PRAKTIKUM STRUKTUR DATA

INF11198

2401020010 – FACHREZI BACHRI

1. Pembuatan codingan untuk metode pengurutan yang dimana mengurutkan dari angka terkecil ke terbesar (ascending order), menggunakan algoritma heap sort

Sourch code algoritma heap sort:

#include <stdio.h>

#include <stdlib.h>

*/// Recursive function heapify. Pahami cara kerjanya*

*void* heapify(*int* *arr*[], *int* *n*, *int* *i*) {

*int* largest = *i*; *// Initialize largest as root*

*int* left = 2 \* *i* + 1; *// Left child*

*int* right = 2 \* *i* + 2; *// Right child*

*// Check if left child is larger than root*

    if (left < *n* && *arr*[left] > *arr*[largest])

        largest = left;

*// Check if right child is larger than the largest so far*

    if (right < *n* && *arr*[right] > *arr*[largest])

        largest = right;

*// If largest is not root, swap and continue heapifying*

    if (largest != *i*) {

*int* temp = *arr*[*i*];

*arr*[*i*] = *arr*[largest];

*arr*[largest] = temp;

        heapify(*arr*, *n*, largest);

    }

}

*// Fungsi heap sort dimana menggunakan fungsi rekursif heapify*

*void* heapSort(*int* *arr*[], *int* *n*) {

*// Build max heap*

    for (*int* i = *n* / 2 - 1; i >= 0; i--)

        heapify(*arr*, *n*, i);

*// Extract elements from the heap one by one*

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

*// Move current root to end*

*int* temp = *arr*[0];

*arr*[0] = *arr*[i];

*arr*[i] = temp;

*// Call heapify on the reduced heap*

        heapify(*arr*, i, 0);

    }

}

*// Tampilkan isi data*

*void* printArray(*int* *arr*[], *int* *n*) {

    for (*int* i = 0; i < *n*; i++){

        printf("%d ", *arr*[i]);

    }

    printf("\n");

}

*int*\* readCSV(const *char* \**filename*, *int* \**count*) {

    FILE \*file = fopen(*filename*, "r");

    if (file == NULL) {

        printf("Error: Cannot open file %s\n", *filename*);

        \**count* = 0;

        return NULL;

    }

*char* line[256];

    fgets(line, sizeof(line), file);

*int* capacity = 10;

*int* \*arr = (*int* \*)malloc(capacity \* sizeof(*int*));

    if (arr == NULL) {

        printf("Error: Memory allocation failed\n");

        fclose(file);

        \**count* = 0;

        return NULL;

    }

*int* n = 0;

*int* num;

    while (fscanf(file, "%d,", &num) == 1 || fscanf(file, "%d", &num) == 1) {

        if (n >= capacity) {

            capacity \*= 2;

*int* \*temp = (*int* \*)realloc(arr, capacity \* sizeof(*int*));

            if (temp == NULL) {

                printf("Error: Memory reallocation failed\n");

                free(arr);

                fclose(file);

                \**count* = 0;

                return NULL;

            }

            arr = temp;

        }

        arr[n] = num;

        n++;

    }

    fclose(file);

    \**count* = n;

    return arr;

}

*// Program utama. Modifikasi untuk dapat menerima dan membaca data file*

*int* main(*int* *argc*, *char* \**argv*[]) {

*int* n;

*int* \*arr = readCSV(*argv*[1], &n);

    printf("Original Array: ");

    printArray(arr, n);

    heapSort(arr, n);

    printf("Sorted Array: ");

    printArray(arr, n);

    free(arr);

    return 0;

}

1. Pembuatan codingan untuk metode pengurutan yang dimana mengurutkan dari angka terkecil ke terbesar (ascending order), menggunakan algoritma merge sort

Sourch code algoritma merge sort:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

*// Function to merge two subarrays*

*void* merge(*int* *arr*[], *int* *left*, *int* *mid*, *int* *right*) {

*int* n1 = *mid* - *left* + 1;

*int* n2 = *right* - *mid*;

*// Create temporary arrays*

*int* leftArr[n1], rightArr[n2];

*// Copy data to temporary arrays*

    for (*int* i = 0; i < n1; i++)

        leftArr[i] = *arr*[*left* + i];

    for (*int* j = 0; j < n2; j++)

        rightArr[j] = *arr*[*mid* + 1 + j];

*// Merge the temporary arrays back into arr[]*

*int* i = 0, j = 0, k = *left*;

    while (i < n1 && j < n2) {

        if (leftArr[i] <= rightArr[j]) {

*arr*[k] = leftArr[i];

            i++;

        } else {

*arr*[k] = rightArr[j];

            j++;

        }

        k++;

    }

*// Copy remaining elements of leftArr[], if any*

    while (i < n1) {

*arr*[k] = leftArr[i];

        i++;

        k++;

    }

*// Copy remaining elements of rightArr[], if any*

    while (j < n2) {

*arr*[k] = rightArr[j];

        j++;

        k++;

    }

}

*// Fungsi rekursif merge sort yang memakai fungsi merge untuk gabung*

*void* mergeSort(*int* *arr*[], *int* *left*, *int* *right*) {

    if (*left* < *right*) {

*int* mid = *left* + (*right* - *left*) / 2;

*// Recursively sort first and second halves*

        mergeSort(*arr*, *left*, mid);

        mergeSort(*arr*, mid + 1, *right*);

*// Merge the sorted halves*

        merge(*arr*, *left*, mid, *right*);

    }

}

*// Fungsi utama. Modifikasi agar membaca data file CSV.*

*int* main(*int* *argc*, *char* \**argv*[]) {

    FILE \*file = fopen(*argv*[1], "r");

*char* line[256];

    fgets(line, sizeof(line), file);

*int* arr[10000];

*int* n = 0;

    while (fgets(line, sizeof(line), file) && n < 10000) {

*char* \*token = strtok(line, ",\n");

        while (token != NULL && n < 10000) {

            arr[n] = atoi(token);

            n++;

            token = strtok(NULL, ",\n");

        }

    }

    fclose(file);

    printf("Original array: ");

    for (*int* i = 0; i < n; i++)

        printf("%d ", arr[i]);

    printf("\n");

    mergeSort(arr, 0, n - 1);

    printf("Sorted array: ");

    for (*int* i = 0; i < n; i++)

        printf("%d ", arr[i]);

    printf("\n");

    return 0;

}

1. Pembuatan codingan untuk metode pengurutan yang dimana mengurutkan dari angka terkecil ke terbesar (ascending order), menggunakan algoritma insertion sort

Sourch code algoritma insertion sort:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define RUN 32

// Function to perform insertion sort

void insertionSort(int arr[], int left, int right) {

    for (int i = left + 1; i <= right; i++) {

        int temp = arr[i];

        int j = i - 1;

        while (j >= left && arr[j] > temp) {

            arr[j + 1] = arr[j];

            j--;

        }

        arr[j + 1] = temp;

    }

}

// Function to merge two sorted subarrays

void merge(int arr[], int left, int mid, int right) {

    int len1 = mid - left + 1, len2 = right - mid;

    int leftArr[len1], rightArr[len2];  // Array statis (VLA)

    for (int i = 0; i < len1; i++) leftArr[i] = arr[left + i];

    for (int i = 0; i < len2; i++) rightArr[i] = arr[mid + 1 + i];

    int i = 0, j = 0, k = left;

    while (i < len1 && j < len2) {

        if (leftArr[i] <= rightArr[j]) {

            arr[k++] = leftArr[i++];

        } else {

            arr[k++] = rightArr[j++];

        }

    }

    while (i < len1)

        arr[k++] = leftArr[i++];

    while (j < len2)

        arr[k++] = rightArr[j++];

}

// Function to perform TimSort

void timSort(int arr[], int n) {

    for (int i = 0; i < n; i += RUN) {

        int right = (i + RUN - 1 < n - 1) ? i + RUN - 1 : n - 1;

        insertionSort(arr, i, right);

    }

    for (int size = RUN; size < n; size = 2 \* size) {

        for (int left = 0; left < n; left += 2 \* size) {

            int mid = left + size - 1;

            int right = ((left + 2 \* size - 1) < (n - 1)) ? (left + 2 \* size - 1) : (n - 1);

            if (mid < right) {

                merge(arr, left, mid, right);

            }

        }

    }

}

// Main function. Modify here to read data from CSV file.

void printArray(int arr[], int n) {

    for (int i = 0; i < n; i++) {

        printf("%d ", arr[i]);

    }

    printf("\n");

}

// Main function dengan array statis

int main(int argc, char \*argv[]) {

    FILE \*file = fopen(argv[1], "r");

    char line[256];

    fgets(line, sizeof(line), file);

    int arr[10000];

    int n = 0;

    while (fgets(line, sizeof(line), file) && n < 10000) {

        char \*token = strtok(line, ",\n");

        while (token != NULL && n < 10000) {

            arr[n] = atoi(token);

            n++;

            token = strtok(NULL, ",\n");

        }

    }

    fclose(file);

    printf("Original Array: ");

    printArray(arr, n);

    timSort(arr, n);

    printf("Sorted Array: ");

    printArray(arr, n);

    return 0;

}

1. Pembuatan codingan untuk metode pengurutan yang dimana mengurutkan dari angka terkecil ke terbesar (ascending order), menggunakan algoritma insertion sort

Sourch code algoritma insertion sort:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

*void* countingSort(*int* *array*[], *int* *size*) {

*int* output[10000];

*int* max = *array*[0];

*int* i;

*// Find the maximum value in the array*

    for (i = 1; i < *size*; i++) {

        if (*array*[i] > max) {

            max = *array*[i];

        }

    }

*// Cek apakah max melebihi batas*

    if (max > 100000) {

        printf("Warning: Maximum value (%d) exceeds MAX\_VALUE (%d)\n", max, 100000);

        max = 100000;

    }

*// Initialize the count array*

*int* count[max + 1];

    memset(count, 0, sizeof(count));

*// Count the occurrences of each element*

    for (i = 0; i < *size*; i++) {

        count[*array*[i]]++;

    }

*// Update the count array to store cumulative counts*

    for (i = 1; i <= max; i++) {

        count[i] += count[i - 1];

    }

*// Build the output array*

    for (i = *size* - 1; i >= 0; i--) {

        output[count[*array*[i]] - 1] = *array*[i];

        count[*array*[i]]--;

    }

*// Copy the sorted elements back to the original array*

    for (i = 0; i < *size*; i++) {

*array*[i] = output[i];

    }

}

*int* main(*int* *argc*, *char* \**argv*[]) {

    FILE \*file = fopen(*argv*[1], "r");

*int* array[10000];

*int* size = 0;

*char* line[1024];

    while (fgets(line, sizeof(line), file) && size < 10000) {

*char* \*token = strtok(line, ",\n");

        while (token != NULL && size < 10000) {

            if (strcmp(token, "Angka") != 0) {

                array[size++] = atoi(token);

            }

            token = strtok(NULL, ",\n");

        }

    }

    fclose(file);

    printf("Original array: ");

    for (*int* i = 0; i < size; i++) {

        printf("%d ", array[i]);

    }

    printf("\n");

    countingSort(array, size);

    printf("Sorted array: ");

    for (*int* i = 0; i < size; i++) {

        printf("%d ", array[i]);

    }

    printf("\n");

    return 0;

}

1. **Comparison Sort (Heap, Merge, Tim)** cocok untuk pengurutan umum yang melibatkan perbandingan antar nilai. **Non-Comparison Sort (Counting)** lebih cepat jika data berupa angka bulat dengan rentang nilai terbatas.