Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования

**«Пермский национальный исследовательский
политехнический университет»**

Электротехнический факультет
Кафедра «Информационные технологии и автоматизированные системы»
направление подготовки: 09.03.01– «Информатика и вычислительная
техника»

Отчет по
лабораторной работе 12

Выполнил студент гр. ИВТ-23-1б
Давыдов Андрей Юрьевич

Проверил:
ст. преп. каф. ИТАС
Яруллин Денис Владимирович

(оценка) (подпись)

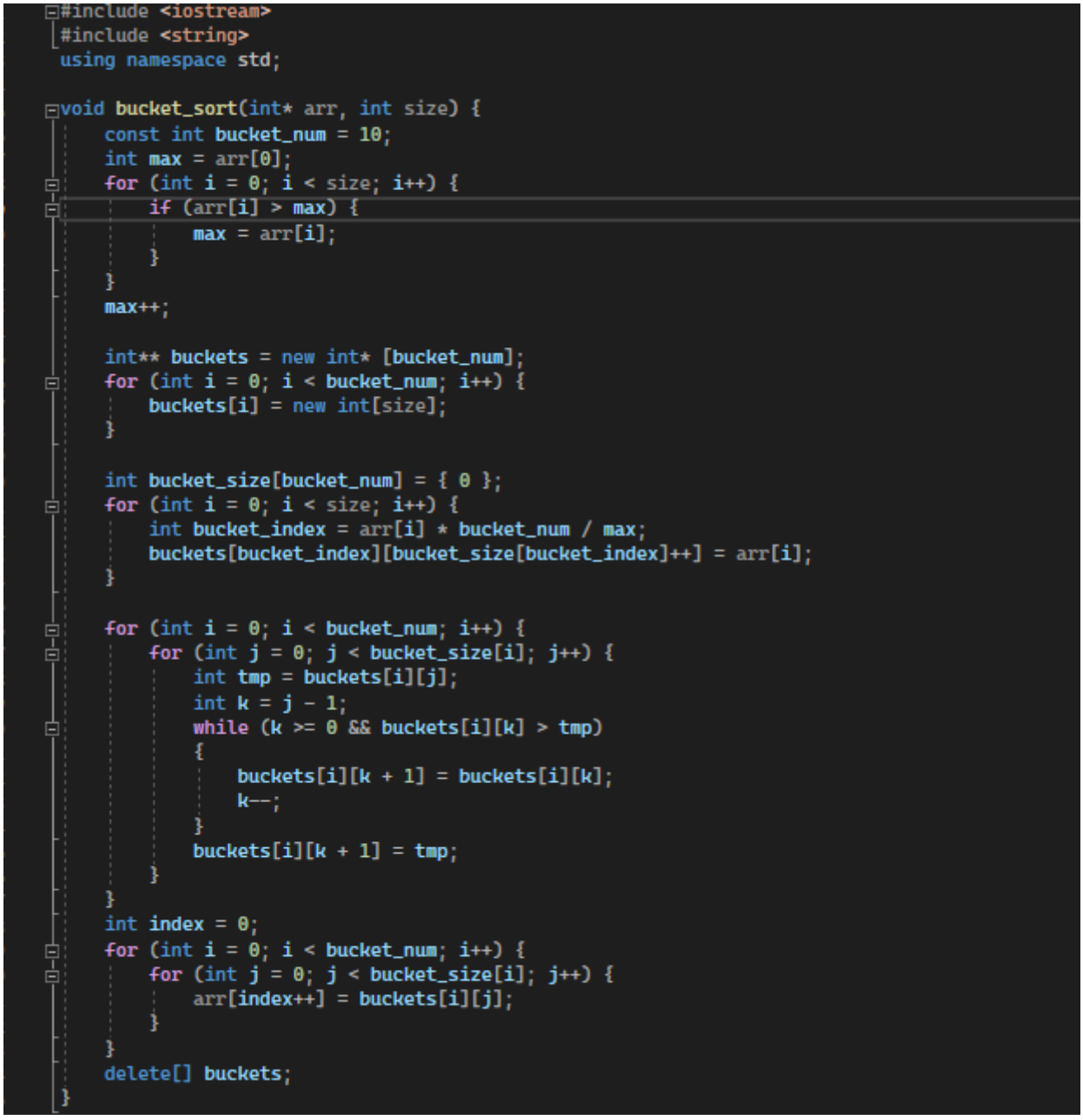
(дата)

г. Пермь, 2023

Постановка задачи

Реализовать 4 сортировки: блочная, подсчетом, слияния, быстрая по Ломоту.

Код программы:



```cpp
#include <iostream>
#include <string>
using namespace std;

void bucket_sort(int* arr, int size) {
    const int bucket_num = 10;
    int max = arr[0];
    for (int i = 0; i < size; i++) {
        if (arr[i] > max) {
            max = arr[i];
        }
    }
    max++;

    int** buckets = new int* [bucket_num];
    for (int i = 0; i < bucket_num; i++) {
        buckets[i] = new int[size];
    }

    int bucket_size[bucket_num] = { 0 };
    for (int i = 0; i < size; i++) {
        int bucket_index = arr[i] * bucket_num / max;
        buckets[bucket_index][bucket_size[bucket_index]++] = arr[i];
    }

    for (int i = 0; i < bucket_num; i++) {
        for (int j = 0; j < bucket_size[i]; j++) {
            int tmp = buckets[i][j];
            int k = j - 1;
            while (k >= 0 && buckets[i][k] > tmp)
            {
                buckets[i][k + 1] = buckets[i][k];
                k--;
            }
            buckets[i][k + 1] = tmp;
        }
    }
    int index = 0;
    for (int i = 0; i < bucket_num; i++) {
        for (int j = 0; j < bucket_size[i]; j++) {
            arr[index++] = buckets[i][j];
        }
    }
    delete[] buckets;
}
```

```
void countingSort(int* dataArray, int arraySize) {
    int largestValue = dataArray[0];
    for (int i = 1; i < arraySize; i++) {
        if (largestValue < dataArray[i]) {
            largestValue = dataArray[i];
        }
    }
    largestValue++;

    int* countingArray = new int[largestValue];
    int* sortedArray = new int[arraySize];

    for (int i = 0; i < largestValue; i++) {
        countingArray[i] = 0;
    }
    for (int i = 0; i < arraySize; i++) {
        countingArray[dataArray[i]]++;
    }
    for (int i = 1; i < largestValue; ++i) {
        countingArray[i] += countingArray[i - 1];
    }
    for (int i = arraySize - 1; i >= 0; i--) {
        sortedArray[countingArray[dataArray[i]] - 1] = dataArray[i];
        countingArray[dataArray[i]]--;
    }
    for (int i = 0; i < arraySize; i++) {
        dataArray[i] = sortedArray[i];
    }

    delete[] countingArray;
    delete[] sortedArray;
}
void mergeArrays(int* sequence, int start, int middle, int end) {
    int leftSubarraySize = middle - start + 1;
    int rightSubarraySize = end - middle;
    int* leftSubarray = new int[leftSubarraySize];
    int* rightSubarray = new int[rightSubarraySize];

    for (int i = 0; i < leftSubarraySize; i++) {
        leftSubarray[i] = sequence[start + i];
    }
    for (int i = 0; i < rightSubarraySize; i++) {
        rightSubarray[i] = sequence[middle + 1 + i];
    }

    int indexLeft = 0, indexRight = 0, indexMerged = start;
    while (indexLeft < leftSubarraySize && indexRight < rightSubarraySize) {
        if (leftSubarray[indexLeft] <= rightSubarray[indexRight]) {
            sequence[indexMerged] = leftSubarray[indexLeft];
            indexLeft++;
        }
        else {
            sequence[indexMerged] = rightSubarray[indexRight];
            indexRight++;
        }
        indexMerged++;
    }

    while (indexLeft < leftSubarraySize) {
        sequence[indexMerged] = leftSubarray[indexLeft];
        indexMerged++;
        indexLeft++;
    }

    while (indexRight < rightSubarraySize) {
        sequence[indexMerged] = rightSubarray[indexRight];
        indexMerged++;
        indexRight++;
    }

    delete[] leftSubarray;
    delete[] rightSubarray;
}
```

```cpp
void mergeSort(int* sequence, int start, int end) {
    if (start < end) {
        int middle = start + (end - start) / 2;
        mergeSort(sequence, start, middle);
        mergeSort(sequence, middle + 1, end);
        mergeArrays(sequence, start, middle, end);
    }
}

int partition(int* sequence, int start, int end, int pivotValue) {
    int pivotIndex = start;
    for (int i = start; i <= end; i++) {
        if (sequence[i] <= pivotValue) {
            std::swap(sequence[pivotIndex], sequence[i]);
            pivotIndex++;
        }
    }
    pivotIndex--;
    return pivotIndex;
}

void quickSort(int* sequence, int start, int end) {
    if (start < end) {
        int pivotValue = sequence[end];
        int partitionIndex = partition(sequence, start, end, pivotValue);
        quickSort(sequence, start, partitionIndex - 1);
        quickSort(sequence, partitionIndex + 1, end);
    }
}

void displayArray(int* array, int size) {
    for (int i = 0; i < size; i++) {
        cout << array[i] << " ";
    }
    cout << "\n";
}
// Menu function to choose the sorting algorithm
void menu(int* array, int size) {
    int choice;
    cout << "Select the sorting algorithm:\n";
    cout << "1. Bucket Sort\n";
    cout << "2. Counting Sort\n";
    cout << "3. Merge Sort\n";
    cout << "4. Quick Sort\n";
    cout << "Enter your choice (1-4): ";
    cin >> choice;

    switch (choice) {
    case 1:
        bucket_sort(array, size);
        break;
    case 2:
        countingSort(array, size);
        break;
    case 3:
        mergeSort(array, 0, size - 1);
        break;
    case 4:
        quickSort(array, 0, size - 1);
        break;
    default:
        cout << "Invalid choice. Please enter a number between 1 and 4.\n";
        return;
    }

    cout << "Sorted array: ";
    displayArray(array, size);
}

// Main function to run the program
int main() {
    int array[] = { 3, 6, 8, 10, 1, 2, 1 };
    int size = sizeof(array) / sizeof(array[0]);

    cout << "Original array: ";
    displayArray(array, size);

    menu(array, size);

    return 0;
}
```

Тест программы

```
Original array: 3 6 8 10 1 2 1
Select the sorting algorithm:
1. Bucket Sort
2. Counting Sort
3. Merge Sort
4. Quick Sort
Enter your choice (1-4): 1
Sorted array: 1 1 2 3 6 8 10
```

```
Original array: 3 6 8 10 1 2 1
Select the sorting algorithm:
1. Bucket Sort
2. Counting Sort
3. Merge Sort
4. Quick Sort
Enter your choice (1-4): 2
Sorted array: 1 1 2 3 6 8 10
```

```
Original array: 3 6 8 10 1 2 1
Select the sorting algorithm:
1. Bucket Sort
2. Counting Sort
3. Merge Sort
4. Quick Sort
Enter your choice (1-4): 3
Sorted array: 1 1 2 3 6 8 10
```

```
Original array: 3 6 8 10 1 2 1
Select the sorting algorithm:
1. Bucket Sort
2. Counting Sort
3. Merge Sort
4. Quick Sort
Enter your choice (1-4): 4
Sorted array: 1 1 2 3 6 8 10
```