

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»
(БГТУ им. В.Г. Шухова)**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

Лабораторная работа №3

по дисциплине: Алгоритмы и структуры данных

тема: Сравнительный анализ методов сортировки (Pascal/C)

Выполнил: ст. группы ПВ-223
Пахомов Владислав Андреевич

Проверили: асс. Солонченко Роман
Евгеньевич

Белгород 2023 г.

Лабораторная работа №3

Сравнительный анализ методов сортировки (Pascal/C)

Цель работы: изучение методов сортировки массивов и приобретение навыков в проведении сравнительного анализа различных методов сортировки.

1. Листинг программы. *main.c*

```
#include <time.h>

#include <algc.h>

#define LOW 5
#define HIGH 45
#define STEP 5

int main() {
    srand(time(0));

    comparesExperiment(insertionSort, "insertion sort", LOW, HIGH, STEP);
    comparesExperiment(selectionSort, "selection sort", LOW, HIGH, STEP);
    comparesExperiment(bubbleSort, "bubble sort", LOW, HIGH, STEP);
    comparesExperiment(bubbleSortMod1, "bubble sort first modification", LOW, HIGH, STEP);
    comparesExperiment(bubbleSortMod2, "bubble sort second modification", LOW, HIGH,
        ↪ STEP);
    comparesExperiment(shellSort, "shell sort", LOW, HIGH, STEP);
    comparesExperiment(hoarSort, "quick sort", LOW, HIGH, STEP);
    comparesExperiment(heapSort, "heap sort", LOW, HIGH, STEP);

    return 0;
}
```

sorts.h

```
#ifndef SORTS
#define SORTS

#include <stdbool.h>

typedef void (*SortingFunction)(int*, int, int*);

#define INC_COMPARES(comps) ((!comps || ++(*comps)))

void swap(int* a, int* b);
```

```

bool isOrdered(int* a, int size);
bool isOrderedBackwards(int* a, int size);
void genOrdered(int *a, int size);
void genOrderedBackwards(int *a, int size);
void genRandom(int *a, int size);

void insertionSort(int* data, int size, int* comps);
void selectionSort(int* data, int size, int* comps);
void bubbleSort(int* data, int size, int* comps);
void bubbleSortMod1(int* data, int size, int* comps);
void bubbleSortMod2(int* data, int size, int* comps);
void shellSort(int* data, int size, int* comps);
void hoarSort(int* data, int size, int* comps);
void heapSort(int* data, int size, int* comps);

void comparesExperiment(SortingFunction function, char *sortingFunctionName, int low, int
↪ high, int step);

#endif

```

utility.c

```

#include <lab3/sorts.h>

#include <assert.h>
#include <limits.h>
#include <stdlib.h>
#include <stdbool.h>
#include <stdio.h>

void swap(int* a, int* b) {
    int t = *a;
    *a = *b;
    *b = t;
}

bool isOrdered(int* a, int size) {
    for (int i = 0; i < size - 1; i++)
        if (a[i] > a[i + 1]) return false;

    return true;
}

```

```

bool isOrderedBackwards(int* a, int size) {
    for (int i = 0; i < size - 1; i++)
        if (a[i] < a[i + 1]) return false;

    return true;
}

void genOrdered(int *a, int size) {
    int current = INT_MIN;

    for (int i = 0; i < size; i++) {
        a[i] = current;
        current += rand() % RAND_MAX;
    }
}

void genOrderedBackwards(int *a, int size) {
    int current = INT_MAX;

    for (int i = 0; i < size; i++) {
        a[i] = current;
        current -= rand() % RAND_MAX;
    }
}

void genRandom(int *a, int size) {
    for (int i = 0; i < size; i++)
        a[i] = rand() % RAND_MAX;
}

void comparesExperiment(SortingFunction function, char *sortingFunctionName, int low, int
↪ high, int step) {
    printf("=====\n\n");
    printf("Launching comparing experiment for function %s\n", sortingFunctionName);

    printf("\nOrdered array results:\n");
    for (int i = low; i <= high; i += step) {
        int* array = malloc(i * sizeof(int));
        genOrdered(array, i);
        int compares = 0;
        function(array, i, &compares);
        assert(isOrdered(array, i));
        printf("For %3d elements: %7d compares\n", i, compares);
    }
}

```

```

        free(array);
    }

    printf("\nOrdered backwards array results:\n");
    for (int i = low; i <= high; i += step) {
        int* array = malloc(i * sizeof(int));
        genOrderedBackwards(array, i);
        int compares = 0;
        function(array, i, &compares);
        assert(isOrdered(array, i));
        printf("For %3d elements: %7d compares\n", i, compares);
        free(array);
    }

    printf("\nRandom order array results:\n");
    for (int i = low; i <= high; i += step) {
        int* array = malloc(i * sizeof(int));
        genRandom(array, i);
        int compares = 0;
        function(array, i, &compares);
        assert(isOrdered(array, i));
        printf("For %3d elements: %7d compares\n", i, compares);
        free(array);
    }

    printf("\n");
}

```

insertionsort.c

```

#include <lab3/sorts.h>

#include <string.h>

void insertionSort(int* data, int size, int* comps) {
    for (int i = 1; INC_COMPARES(comps) && i < size; i++) {
        int j = i - 1;
        while (INC_COMPARES(comps) && j >= 0 && INC_COMPARES(comps) && data[j] > data[i])
            j--;

        int element = data[i];
        memcpy(data + j + 2, data + j + 1, (i - j - 1) * sizeof(int));
        data[j + 1] = element;
    }
}

```

```
    }  
}
```

selectionsort.c

```
#include <lab3/sorts.h>  
  
void selectionSort(int* data, int size, int* comps) {  
    for (int i = 0; INC_COMPARES(comps) && i < size - 1; i++) {  
        int minIndex = i;  
        for (int j = i + 1; INC_COMPARES(comps) && j < size; j++) {  
            if (INC_COMPARES(comps) && data[j] < data[minIndex])  
                minIndex = j;  
        }  
  
        swap(data + i, data + minIndex);  
    }  
}
```

bubblesort.c

```
#include <lab3/sorts.h>  
  
void bubbleSort(int* data, int size, int* comps) {  
    for (int i = 0; INC_COMPARES(comps) && i < size; i++) {  
        for (int j = size - 1; INC_COMPARES(comps) && j > i; j--) {  
            if (INC_COMPARES(comps) && data[j] < data[j - 1])  
                swap(data + j, data + j - 1);  
        }  
    }  
}
```

bubblesortmod1.c

```
#include <lab3/sorts.h>  
  
#include <stdbool.h>  
  
void bubbleSortMod1(int* data, int size, int* comps) {  
    for (int i = 0; INC_COMPARES(comps) && i < size; i++) {  
        bool anyCompsDone = false;
```

```

    for (int j = size - 1; INC_COMPARES(comps) && j > i; j--) {
        if (INC_COMPARES(comps) && data[j] < data[j - 1]) {
            swap(data + j, data + j - 1);
            anyCompsDone = true;
        }
    }

    if (!anyCompsDone)
        break;
}
}

```

bubblesortmod2.c

```

#include <lab3/sorts.h>

#include <stdbool.h>

void bubbleSortMod2(int *data, int size, int *comps)
{
    for (int i = 0; INC_COMPARES(comps) && i < size; i++) {
        int k = size;

        for (int j = size - 1; INC_COMPARES(comps) && j > i; j--) {
            if (INC_COMPARES(comps) && data[j] < data[j - 1]) {
                swap(data + j, data + j - 1);
                k = j - 1;
            }
        }

        i = k;
    }
}

```

shellsort.c

```

#include <lab3/sorts.h>

#include <math.h>

void shellSort(int* data, int size, int* comps) {
    int t = log2(size) / log2(3) - 1;

```

```

t = INC_COMPARES(comps) && t < 0 ? 0 : t;

int h = 1;
for (int i = 0; INC_COMPARES(comps) && i < t; i++)
    h = h * 3 + 1;

while (INC_COMPARES(comps) && h >= 1) {
    for (int i = h; INC_COMPARES(comps) && i < size; i++) {
        int j = i;

        while (INC_COMPARES(comps) && j - h >= 0 && INC_COMPARES(comps) && data[j - h]
            ↪ > data[j]) {
            swap(data + j - h, data + j);
            j -= h;
        }
    }

    h = (h - 1) / 3;
}
}

```

hoarsort.c

```

#include <lab3/sorts.h>

#include <malloc.h>

void hoarSort(int *data, int size, int *comps) {
    if (INC_COMPARES(comps) && size <= 1) return;

    int i = 0, j = size - 1;
    int midElement = data[size / 2];
    while (INC_COMPARES(comps)) {
        if (INC_COMPARES(comps) && data[i] < midElement) {
            i++;
            continue;
        }

        if (INC_COMPARES(comps) && data[j] > midElement) {
            j--;
            continue;
        }
    }
}

```



```

        if (INC_COMPARES(comps) && i >= j)
            break;

        swap(data + (i++), data + (j--));
    }

    hoarSort(data, i, comps);
    hoarSort(data + i, size - i, comps);
}

```

heapsort.c

```

#include <lab3/sorts.h>

void heapSort(int* data, int size, int* comps) {

    // Создаём кучу
    for (int i = 1; INC_COMPARES(comps) && i < size; i++) {
        int currentIndex = i;
        int parentIndex = (currentIndex - 1) / 2;

        while (INC_COMPARES(comps) && currentIndex != 0 && INC_COMPARES(comps) &&
            ↪ data[parentIndex] < data[currentIndex]) {
            swap(data + parentIndex, data + currentIndex);
            currentIndex = parentIndex;
            parentIndex = (currentIndex - 1) / 2;
        }
    }

    // Удаляем из кучи элементы
    for (int i = 0; INC_COMPARES(comps) && i < size - 1; i++) {
        swap(data, data + size - 1 - i);

        int currentIndex = 0;
        int heapSize = size - i - 1;
        while (1) {
            int leftChildIndex = 2 * currentIndex + 1;
            int rightChildIndex = 2 * currentIndex + 2;

            if (INC_COMPARES(comps) && leftChildIndex < heapSize && INC_COMPARES(comps) &&
            ↪ rightChildIndex < heapSize) {
                if (INC_COMPARES(comps) && data[currentIndex] < data[leftChildIndex] &&
                    INC_COMPARES(comps) && data[leftChildIndex] > data[rightChildIndex]) {

```

```

        swap(data + leftChildIndex, data + currentIndex);
        currentIndex = leftChildIndex;
    } else if (INC_COMPARES(comps) && data[currentIndex] <
        ↪ data[rightChildIndex] &&
            INC_COMPARES(comps) && data[rightChildIndex] >
            ↪ data[leftChildIndex]) {
        swap(data + rightChildIndex, data + currentIndex);
        currentIndex = rightChildIndex;
    } else break;
} else if (INC_COMPARES(comps) && leftChildIndex < heapSize &&
    ↪ INC_COMPARES(comps) && data[currentIndex] < data[leftChildIndex]) {
    swap(data + leftChildIndex, data + currentIndex);
    break;
} else if (INC_COMPARES(comps) && rightChildIndex < heapSize &&
    ↪ INC_COMPARES(comps) && data[rightChildIndex] > data[rightChildIndex]) {
    swap(data + rightChildIndex, data + currentIndex);
    break;
} else break;
}
}
}

```

2. Результаты работы программы.

Результаты экспериментов для упорядоченных по возрастанию массивов

Сортировка	Количество элементов в массиве								
	5	10	15	20	25	30	35	40	45
Включением	13	28	43	58	73	88	103	118	133
Выбором	29	109	239	419	649	929	1259	1639	2069
Обменом	31	111	241	421	651	931	1261	1641	2071
Обменом 1	10	20	30	40	50	60	70	80	90
Обменом 2	11	21	31	41	51	61	71	81	91
Шелла	17	53	83	113	143	227	272	317	362
Хоара	44	115	192	282	373	461	561	666	771
Пирамидальная	50	147	264	403	551	705	871	1047	1229

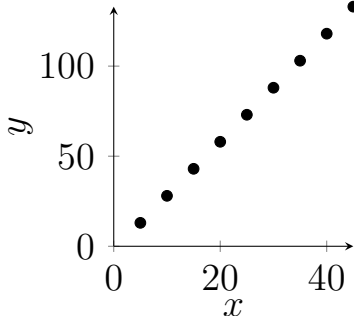
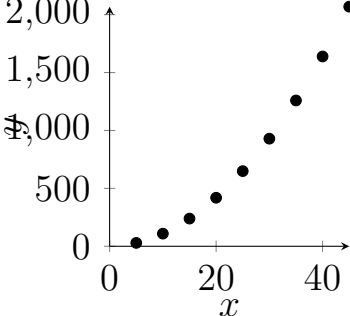
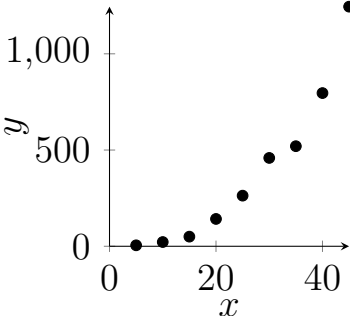
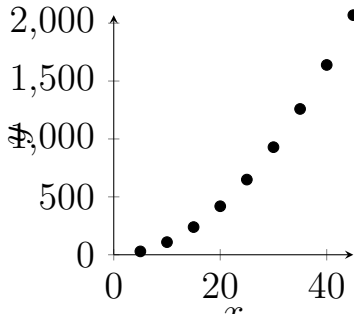
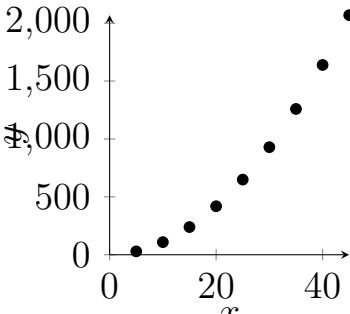
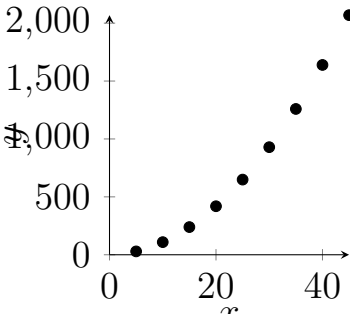
Результаты экспериментов для упорядоченных по убыванию массивов

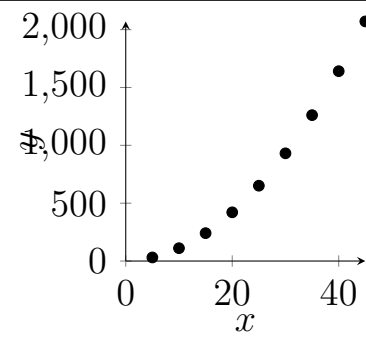
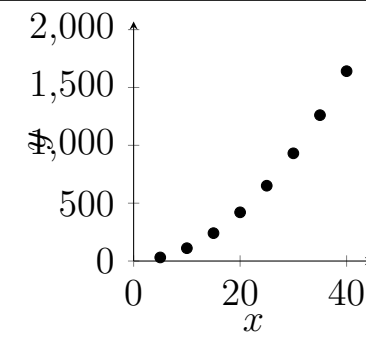
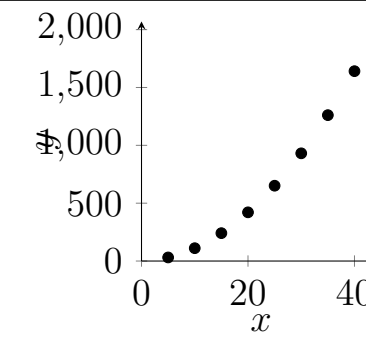
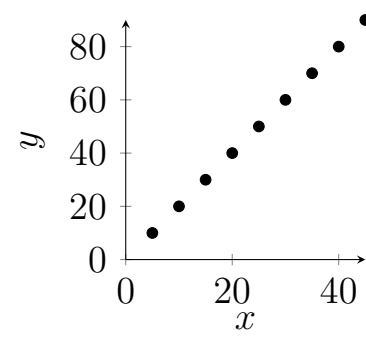
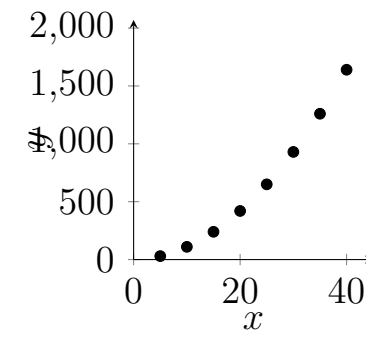
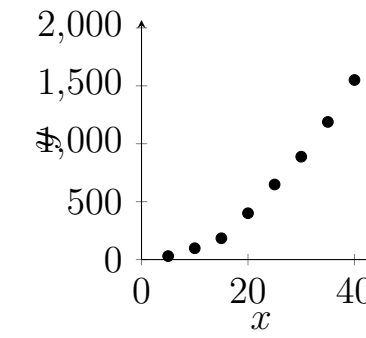
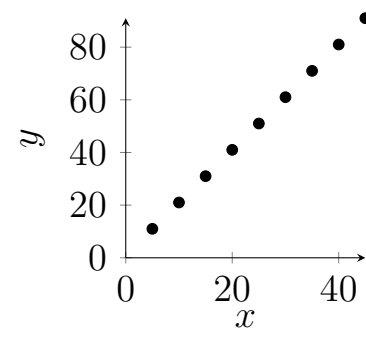
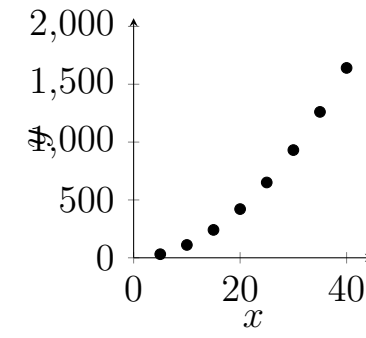
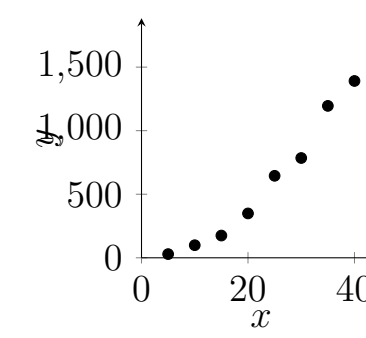
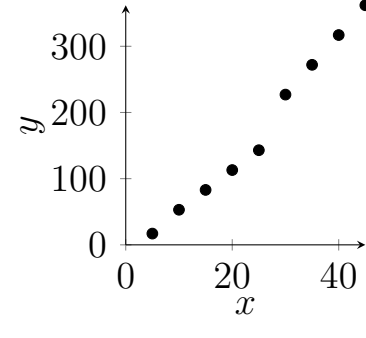
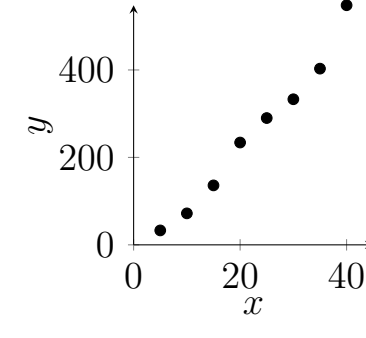
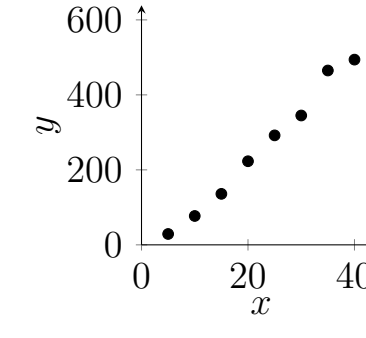
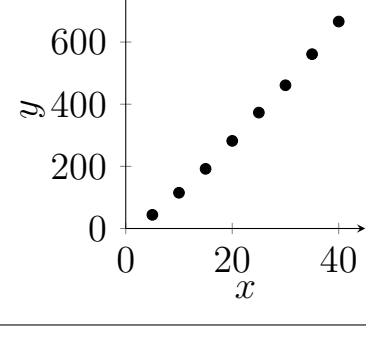
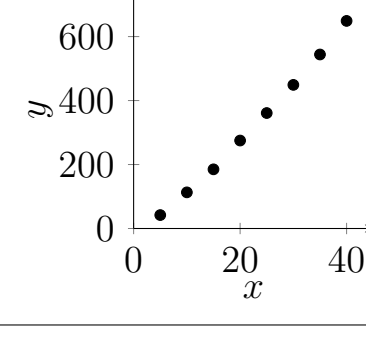
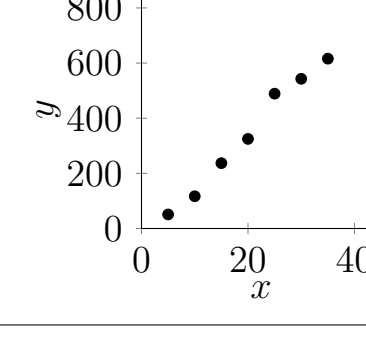
Сортировка	Количество элементов в массиве								
	5	10	15	20	25	30	35	40	45
Включением	29	109	239	419	649	929	1259	1639	2069
Выбором	29	109	239	419	649	929	1259	1639	2069
Обменом	31	111	241	421	651	931	1261	1641	2071
Обменом 1	30	110	240	420	650	930	1260	1640	2070
Обменом 2	31	111	241	421	651	931	1261	1641	2071
Шелла	33	72	136	234	290	333	403	548	535
Хоара	42	113	185	275	361	449	544	649	749
Пирамидальная	42	119	217	339	437	571	700	816	984

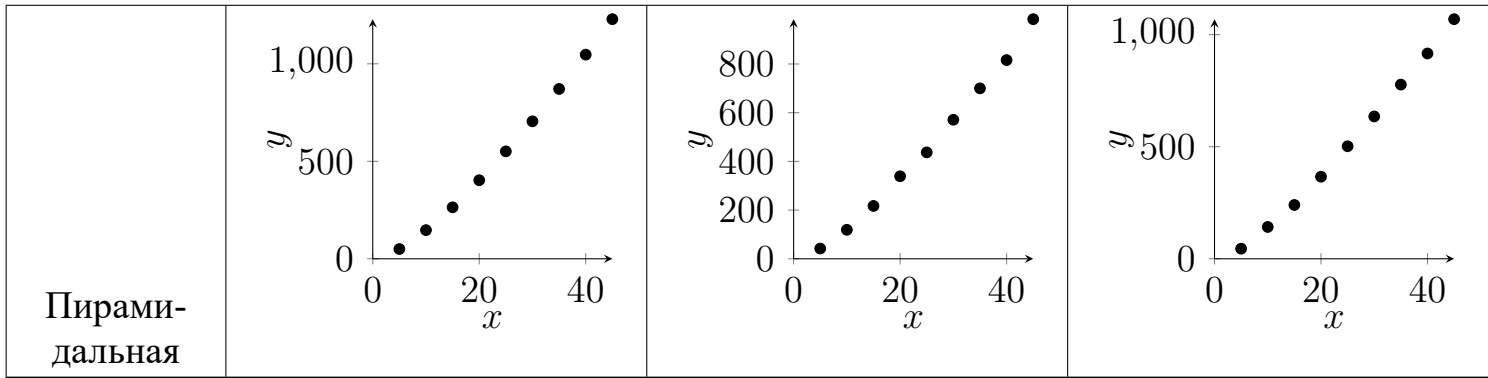
Результаты экспериментов для неупорядоченных массивов

Сортировка	Количество элементов в массиве								
	5	10	15	20	25	30	35	40	45
Включением	5	22	50	142	263	459	520	796	1245
Выбором	29	109	239	419	649	929	1259	1639	2069
Обменом	31	111	241	421	651	931	1261	1641	2071
Обменом 1	30	98	184	400	648	888	1188	1550	2068
Обменом 2	29	99	175	349	645	785	1195	1391	1885
Шелла	29	77	136	223	292	345	465	494	639
Хоара	51	117	237	325	489	543	616	869	829
Пирамидальная	45	142	240	366	502	635	777	916	1069

3. Графики зависимостей ФВС.

Сортировка	Упоряд. по возрастанию массив	Упоряд. по убыванию массив	Неупорядоченный массив
Включением			
Выбором			

Обменом			
Обменом 1			
Обменом 2			
Шелла			
Хоара			



4. Выводы по работе.

Сортировка вставками:

В лучшем случае, если массив отсортирован по возрастанию, $t = 1 + N \cdot 5$, в худшем случае, если массив отсортирован по убыванию, $t = 1 + N \cdot (1 + 3 \cdot (1 + 2 + 3 + \dots + N - 1) + 2 + (1 + 2 + 3 + \dots + N)) = 1 + N \cdot (3 + 4 \cdot \frac{N-1}{2}) = 1 + 5 \cdot N + 4 \cdot N^2$.

Для отсортированного по возрастанию массива: $\Omega(N)$

Для отсортированного по убыванию массива: $O(N^2)$

Для неотсортированного массива: $\Theta(N^2)$

Сортировка выбором:

$t = 1 + N \cdot (3 + 3 \cdot (N - 1 + N - 2 + \dots + 3 + 2 + 1)) = 1 + N \cdot (3 + \frac{3}{2}(N - 1)) = 1 + 3 \cdot N + \frac{3}{2}(N^2 - N) = 1 + \frac{3}{2} \cdot N + \frac{3}{2} \cdot N^2$.

Для отсортированного по возрастанию массива: $\Omega(N^2)$

Для отсортированного по убыванию массива: $O(N^2)$

Для неотсортированного массива: $\Theta(N^2)$

Обменом:

$t = 1 + N \cdot (1 + 3 \cdot (N - 1 + N - 2 + \dots + 3 + 2 + 1)) = 1 + N \cdot (1 + \frac{3}{2}(N - 1)) = 1 + N + \frac{3}{2}(N^2 - N) = 1 - \frac{1}{2} \cdot N + \frac{3}{2} \cdot N^2$.

Для отсортированного по возрастанию массива: $\Omega(N^2)$

Для отсортированного по убыванию массива: $O(N^2)$

Для неотсортированного массива: $\Theta(N^2)$

Обменом 1:

В лучшем случае, если массив отсортирован по возрастанию, $t = 5 + 3 \cdot (N - 1 + N - 2 + \dots + 3 + 2 + 1) = 3.5 + 1.5 \cdot N$, в худшем случае, если массив отсортирован по убыванию, $t = 1 + N \cdot (5 + 5 \cdot (N - 1 + N - 2 + \dots + 3 + 2 + 1)) = 1 + N \cdot (5 + \frac{5N-5}{2}) = 1 + N \cdot (2.5 + 2.5N) = 1 + 2.5N + 2.5N^2$.

Для отсортированного по возрастанию массива: $\Omega(N)$

Для отсортированного по убыванию массива: $O(N^2)$

Для неотсортированного массива: $\Theta(N^2)$

Обменом 2:

В лучшем случае, если массив отсортирован по возрастанию, $t = 5 + 3 \cdot (N - 1 +$

$N - 2 + \dots + 3 + 2 + 1) = 3.5 + 1.5 \cdot N$, в худшем случае, если массив отсортирован по неубыванию, $t = 1 + N \cdot (5 + 5 \cdot (N - 1 + N - 2 + \dots + 3 + 2 + 1)) = 1 + N \cdot (5 + \frac{5N-5}{2}) = 1 + N \cdot (2.5 + 2.5N) = 1 + 2.5N + 2.5N^2$.

Для отсортированного по возрастанию массива: $\Omega(N)$

Для отсортированного по убыванию массива: $O(N^2)$

Для неотсортированного массива: $\Theta(N^2)$

Сортировка Шелла:

$t = 3 + ((\log_3 N) - 1) + ((\log_3 N) - 1) \cdot ((N / (3 * h_i + 1)) \cdot (1 + 2 \cdot ((1 + (1 + 3 * h_i + 1) + (1 + 2 \cdot (3 * h_i + 1)) + (1 + 3 \cdot (3 * h_i + 1)) + \dots + (1 + N / (3 * h_i + 1)))))) + 1)$

Анализ сортировки Шелла математически сложен, в случае правильного выбора шагов порядок ФВС будет выглядеть как $O(N^{1.2})$ или $O(N(\log_2 N))$.

Сортировка Хоара:

$t = 4 + 4 \cdot N + 2 \cdot t(N/2)$

ФВС алгоритма не зависит от упорядоченности массива, она зависит от алгоритма нахождения разделителя. Поэтому для отсортированных по убыванию, возрастанию и неотсортированных массивов

$O(N^2)$ - если разделитель выбран неудачно

$\Omega(N \cdot \log N)$ - если разделитель выбран удачно

$\Theta(N \cdot \log N)$.

Пирамидальная сортировка:

$t = 2 + N \cdot (4 + 4 \cdot (\log_2 1 + \log_2 2 + \log_2 3 + \dots + \log_2 N)) + N \cdot (5 + 10 \cdot (\log_2 1 + \log_2 2 + \log_2 3 + \dots + \log_2 N))$

$O(N \cdot \log N)$

$\Omega(N \cdot \log N)$

$\Theta(N \cdot \log N)$.

Ссылка на репозиторий

Вывод: в ходе лабораторной работы изучили методы сортировки массивов и приобрели навыки в проведении сравнительного анализа различных методов сортировки.