МОСКОВСКИЙ ФИЗИКО-ТЕХНИЧЕСКИЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)
Физтех-школа аэрокосмических технологий

Отчет о выполнении лабораторной работы 1.2.2

Алгоритмы сортировок с различной ассмиптотикой

Ефремова Татьяна, Б03-503

6 октября 2025 г.

Цели работы: посмотреть на алгоритмы с различной асимптотикой, научиться анализировать время работы алгоритмов и включать разные степени оптимизации.

1 Простые сортировки

Сортировка пузырьком

```
void bubbleSort(int *arr){

    int i, j;
    for (i = 0; i < ARR_LEN; i++){

        for (j = 0; j < ARR_LEN - 1; j++){
            if (arr[j] > arr[j+1]){
                swap(arr, j, j+1);
            }
        }
    }
}
```

Сортировка шейкером

```
void shakerSort(int *arr){

    int i, j;
    bool changes = 0;
    for (i = 0; i < (ARR_LEN+1)/2; i++){

        changes = 0;
        for (j = i; j < ARR_LEN-1-i; j++){
            if (arr[j] > arr[j+1]){
                swap(arr, j, j+1);
                changes = 1;
            }
        }
        for (j = ARR_LEN-2-i; j >= i; j--){
            if (arr[j] > arr[j+1]){
                swap(arr, j, j+1);
                changes = 1;
            }
        }
        if (not changes){
            break;
        }
    }
}
```

Сортировка вставкой

```
void insertionSort(int *arr){

    int el, i, j;
    for (i = 1; i < ARR_LEN; i++){
        j = i-1;
        el = arr[i];

        while ((j >= 0) && (el < arr[j])){
            arr[j+1] = arr[j];
            j--;
        }
        arr[j+1] = el;
    }
}
```

Каждая из вышеперечисленных сортировок содержит по одному вложенному циклу, т. е. при исполнении программы производится n^2 итераций, где n – число элементов массива.

Тогда зависимость времени работы сортировки от количества элементов массива квадратичная, $O(n^2)$:

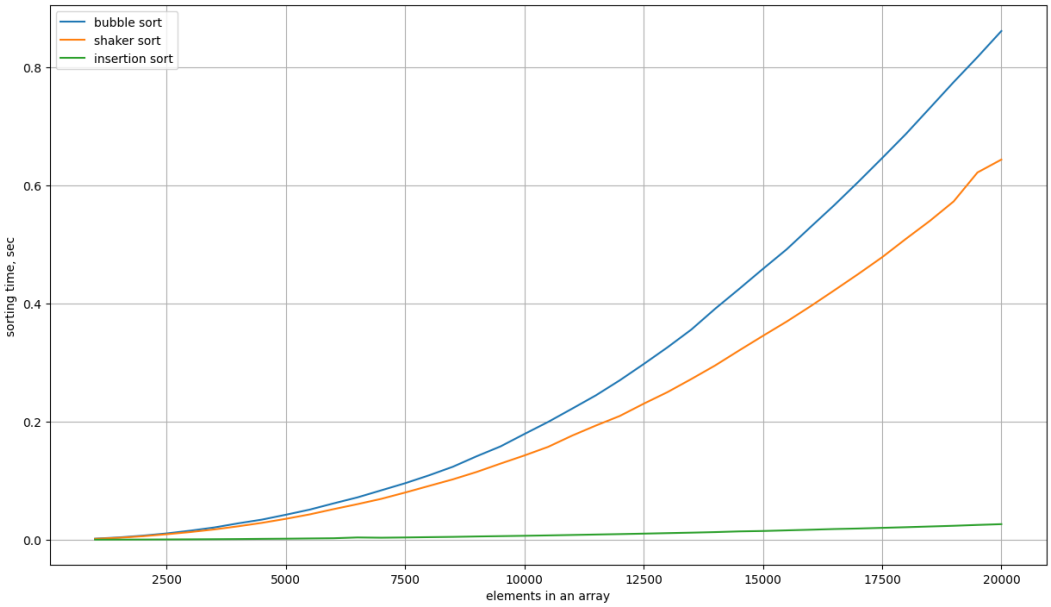

Рис. 1: График зависимости t от n простых сортировок

Зависимость логарифма времени от лографима количества элементов массива – линейная:

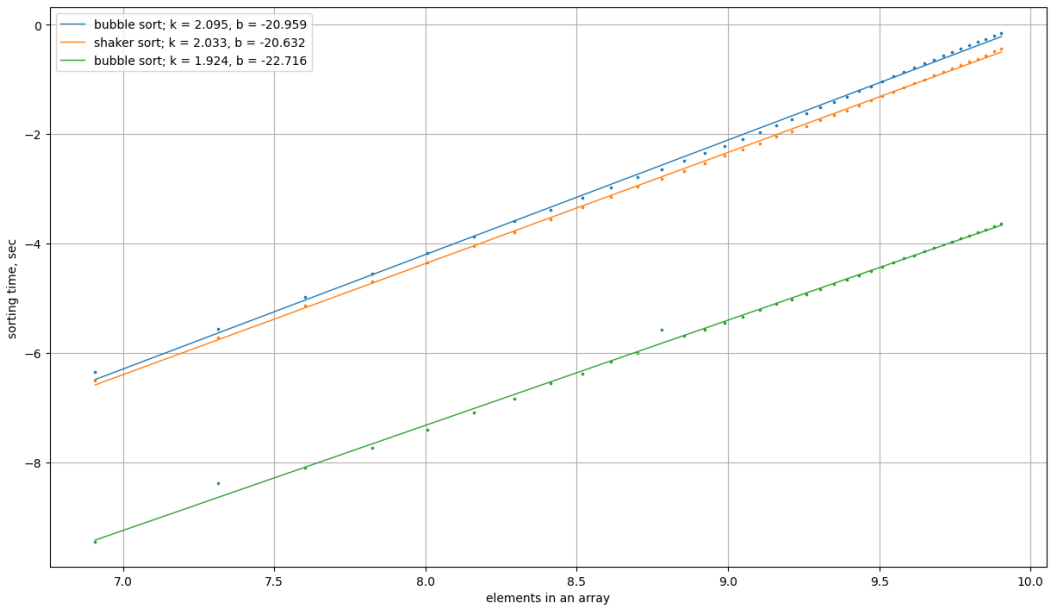

Рис. 2: График зависимости $log(t)$ от $log(n)$

Оптимизация сортировки пузырьком

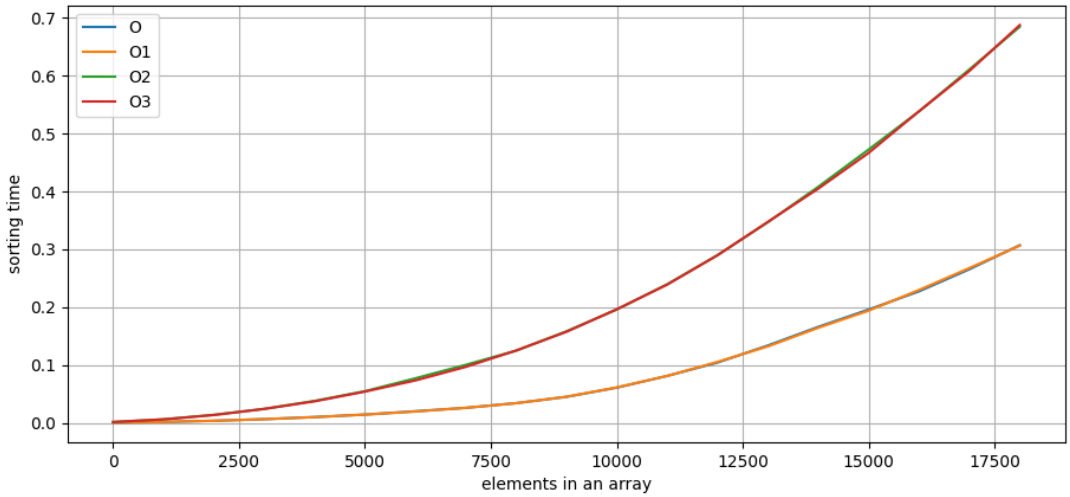

Рис. 3: Enter Caption

2 Быстрые сортировки

Сортировка кучей

```
void insert(int *heap, int N, int x){

    heap[N] = x;
    int cur = N;
    N += 1;
    while ((cur > 0) && (x > heap[(cur-1)/2])){
       heap[cur] = heap[(cur-1)/2];
       cur = (cur-1)/2;
       }
    heap[cur] = x;
}

void deleteMax(int *heap, int N){

    int phldr = heap [0];
    heap[0] = heap[N-1];
    heap[N-1] = phldr;
    N--;

    int x = heap[0];
    int cur = 0;
    int left, right;
    while (true) {
        left = x-1;
        if ((cur*2 + 1) < N){
            left = heap[cur*2 +1];
        }
        right = x -1;
        if ((cur*2 + 2) < N){
            right = heap[2*cur + 2];
        }
        int m = std::max(left, right);
        if (x > m){
            break;
```

```
        }
        else if (m == left){
            heap[cur] = left;
            cur = cur*2 + 1;
        }
        else{
            heap[cur] = right;
            cur = cur*2 + 2;
        }
    heap[cur] = x;
    }
}

void heapSort(int *heap, int *arr){

    int i, j;
    for (i = 0; i < ARR_LEN; i++){
        insert(heap, i, arr[i]);
    }
    for (i = 0; i < ARR_LEN-1; i++){
        deleteMax(heap, ARR_LEN-i);
    }
}
```

Сортировка слиянием

```
void merge(int *arr, int *tarr, int beg, int mid, int en){

    int i1 = beg;
    int i2 = mid;
    int k = beg;

    while ((i1 < mid) && (i2 < en)){
        if (arr[i1] < arr[i2]){
            tarr[k] = arr[i1];
            i1++;
        }
        else{
            tarr[k] = arr[i2];
            i2++;
        }
        k++;
    }
    while (i1 < mid){
        tarr[k] = arr[i1];
        i1++;
        k++;
    }
    while (i2 < en){
        tarr[k] = arr[i2];
        i2++;
        k++;
    }
    for (int k = beg; k < en; k++){
        arr[k] = tarr[k];
    }
}

void mergeSort(int *arr, int *tarr, int beg, int en){

    if (en > beg + 1){
        int mid = (en + beg)/2;
        mergeSort(arr, tarr, beg, mid);
```

```
        mergeSort(arr, tarr, mid, en);
        merge(arr, tarr, beg, mid, en);
    }
}
```

Сортировка Хоара

```
int partition(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, j);
        }
    }
    swap(arr, i + 1, high);
    return (i + 1);
}

void quickSort(int *arr, int low, int high){

    if (low < high){
        int pivot = partition(arr, low, high);

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

Каждая из вышеперечисленных сортировок производит $log(n)$ разбиений массива и проходится по каждому полученному куску n раз т. е. при исполнении программы производится $n \cdot log(n)$ итераций:

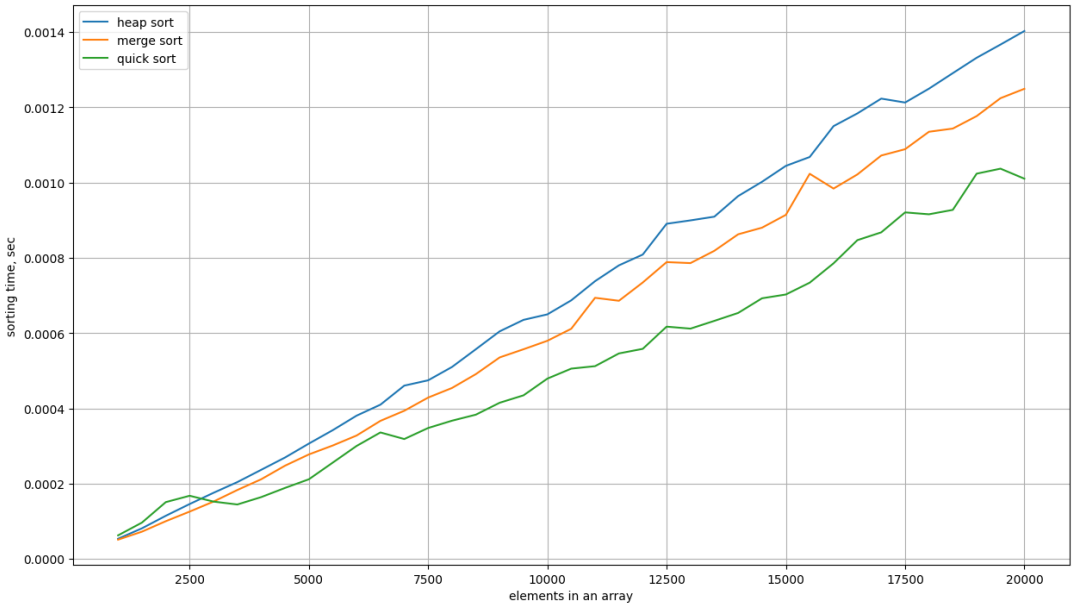

Рис. 4: График зависимости t от n быстрых сортировок

Тогда зависимость времени работы сортировки, деленного на $n \cdot log(n)$ от количества элементов массива должна быть близка к константе:

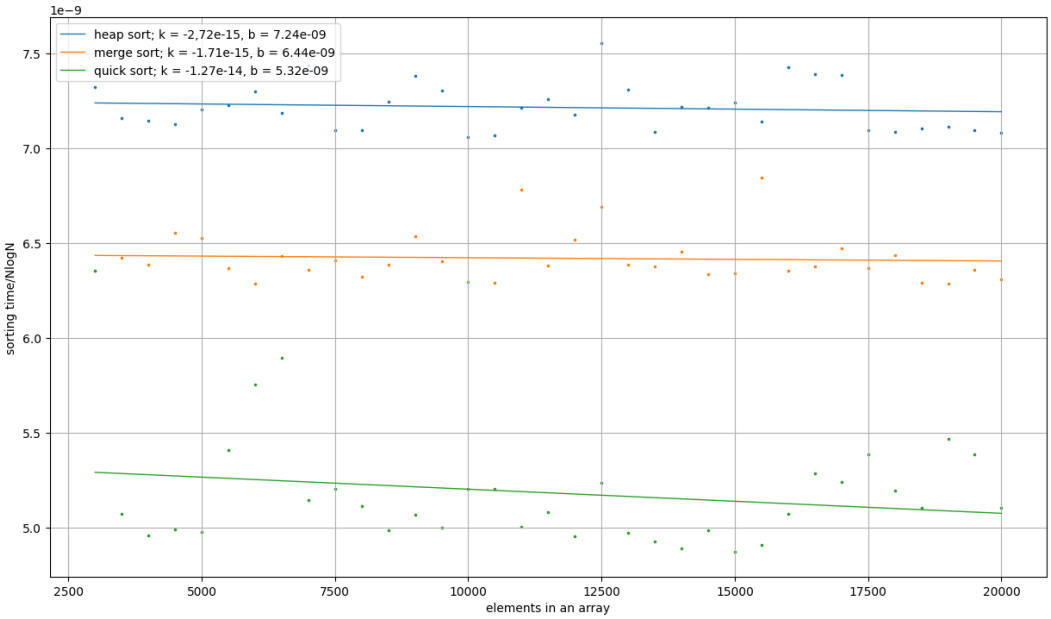

Рис. 5: График зависимости $\frac{t}{n \cdot log(n)}$ от n

3 Сравнение быстрых и медленных сортировок

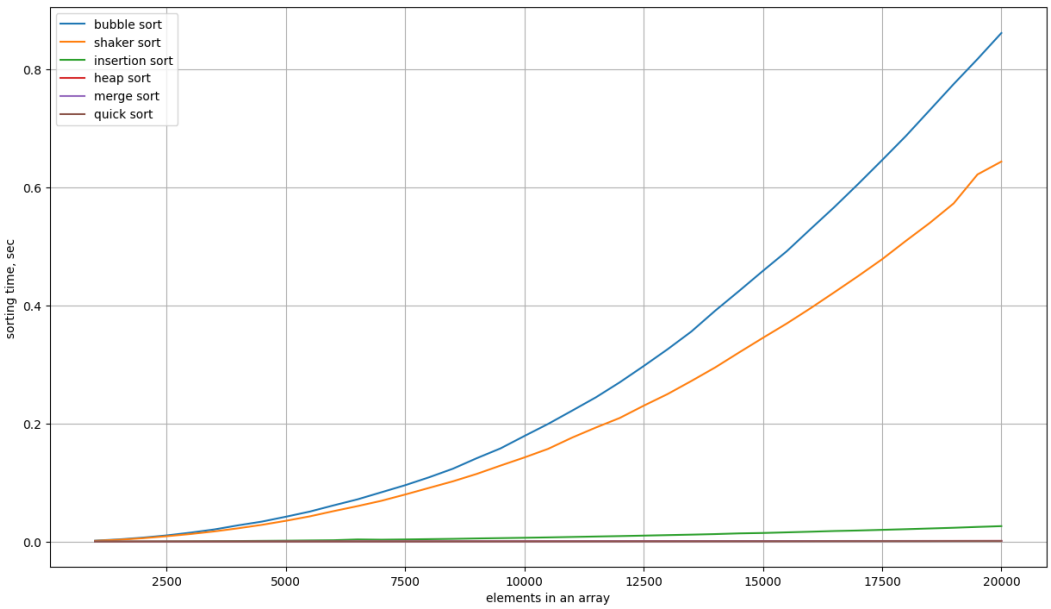

Рис. 6: График зависимостей t от n всех сортировок

4 Зависимость от начальных данных

Bubble Sort

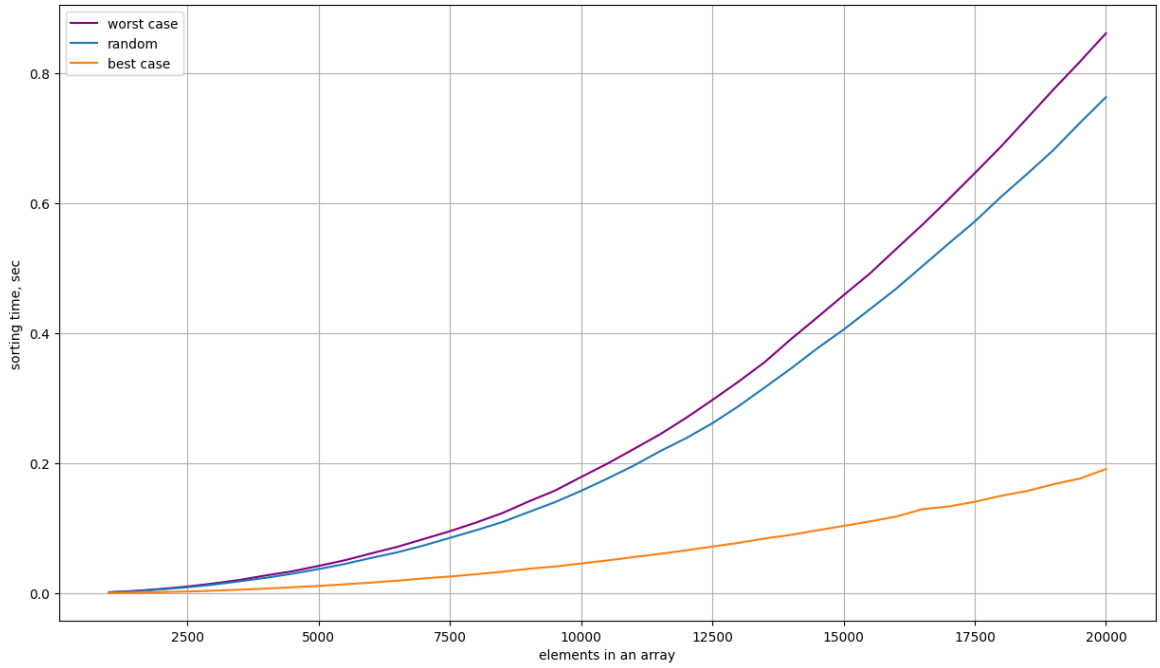

Рис. 7: Сравнение случаев сортировки пузырьком

Insertion Sort

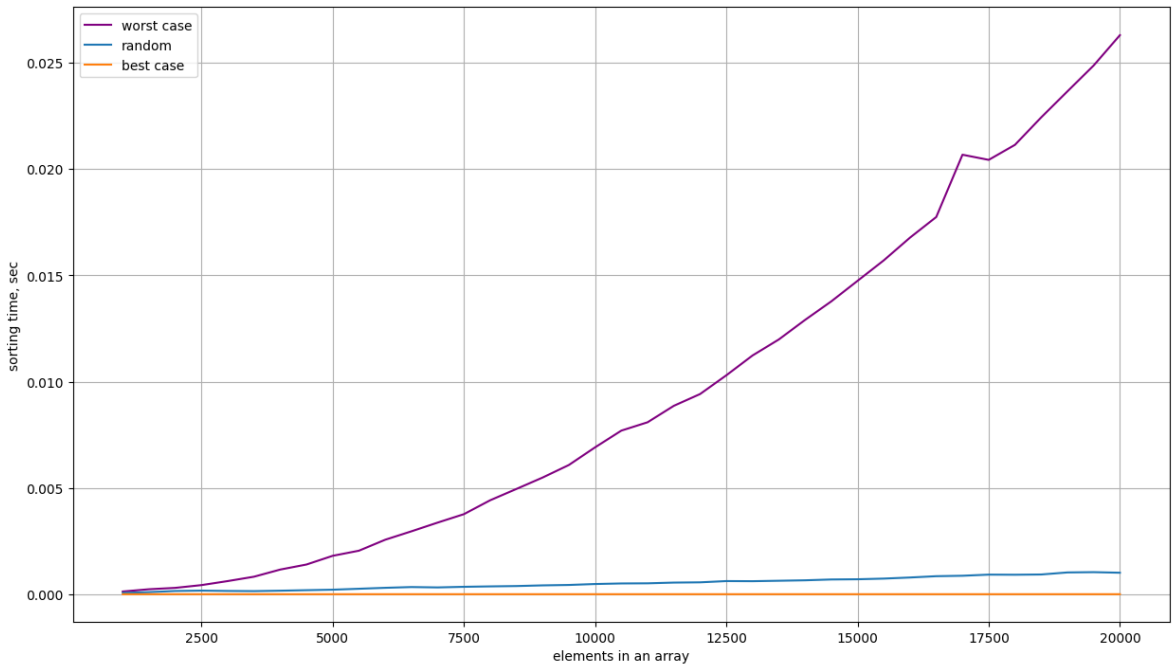

Рис. 8: Сравнение случаев сортировки вставкой

Quick Sort

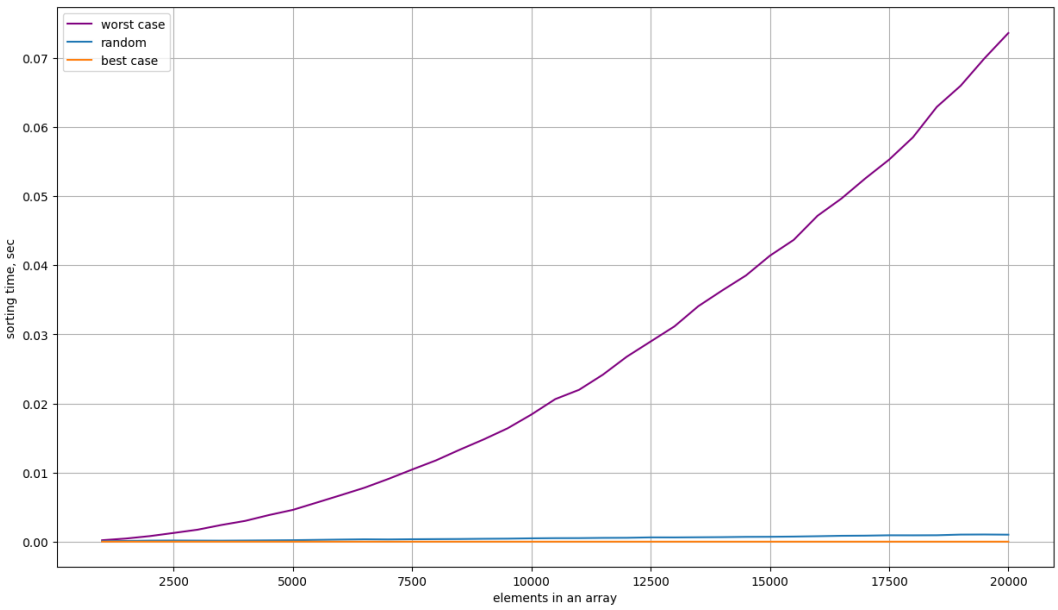

Рис. 9: Сравнение случаев сортировки Хоара

Merge Sort и Heap Sort

У сортировок кучей и слиянием как плохих случаев, так и хороших. Они всегда совершают ≈ одинаковое количество итераций.

5 Сравнение сортировок для C++ и Python

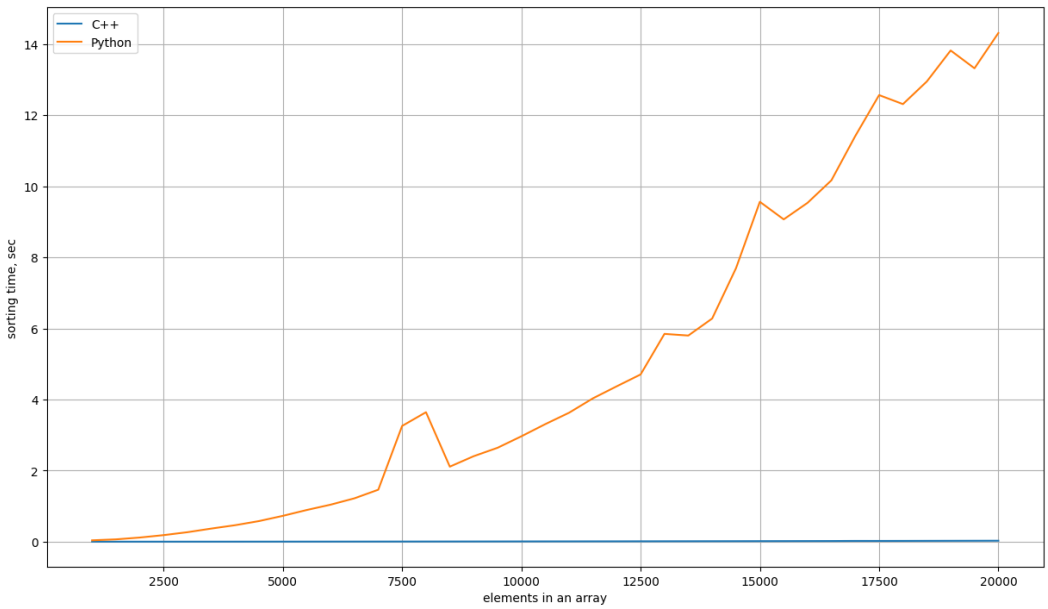

Рис. 10: сравнение графиков сортировки вставкой

6 Выводы

Были продемонстрированы зависимости времени работы различных сортировок от длины массивов, а также как различные оптимизации компилятора ускоряют процесс сортировок. Было изучено время работы сортировок в зависимости от начальных данных. Было показано, насколько язык C++ быстрее Python.