivot < Data[j])
                    j--;
                if (i <= j) {
                    swap(Data[i], Data[j]);
                    i++;
                    j--;
                }
            }

            if (left < i) {
                top++;
                stack[top].left = i;
                stack[top].right = right;
            }
            right = j;
        }
    }
}

int main() {
```

```
    srand(static_cast<unsigned int>(time(0))); // use time(0) for
random seed


    std::cout << "DATA SEBELUM TERURUT" << std::endl;

    for (int i = 0; i < MAX; i++) {

        Data[i] = static_cast<int>(rand() / 1000.0) + 1;

        std::cout << "Data ke " << i << ": " << Data[i] <<
std::endl;

    }


    quickSortNonRecursive();


    std::cout << "\nDATA SETELAH TERURUT" << std::endl;

    for (int i = 0; i < MAX; i++) {

        std::cout << "Data ke " << i << ": " << Data[i] <<
std::endl;

    }


    return 0;

}
```

Output:

```
/tmp/TNPT15JiJQ.o
DATA SEBELUM TERURUT
Data ke 0: 1486957
Data ke 1: 772263
Data ke 2: 1721056
Data ke 3: 1457936
Data ke 4: 280929
Data ke 5: 1180153
Data ke 6: 1497758
Data ke 7: 1682563
Data ke 8: 193711
Data ke 9: 689738

DATA SETELAH TERURUT
Data ke 0: 193711
Data ke 1: 280929
Data ke 2: 689738
Data ke 3: 772263
Data ke 4: 1180153
Data ke 5: 1457936
Data ke 6: 1486957
Data ke 7: 1497758
Data ke 8: 1682563
Data ke 9: 1721056
```

Percobaan 2

Input:

```
#include <iostream>
#include <cstdlib>
#include <ctime>

const int MAX = 10;

int Data[MAX];

// Function to swap two elements
void swap(int& a, int& b) {
    int temp = a;
    a = b;
    b = temp;
}

// Recursive Quick Sort function
void quickSortRecursive(int left, int right) {
    if (left <= right) {
        int pivotIndex = (left + right) / 2;
        int pivot = Data[pivotIndex];
        int i = left;
        int j = right;

        while (i <= j) {
            while (Data[i] < pivot)
                i++;
            while (Data[j] > pivot)
                j--;
            if (i <= j) {
                swap(Data[i], Data[j]);
```

```
                i++;
                j--;
            }
        }

        if (left < j)
            quickSortRecursive(left, j);
        if (i < right)
            quickSortRecursive(i, right);
    }
}

int main() {
    srand(static_cast<unsigned int>(time(0))); // use time(0) for
random seed

    std::cout << "DATA SEBELUM TERURUT" << std::endl;
    for (int i = 0; i < MAX; i++) {
        Data[i] = static_cast<int>(rand() / 1000.0) + 1;
        std::cout << "Data ke " << i << ": " << Data[i] <<
std::endl;
    }

    quickSortRecursive(0, MAX - 1);

    std::cout << "\nDATA SETELAH TERURUT" << std::endl;
    for (int i = 0; i < MAX; i++) {
        std::cout << "Data ke " << i << ": " << Data[i] <<
std::endl;
    }

    return 0;
}
```

Output:

```
/tmp/F6E5DfD6eR.o
DATA SEBELUM TERURUT
Data ke 0: 467427
Data ke 1: 2137079
Data ke 2: 28063
Data ke 3: 599792
Data ke 4: 1784935
Data ke 5: 1904911
Data ke 6: 1577739
Data ke 7: 36326
Data ke 8: 1832885
Data ke 9: 530214

DATA SETELAH TERURUT
Data ke 0: 28063
Data ke 1: 36326
Data ke 2: 467427
Data ke 3: 530214
Data ke 4: 599792
Data ke 5: 1577739
Data ke 6: 1784935
Data ke 7: 1832885
Data ke 8: 1904911
Data ke 9: 2137079
```

Percobaan 3

Input:

```
#include <iostream>
#include <cstdlib>
#include <ctime>

const int MAX = 10;
int Data[MAX];
int temp[MAX];

// Function to merge two sorted subarrays
void merge(int left, int mid, int right) {
    int i, j, k;
    int n1 = mid - left + 1;
    int n2 = right - mid;

    // Copy data to temp arrays
    for (i = 0; i < n1; i++)
        temp[i] = Data[left + i];
```

```
    for (j = 0; j < n2; j++)
        temp[n1 + j] = Data[mid + 1 + j];

    // Merge the temp arrays
    i = 0;
    j = 0;
    k = left;
    while (i < n1 && j < n2) {
        if (temp[i] <= temp[j]) {
            Data[k] = temp[i];
            i++;
        } else {
            Data[k] = temp[j];
            j++;
        }
        k++;
    }

    // Copy the remaining elements
    while (i < n1) {
        Data[k] = temp[i];
        i++;
        k++;
    }

    while (j < n2) {
        Data[k] = temp[j];
        j++;
        k++;
    }
}
```

```
// Recursive Merge Sort function
void mergeSort(int left, int right) {
    if (left < right) {
        int mid = left + (right - left) / 2;
        mergeSort(left, mid);
        mergeSort(mid + 1, right);
        merge(left, mid, right);
    }
}

int main() {
    srand(static_cast<unsigned int>(time(0))); // use time(0) for
random seed

    std::cout << "DATA SEBELUM TERURUT : ";
    for (int i = 0; i < MAX; i++) {
        Data[i] = static_cast<int>(rand() / 1000.0) + 1;
        std::cout << Data[i] << " ";
    }

    mergeSort(0, MAX - 1);

    std::cout << "\n\nDATA SETELAH TERURUT : ";
    for (int i = 0; i < MAX; i++)
        std::cout << Data[i] << " ";
    std::cout << std::endl;

    return 0;
}
```

Output:

```
/tmp/OUENSmnjzT.o
DATA SEBELUM TERURUT : 273881 2013478 1563436 1544538 92112 1799500 1197216 95889 29362 2062021

DATA SETELAH TERURUT : 273881 273881 273881 273881 273881 273881 273881 273881 273881 273881
```

Latihan 1

1) Tambahkan kode program untuk menampilkan perubahan setiap iterasi dari proses pengurutan dengan quick sort dan merge sort.

Input:

```
#include <iostream>
#include <vector>

using namespace std;

void printArray(const vector<int>& arr) {
    for (int i : arr) {
        cout << i << " ";
    }
    cout << endl;
}

int partition(vector<int>& arr, int low, int high) {
    int pivot = arr[high];
    int i = (low - 1);

    for (int j = low; j <= high - 1; j++) {
        if (arr[j] < pivot) {
            i++;
            swap(arr[i], arr[j]);
        }
    }
    swap(arr[i + 1], arr[high]);
```

```
    return (i + 1);
}

void quickSort(vector<int>& arr, int low, int high) {
    if (low < high) {
        int pi = partition(arr, low, high);

        cout << "Quick Sort Iteration: ";
        printArray(arr);

        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}

void merge(vector<int>& arr, int const left, int const mid, int
const right) {
    auto const subArrayOne = mid - left + 1;
    auto const subArrayTwo = right - mid;

    vector<int> leftArray(subArrayOne), rightArray(subArrayTwo);

    for (auto i = 0; i < subArrayOne; i++)
        leftArray[i] = arr[left + i];
    for (auto j = 0; j < subArrayTwo; j++)
        rightArray[j] = arr[mid + 1 + j];

    auto indexOfSubArrayOne = 0, indexOfSubArrayTwo = 0;
    int indexOfMergedArray = left;

    while (indexOfSubArrayOne < subArrayOne && indexOfSubArrayTwo <
subArrayTwo) {
        if (leftArray[indexOfSubArrayOne] <=
rightArray[indexOfSubArrayTwo]) {
```

```
            arr[indexOfMergedArray] = leftArray[indexOfSubArrayOne];
            indexOfSubArrayOne++;
        } else {
            arr[indexOfMergedArray] =
rightArray[indexOfSubArrayTwo];
            indexOfSubArrayTwo++;
        }
        indexOfMergedArray++;
    }

    while (indexOfSubArrayOne < subArrayOne) {
        arr[indexOfMergedArray] = leftArray[indexOfSubArrayOne];
        indexOfSubArrayOne++;
        indexOfMergedArray++;
    }

    while (indexOfSubArrayTwo < subArrayTwo) {
        arr[indexOfMergedArray] = rightArray[indexOfSubArrayTwo];
        indexOfSubArrayTwo++;
        indexOfMergedArray++;
    }
}

void mergeSort(vector<int>& arr, int const begin, int const end) {
    if (begin >= end)
        return;

    auto mid = begin + (end - begin) / 2;
    mergeSort(arr, begin, mid);
    mergeSort(arr, mid + 1, end);
    merge(arr, begin, mid, end);

    cout << "Merge Sort Iteration: ";
```

```
    printArray(arr);
}

int main() {
    vector<int> arr = {9, 3, 8, 5, 4, 6, 7, 2, 1};
    cout << "Original Array: ";
    printArray(arr);

    cout << "Starting Quick Sort:" << endl;
    quickSort(arr, 0, arr.size() - 1);

    cout << "Starting Merge Sort:" << endl;
    mergeSort(arr, 0, arr.size() - 1);

    return 0;
}
```

Output:

```
/tmp/JQEzZRFDkv.o
Original Array: 9 3 8 5 4 6 7 2 1
Starting Quick Sort:
Quick Sort Iteration: 1 3 8 5 4 6 7 2 9
Quick Sort Iteration: 1 3 8 5 4 6 7 2 9
Quick Sort Iteration: 1 2 8 5 4 6 7 3 9
Quick Sort Iteration: 1 2 3 5 4 6 7 8 9
Quick Sort Iteration: 1 2 3 5 4 6 7 8 9
Quick Sort Iteration: 1 2 3 5 4 6 7 8 9
Quick Sort Iteration: 1 2 3 5 4 6 7 8 9
Quick Sort Iteration: 1 2 3 4 5 6 7 8 9
Starting Merge Sort:
Merge Sort Iteration: 1 2 3 4 5 6 7 8 9
Merge Sort Iteration: 1 2 3 4 5 6 7 8 9
Merge Sort Iteration: 1 2 3 4 5 6 7 8 9
Merge Sort Iteration: 1 2 3 4 5 6 7 8 9
Merge Sort Iteration: 1 2 3 4 5 6 7 8 9
Merge Sort Iteration: 1 2 3 4 5 6 7 8 9
Merge Sort Iteration: 1 2 3 4 5 6 7 8 9
Merge Sort Iteration: 1 2 3 4 5 6 7 8 9
```

Latihan 2

2) Tambahkan kode program untuk menghitung banyaknya perbandingan dan pergeseran pada algoritma quick sort dan merge sort.

Input:

```
#include <iostream>
#include <vector>

using namespace std;

int comparisons = 0;
int swaps = 0;

void printArray(const vector<int>& arr) {
    for (int i : arr) {
        cout << i << " ";
    }
    cout << endl;
}

int partition(vector<int>& arr, int low, int high) {
    int pivot = arr[high];
    int i = (low - 1);

    for (int j = low; j <= high - 1; j++) {
        comparisons++;
        if (arr[j] < pivot) {
            i++;
            swap(arr[i], arr[j]);
            swaps++;
        }
    }
    swap(arr[i + 1], arr[high]);
    swaps++;
```

```
    return (i + 1);
}

void quickSort(vector<int>& arr, int low, int high) {
    if (low < high) {
        int pi = partition(arr, low, high);

        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}

void merge(vector<int>& arr, int const left, int const mid, int
const right) {
    auto const subArrayOne = mid - left + 1;
    auto const subArrayTwo = right - mid;

    vector<int> leftArray(subArrayOne), rightArray(subArrayTwo);

    for (auto i = 0; i < subArrayOne; i++)
        leftArray[i] = arr[left + i];
    for (auto j = 0; j < subArrayTwo; j++)
        rightArray[j] = arr[mid + 1 + j];

    auto indexOfSubArrayOne = 0, indexOfSubArrayTwo = 0;
    int indexOfMergedArray = left;

    while (indexOfSubArrayOne < subArrayOne && indexOfSubArrayTwo <
subArrayTwo) {
        comparisons++;
        if (leftArray[indexOfSubArrayOne] <=
rightArray[indexOfSubArrayTwo]) {
            arr[indexOfMergedArray] = leftArray[indexOfSubArrayOne];
            indexOfSubArrayOne++;
```

```
        } else {
            arr[indexOfMergedArray] =
rightArray[indexOfSubArrayTwo];
            indexOfSubArrayTwo++;
        }
        indexOfMergedArray++;
    }

    while (indexOfSubArrayOne < subArrayOne) {
        arr[indexOfMergedArray] = leftArray[indexOfSubArrayOne];
        indexOfSubArrayOne++;
        indexOfMergedArray++;
    }

    while (indexOfSubArrayTwo < subArrayTwo) {
        arr[indexOfMergedArray] = rightArray[indexOfSubArrayTwo];
        indexOfSubArrayTwo++;
        indexOfMergedArray++;
    }
}

void mergeSort(vector<int>& arr, int const begin, int const end) {
    if (begin >= end)
        return;

    auto mid = begin + (end - begin) / 2;
    mergeSort(arr, begin, mid);
    mergeSort(arr, mid + 1, end);
    merge(arr, begin, mid, end);
}

int main() {
    vector<int> arr = {9, 3, 8, 5, 4, 6, 7, 2, 1};
```

```
    cout << "Original Array: ";
    printArray(arr);

    quickSort(arr, 0, arr.size() - 1);
    cout << "Quick Sort - Comparisons: " << comparisons << ", Swaps: " << swaps << endl;

    // Reset counters for merge sort
    comparisons = 0;
    swaps = 0;

    mergeSort(arr, 0, arr.size() - 1);
    cout << "Merge Sort - Comparisons: " << comparisons << ", Swaps: " << swaps << endl;

    return 0;
}
```

Output:

```
/tmp/v6SyheJ8SS.o
Original Array: 9 3 8 5 4 6 7 2 1
Quick Sort - Comparisons: 36, Swaps: 24
Merge Sort - Comparisons: 16, Swaps: 0
```

Latihan 3

3) Implementasikan pengurutan data Pegawai pada tugas pendahuluan dengan ketentuan :

a. Metode pengurutan dapat dipilih.

b. Pengurutan dapat dipilih secara urut naik atau turun.

c. Pengurutan dapat dipilih berdasarkan NIP dan NAMA.

d. Gunakan struktur data array.

Input:

```
#include <iostream>
#include <algorithm>
#include <vector>
#include <string>

using namespace std;

struct Pegawai {
    string NIP;
    string NAMA;
};

bool compareByNIPAsc(const Pegawai& a, const Pegawai& b) {
    return a.NIP < b.NIP;
}

bool compareByNIPDesc(const Pegawai& a, const Pegawai& b) {
    return a.NIP > b.NIP;
}

bool compareByNamaAsc(const Pegawai& a, const Pegawai& b) {
    return a.NAMA < b.NAMA;
}

bool compareByNamaDesc(const Pegawai& a, const Pegawai& b) {
    return a.NAMA > b.NAMA;
}

void printPegawai(const vector<Pegawai>& pegawai) {
    for (const auto& p : pegawai) {
        cout << "NIP: " << p.NIP << ", NAMA: " << p.NAMA << endl;
```

```
    }
}

int main() {
    vector<Pegawai> pegawai = {
        {"19820304", "Dicky"},
        {"19810602", "Ibnu"},
        {"19850708", "Adit"},
        {"19830910", "Dandi"}
    };

    int metodeSort, jenisSort;
    cout << "Pilih metode sort (1 untuk NIP, 2 untuk NAMA): ";
    cin >> metodeSort;
    cout << "Pilih jenis sort (1 untuk Ascending, 2 untuk
Descending): ";
    cin >> jenisSort;

    if (metodeSort == 1) {
        if (jenisSort == 1) {
            sort(pegawai.begin(), pegawai.end(), compareByNIPAsc);
        } else {
            sort(pegawai.begin(), pegawai.end(), compareByNIPDesc);
        }
    } else {
        if (jenisSort == 1) {
            sort(pegawai.begin(), pegawai.end(), compareByNamaAsc);
        } else {
            sort(pegawai.begin(), pegawai.end(), compareByNamaDesc);
        }
    }

    cout << "Data Pegawai setelah diurutkan:" << endl;
```

```
    printPegawai(pegawai);


    return 0;
}
```

Output:

```
/tmp/c9ESF9kXKT.o
Pilih metode sort (1 untuk NIP, 2 untuk NAMA): 1
Pilih jenis sort (1 untuk Ascending, 2 untuk Descending): 2
Data Pegawai setelah diurutkan:
NIP: 19850708, NAMA: Adit
NIP: 19830910, NAMA: Dandi
NIP: 19820304, NAMA: Dicky
NIP: 19810602, NAMA: Ibnu
```

Latihan 4

4) Berikan kesimpulan dari percobaan dan latihan yang telah Anda lakukan.

Algoritma Quick Sort:

Quick sort adalah algoritma pengurutan yang efisien dan umum digunakan, yang bekerja berdasarkan prinsip divide and conquer. Kode yang diberikan menunjukkan bagaimana quick sort membagi array menjadi dua sub-array, mengurutkan mereka secara independen, dan kemudian menggabungkannya.

Dengan menambahkan penghitungan perbandingan dan pergeseran, kita dapat menganalisis efisiensi algoritma dalam hal jumlah operasi yang dilakukan.

Algoritma Merge Sort:

Merge sort juga menggunakan pendekatan divide and conquer untuk mengurutkan array. Algoritma ini stabil dan efisien untuk dataset yang besar, tetapi memerlukan ruang tambahan untuk array sementara. Seperti quick sort, kode juga mencakup penghitungan perbandingan dan pergeseran untuk analisis kinerja.