Харківський національний університет імені В.Н. Каразіна
Навчально-науковий інститут комп’ютерних наук та штучного інтелекту

ЗВІТ

З ПРАКТИЧНОЇ РОБОТИ №7

дисципліна: «Алгоритмізація та програмування»

Виконав: студент 2 курсу групи КС22
Спеціальності 122 «Комп’ютерні науки»
Скрипняк Тарас Артемович
Прийняв: викладач
Олешко О.І.

Харків – 2024

Завдання №1: За формулами з https://en.wikipedia.org/wiki/Shellsort реалізувати програму, що розраховує РЯД проміжків для сортування Шелла:

1, 4, 9, 23, 57, 138, 326, 749, 1695, 3785, 8359, 18298, 39744.....

Користувач вводить розмірність масиву вхідних даних, програма розраховує та виводить проміжки для наступних алгоритмів:
- Shell
- Frank & Lazarus
- Papernov & Stasevich
- Pratt
- Knuth, 1973, [3] заснований на Pratt
- Incerpi & Sedgewick, 1985, [11] Knuth
- Sedgewick, 1982
- Sedgewick, 1986
- Tokuda

Проводимо порівняльне дослідження часу роботи алгоритму сортування Шелла при класичних проміжках та при використанні ряду Ciura.

Проводимо дослідження на тому самому випадковому файлі, що генерується як у попередньої роботі.
Розмір файлу – 4000 елементів.

Примітка: звіт по роботі повинен включати опис сортування Шелла, одержані результати дослідженнь, висновки. А ви маєте бути готові до 5 хв. доповіді за вашим звітом.

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

// general purpose
// used in sort complexity testing, to generate best, worst and avg case
scenarios

void printArray(int arr[], int size) {
    printf("[ ");
    for (int i = 0; i < size-1; i++)
        printf("%d, ", arr[i]);
    printf("%d ]\n", arr[size-1]);
}

void fillArray(int arr[], int size) {
    for (int i = 0; i < size; i++) arr[i] = i + 1;
}
```

```
void reverseArray(int arr[], int size) {
    for (int i = 0; i < size / 2; i++)
    {
        int temp = arr[i];
        arr[i] = arr[size - i - 1];
        arr[size - i - 1] = temp;
    }
}

void shuffleArray(int arr[], int size) {
    srand(time(0));
    for (int i = size - 1; i > 0; i--)
    {
        int j = rand() % (i + 1);
        int temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
    }
}

// shellsort gaps
// each one is named after it's first author (or in case of Sedgewick, with year
of publication)

void shellGaps(unsigned int gaps[], unsigned int gapCount, unsigned int
arraySize) {
    int i = 0;
    for (int g = arraySize / 2; g >= 1 && i < gapCount; i++, g /= 2)
    {
        gaps[i] = g;
    }

    // this is to fill excess space in the array
    if (i < gapCount - 1) for (i; i < gapCount; i++) gaps[i] = 0;
}

void frankGaps(unsigned int gaps[], unsigned int gapCount, unsigned int
arraySize) {
    for (int i = 1, g = 2 * (arraySize / 4) + 1; i <= gapCount; i++, g = 2 *
(arraySize / (int) pow(2, i+1)) + 1)
    {
        gaps[i-1] = g;
    }
}

void papernovGaps(unsigned int gaps[], unsigned int gapCount, unsigned) {
    gaps[gapCount - 1] = 1;
    for (int i = 1; i < gapCount; i++)
        gaps[gapCount-i-1] = pow(2, i) + 1;
}

void prattGaps(unsigned int gaps[], unsigned int gapCount, unsigned int
arraySize) {
    int i2 = 0, i3 = 0;
    int next2 = 2, next3 = 3;
    int current = 1;

    int temp[gapCount];
    int index = 0;

    temp[index++] = current;
    while (index < gapCount) {
        current = (next2 < next3) ? next2 : next3;
        temp[index++] = current;
```

```
        if (current == next2)
            next2 = temp[++i2] * 2;
        if (current == next3)
            next3 = temp[++i3] * 3;
    }

    for (int i = 0; i < gapCount; i++)
        gaps[i] = temp[gapCount - 1 - i];
}

void knuthGaps(unsigned int gaps[], unsigned int gapCount, unsigned int
arraySize) {
    int c = ceil((double)arraySize/3);
    for (int i = 0, g = (pow(3, i+1))/2; g <= c; i++, g = (pow(3, i+1))/2)
        gaps[gapCount-1-i] = g;
}

const unsigned int coprimes[] = { 1, 3, 7, 16, 41, 101, 247, 613, 1529, 3821,
9539, 23843, 59611, 149015, 372539, 931327, 2328307, 5820767, 14551919,
36379789, 90949471, 227373677, 568434193, 1421085473 };
void incerpiGaps(unsigned int gaps[], unsigned int gapCount, unsigned) {
    for (int k = 0; k < gapCount; k++) {
        int r = sqrt(2*k + sqrt(2*k));
        int b = 0.5*(r*r + r) - k;
        int mult = 1;

        for (int q = 0; q <= r; q++) {
            if (q == b) continue;
            mult *= coprimes[q];
        }

        gaps[gapCount-k-1] = mult;
    }
}

void sedgewick82Gaps(unsigned int gaps[], unsigned int gapCount, unsigned) {
    gaps[gapCount - 1] = 1;
    for (int i = 1; i < gapCount; i++)
    {
        gaps[gapCount-i-1] = pow(4, i) + 3 * pow(2, i-1) + 1;
    }
}

void sedgewick86Gaps(unsigned int gaps[], unsigned int gapCount, unsigned) {
    gaps[gapCount - 1] = 1;
    for (int i = 1; i < gapCount; i++)
    {
        gaps[gapCount-i-1] = i % 2 ?
            8*pow(2, i) - 6*pow(2, (i+1)/2) + 1 :
            9*(pow(2, i) - pow(2, i/2)) + 1;
    }
}

void tokudaGaps(unsigned int gaps[], unsigned int gapCount, unsigned) {
    for (int i = 0; i < gapCount; i++)
        gaps[gapCount-i-1] = ceil((pow(9.0/4, i+1) - 1) / (9.0/4 - 1));
}

const int ciura[] = { 1, 4, 10, 23, 57, 132, 301, 701, 1750 };
void ciuraGaps(unsigned int gaps[], unsigned int gapCount, unsigned int
arraySize) {
    for (int i = 0; i  < gapCount; i++)
        gaps[gapCount-i-1] = i < 9 ? ciura[i] : ciura[8] * pow(2.25, i-8);
}

// shellsort gaps array length calculators
```

```
// made for convenience

int shellCalc(unsigned int arraySize) {
    return (int)log2(arraySize);
}

int frankCalc(unsigned int arraySize) {
    int k = 1;
    for (int g = 2 * (arraySize / 4) + 1; g > 1; k++, g = 2 * ( arraySize /
(int)pow(2, k+1) ) + 1);
    return k;
}

int papernovCalc(unsigned int arraySize) {
    int i = 1;
    for (int g = pow(2, i) + 1; g < arraySize; g = pow(2, ++i) + 1);
    return i;
}

int prattCalc(unsigned int arraySize) {
    int count = 0;
    for (int a = 1; a < arraySize; a *= 2)
        for (int b = a; b < arraySize; b *= 3, count++);
    return count;
}

int knuthCalc(unsigned int arraySize) {
    int c = ceil((double)arraySize/3), i = 1;
    for (int g = (pow(3, i+1))/2; g <= c; i++, g = (pow(3, i+1))/2);
    return i;
}

int incerpiCalc(unsigned int arraySize) {
    int k = 0;
    for (int mult = 1; mult < arraySize; k++) {
        int r = sqrt(2*k + sqrt(2*k));
        int b = 0.5*(r*r + r) - k;
        mult = 1;
        for (int q = 0; q <= r; q++) {
            if (q == b) continue;
            mult *= coprimes[q];
        }
    }

    return k + 1;
}

int sedgewick82Calc(unsigned int arraySize) {
    int i = 1;
    for (int g = 4 + 3 + 1; g < arraySize; i++, g = pow(4, i) + 3 * pow(2, i-1)
+ 1);
    return i;
}

int sedgewick86Calc(unsigned int arraySize) {
    int i = 1;
    for (int g = i % 2 ?
            8*pow(2, i) - 6*pow(2, (i + 1)/2) + 1 :
            9*(pow(2, i) - pow(2, i / 2)) + 1;
        g < arraySize;
        i++, g = i % 2 ?
            8*pow(2, i) - 6*pow(2, (i + 1)/2) + 1 :
            9*(pow(2, i) - pow(2, i / 2)) + 1);

    return i;
}
```

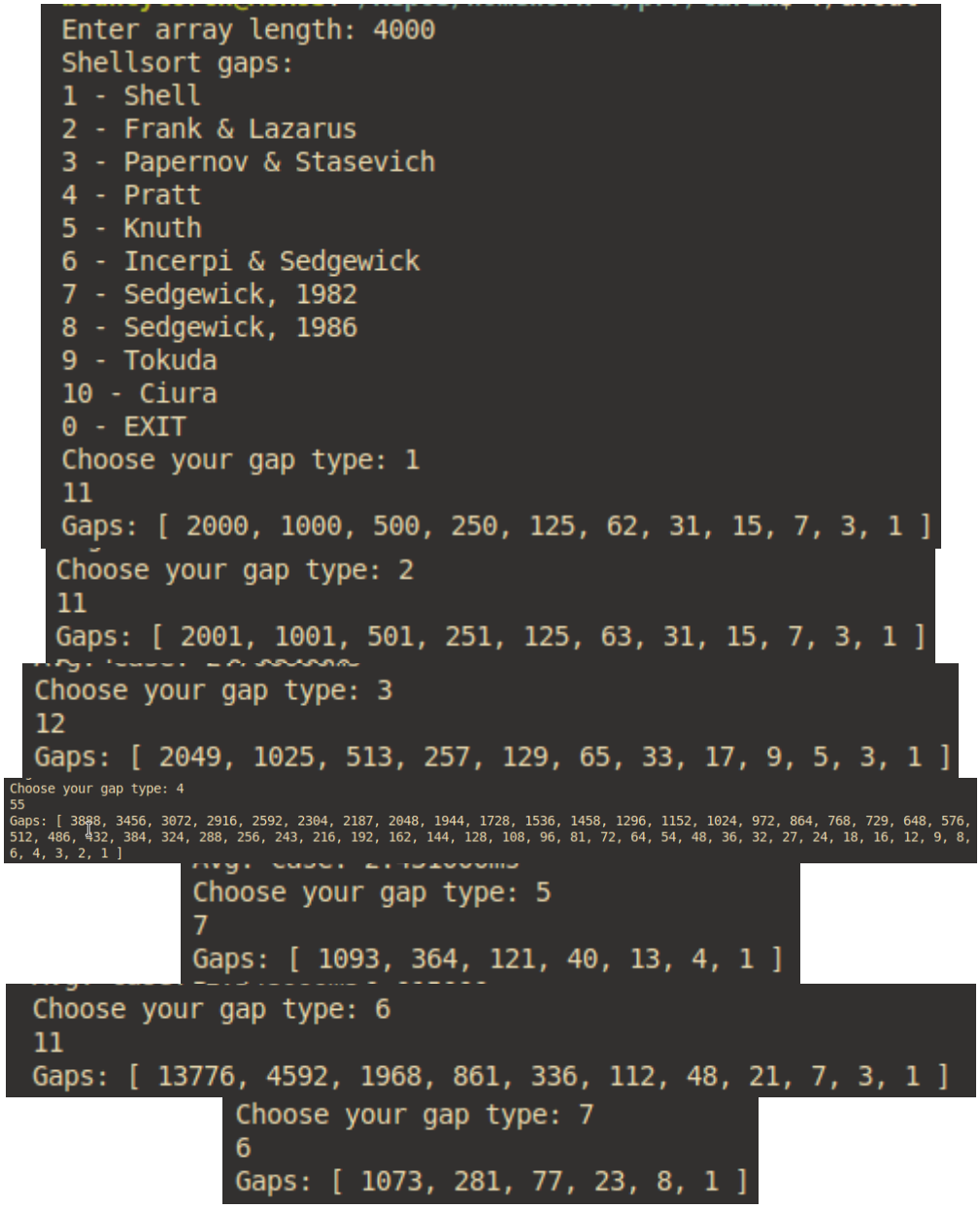
```
int tokudaCalc(unsigned int arraySize) {
    int i = 0;
    for (int g = 1; g < arraySize; i++, g = ceil((pow(9.0/4, i+1) - 1) / (9.0/4
- 1)));
    return i;
}

int ciuraCalc(unsigned int arraySize) {
    int i, h;
    for (i = 1; ciura[i] < arraySize && i < 9; i++);
    for (int h = 2.25*ciura[8]; h < arraySize; i++, h*=2.25);
    return i+1;
}

// shellsort

void shellSort(int array[], int size, unsigned int gaps[], unsigned int
gapCount) {
    for (int gapIndex = 0; gapIndex < gapCount; gapIndex++)
    {
        int gap = gaps[gapIndex];
        // gap optimization
        if (gap >= size || gap < 1) continue;

        for (int i = 0; i < size; i+= gap)
        {
            // insertion sort for the subarray
            int key = array[i];
            int j = i - gap;

            while (j >= 0 && array[j] > key) {
                // /* DEBUG */printf("%d %d %d\n", gap, i, j);
                array[j + gap] = array[j];
                j = j - gap;
            }
            array[j + gap] = key;
        }
    }
}

/*
    function to test shellsort with different gap generator function
    first arg is a gaps generator function, with following arguments:
    1 - int* arrayOfGaps
    2 - int gapCount
    3 - int arraySize

    second arg is the max amount of gaps to use with shellsort
    third arg is the size of the generated array to test on
*/
void testSort(void (*gapsFunction)(unsigned int*, unsigned int, unsigned int),
unsigned int gapCount, unsigned int size) {
    // /* DEBUG */ printf("%d\n", gapCount);
    unsigned int gaps[gapCount];
    int arr[size];

    gapsFunction(gaps, gapCount, size);
    printf("Gaps: ");
    printArray(gaps, gapCount);

    // BEST CASE ========
    fillArray(arr, size);
    // /* DEBUG */ printArray(arr, size);
    clock_t start = clock();
    shellSort(arr, size, gaps, gapCount);
```

```
    clock_t end = clock();
    // /* DEBUG */ printArray(arr, size);
    printf("Best case: %lfms\n", ((double)(end - start)) / 1000 );

    // WORST CASE =======
    reverseArray(arr, size);
    // /* DEBUG */ printArray(arr, size);
    start = clock();
    shellSort(arr, size, gaps, gapCount);
    end = clock();
    // /* DEBUG */ printArray(arr, size);
    printf("Worst case: %lfms\n", ((double)(end - start)) / 1000 );

    // AVG CASE =======
    shuffleArray(arr, size);
    // /* DEBUG */ printArray(arr, size);
    start = clock();
    shellSort(arr, size, gaps, gapCount);
    end = clock();
    // /* DEBUG */ printArray(arr, size);
    printf("Avg. case: %lfms\n", ((double)(end - start)) / 1000 );
}

int main() {
    int N;

    printf("Enter array length: ");
    scanf("%d", &N);
    if (N < 1) {
        printf("Arrays can't have negative length.");
        return 1;
    }

    int gapsType;
    printf("Shellsort gaps:\n");
    printf("1 - Shell\n");
    printf("2 - Frank & Lazarus\n");
    printf("3 - Papernov & Stasevich\n");
    printf("4 - Pratt\n");
    printf("5 - Knuth\n");
    printf("6 - Incerpi & Sedgewick\n");
    printf("7 - Sedgewick, 1982\n");
    printf("8 - Sedgewick, 1986\n");
    printf("9 - Tokuda\n");
    printf("10 - Ciura\n");
    printf("0 - EXIT\n");


    while (1) {
        printf("Choose your gap type: ");
        scanf("%d", &gapsType);

        switch (gapsType) {
            case 0: return 0;
            case 1: testSort(shellGaps, shellCalc(N), N); break;
            case 2: testSort(frankGaps, frankCalc(N), N); break;
            case 3: testSort(papernovGaps, papernovCalc(N), N); break;
            case 4: testSort(prattGaps, prattCalc(N), N); break;
            case 5: testSort(knuthGaps, knuthCalc(N), N); break;
            case 6: testSort(incerpiGaps, incerpiCalc(N), N); break;
            case 7: testSort(sedgewick82Gaps, sedgewick82Calc(N), N); break;
            case 8: testSort(sedgewick86Gaps, sedgewick86Calc(N), N); break;
            case 9: testSort(tokudaGaps, tokudaCalc(N), N); break;
            case 10: testSort(ciuraGaps, ciuraCalc(N), N); break;
            default: printf("Gap type is incorrect. Try again.\n");
        }
```

```
    }

    return 0;
}
```

Лістинг - вихідний код програми

Пояснення: Цей алгоритм Шеллсорт — це оптимізований варіант сортування вставками, який використовує декілька проміжків (gap), визначених масивом gaps[]. Спочатку сортуються елементи, розділені великим проміжком, що дозволяє швидко переміщати значення на великі відстані. Потім проміжки зменшуються, і сортування повторюється з меншими відстанями, поки не буде виконане звичайне сортування вставками із сусідніми елементами (gap = 1). Це забезпечує поступове впорядкування масиву, що покращує ефективність порівняно зі звичайним методом вставок.



```
Choose your gap type: 8
10
Gaps: [ 3905, 2161, 929, 505, 209, 109, 41, 19, 5, 1 ]
```

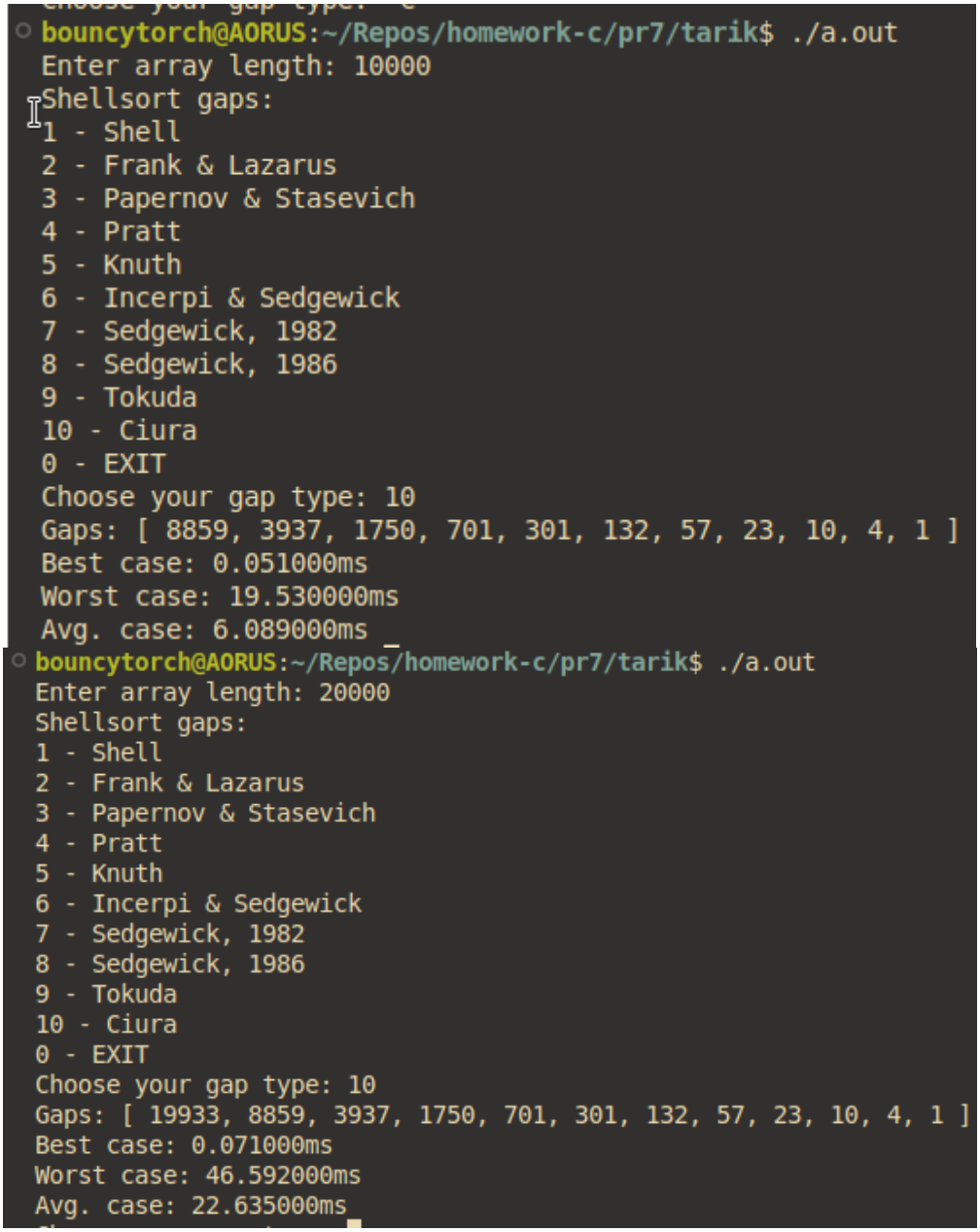
```
Choose your gap type: 9
10
Gaps: [ 2660, 1182, 525, 233, 103, 46, 20, 9, 4, 1 ]
```

```
Choose your gap type: 10
9
9
Gaps: [ 1750, 701, 301, 132, 57, 23, 10, 4, 1 ]
```

```
bouncytorch@AORUS:~/Repos/homework-c/pr7/tarik$ ./a.out
Enter array length: 5000
Shellsort gaps:
1 - Shell
2 - Frank & Lazarus
3 - Papernov & Stasevich
4 - Pratt
5 - Knuth
6 - Incerpi & Sedgewick
7 - Sedgewick, 1982
8 - Sedgewick, 1986
9 - Tokuda
10 - Ciura
0 - EXIT
Choose your gap type: 1
12
Gaps: [ 2500, 1250, 625, 312, 156, 78, 39, 19, 9, 4, 2, 1 ]
Best case: 0.036000ms
Worst case: 8.059000ms
Avg. case: 2.049000ms
bouncytorch@AORUS:~/Repos/homework-c/pr7/tarik$ ./a.out
Enter array length: 10000
Shellsort gaps:
1 - Shell
2 - Frank & Lazarus
3 - Papernov & Stasevich
4 - Pratt
5 - Knuth
6 - Incerpi & Sedgewick
7 - Sedgewick, 1982
8 - Sedgewick, 1986
9 - Tokuda
10 - Ciura
0 - EXIT
Choose your gap type: 1
13
Gaps: [ 5000, 2500, 1250, 625, 312, 156, 78, 39, 19, 9, 4, 2, 1 ]
Best case: 0.063000ms
Worst case: 14.275000ms
Avg. case: 5.327000ms
```

```
bouncytorch@AORUS:~/Repos/homework-c/pr7/tarik$ ./a.out
Enter array length: 30000
Shellsort gaps:
1 - Shell
2 - Frank & Lazarus
3 - Papernov & Stasevich
4 - Pratt
5 - Knuth
6 - Incerpi & Sedgewick
7 - Sedgewick, 1982
8 - Sedgewick, 1986
9 - Tokuda
10 - Ciura
0 - EXIT
Choose your gap type: 1
14
Gaps: [ 15000, 7500, 3750, 1875, 937, 468, 234, 117, 58, 29, 14, 7, 3, 1 ]
Best case: 0.113000ms
Worst case: 99.525000ms
Avg. case: 49.309000ms
```

```
bouncytorch@AORUS:~/Repos/homework-c/pr7/tarik$ ./a.out
Enter array length: 5000
Shellsort gaps:
1 - Shell
2 - Frank & Lazarus
3 - Papernov & Stasevich
4 - Pratt
5 - Knuth
6 - Incerpi & Sedgewick
7 - Sedgewick, 1982
8 - Sedgewick, 1986
9 - Tokuda
10 - Ciura
0 - EXIT
Choose your gap type: 10
Gaps: [ 3937, 1750, 701, 301, 132, 57, 23, 10, 4, 1 ]
Best case: 0.018000ms
Worst case: 7.683000ms
Avg. case: 1.608000ms
```



Рисунки 1-16 - результат виконання програми

N	Ciura's gaps			Shell's gaps		
	Best (ms)	Worst (ms)	Avg (ms)	Best (ms)	Worst (ms)	Avg (ms)
5000	0.018	7.683	1.6	0.036	8.059	2.049
10000	0.051	19.53	6.089	0.063	14.275	5.327
20000	0.071	46.592	22.635	0.113	99.525	49.309

Таблиця – аналіз ефективності проміжків Сіура проти Шелла

Висновок: Проміжки Сіура значно швидше ніж конвенціальні проміжки. Для того, щоб продовжити проміжки Сіура для більших масивів ми використали рекурсивну функцію h_k = h_(k-1) * 2.25.