# Отчет

## Quick Sort:

В целях подсчета времени выполнения программы в зависимости от длины динамического массива, изменим функцию main:

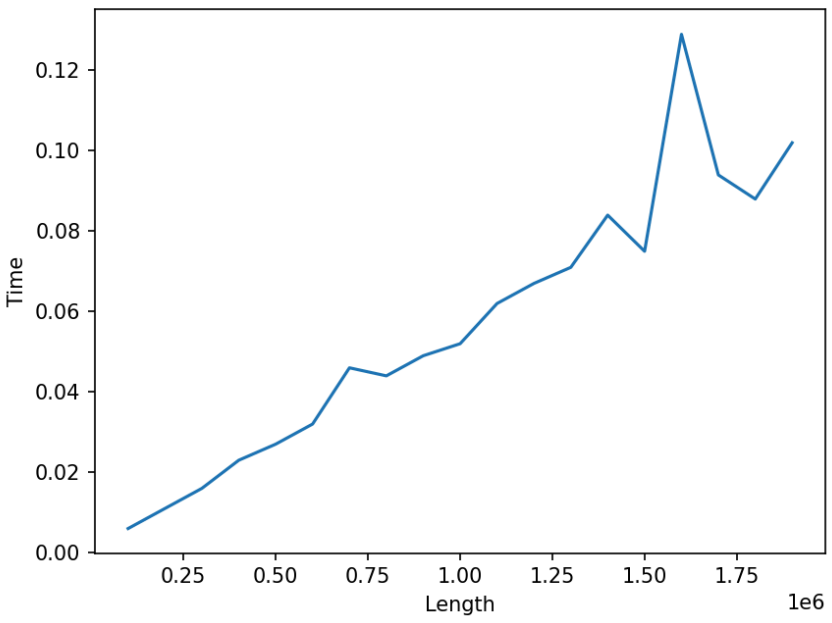
```
int main()
{
    for (int n = 1400000; n < 2000000; n = n + 10000) {
        srand(time(NULL));
        std::clock_t start = std::clock();

        int* array{ new int[n] };
        array = memory(n);

        fill(array, n);

        int size = sizeof(array);

        quickSort(array, 0, size - 1);

        std::clock_t end = std::clock();

        double search_time = static_cast<double>(end - start) / CLOCKS_PER_SEC;


        std::cout <<  "For n: " << n << " Time is: " << search_time << std::endl;

        dlt(array);
    }
```

При выполнении программы получим такие значения

```
For n: 1400000 Time is: 0.089
For n: 1410000 Time is: 0.115
For n: 1420000 Time is: 0.071
For n: 1430000 Time is: 0.07
For n: 1440000 Time is: 0.084
For n: 1450000 Time is: 0.081
For n: 1460000 Time is: 0.073
For n: 1470000 Time is: 0.089
For n: 1480000 Time is: 0.085
For n: 1490000 Time is: 0.076
For n: 1500000 Time is: 0.079
For n: 1510000 Time is: 0.082
For n: 1520000 Time is: 0.074
For n: 1530000 Time is: 0.078
For n: 1540000 Time is: 0.085
For n: 1550000 Time is: 0.073
For n: 1560000 Time is: 0.077
For n: 1570000 Time is: 0.086
For n: 1580000 Time is: 0.091
For n: 1590000 Time is: 0.107
For n: 1600000 Time is: 0.149
For n: 1610000 Time is: 0.096
For n: 1620000 Time is: 0.097
For n: 1630000 Time is: 0.087
For n: 1640000 Time is: 0.096
For n: 1650000 Time is: 0.087
For n: 1660000 Time is: 0.082
For n: 1670000 Time is: 0.09
For n: 1680000 Time is: 0.088
```

```
For n: 1680000 Time is: 0.088
For n: 1690000 Time is: 0.094
For n: 1700000 Time is: 0.086
For n: 1710000 Time is: 0.1
For n: 1720000 Time is: 0.09
For n: 1730000 Time is: 0.121
For n: 1740000 Time is: 0.111
For n: 1750000 Time is: 0.103
For n: 1760000 Time is: 0.101
For n: 1770000 Time is: 0.107
For n: 1780000 Time is: 0.108
For n: 1790000 Time is: 0.108
For n: 1800000 Time is: 0.103
For n: 1810000 Time is: 0.099
For n: 1820000 Time is: 0.11
For n: 1830000 Time is: 0.106
For n: 1840000 Time is: 0.119
For n: 1850000 Time is: 0.099
For n: 1860000 Time is: 0.166
For n: 1870000 Time is: 0.191
For n: 1880000 Time is: 0.121
For n: 1890000 Time is: 0.109
For n: 1900000 Time is: 0.117
For n: 1910000 Time is: 0.117
For n: 1920000 Time is: 0.108
For n: 1930000 Time is: 0.151
For n: 1940000 Time is: 0.109
For n: 1950000 Time is: 0.179
For n: 1960000 Time is: 0.148
For n: 1970000 Time is: 0.134
For n: 1980000 Time is: 0.277
For n: 1990000 Time is: 0.243
```

Представим таблицу значений в виде диаграммы:



## Bubble Sort

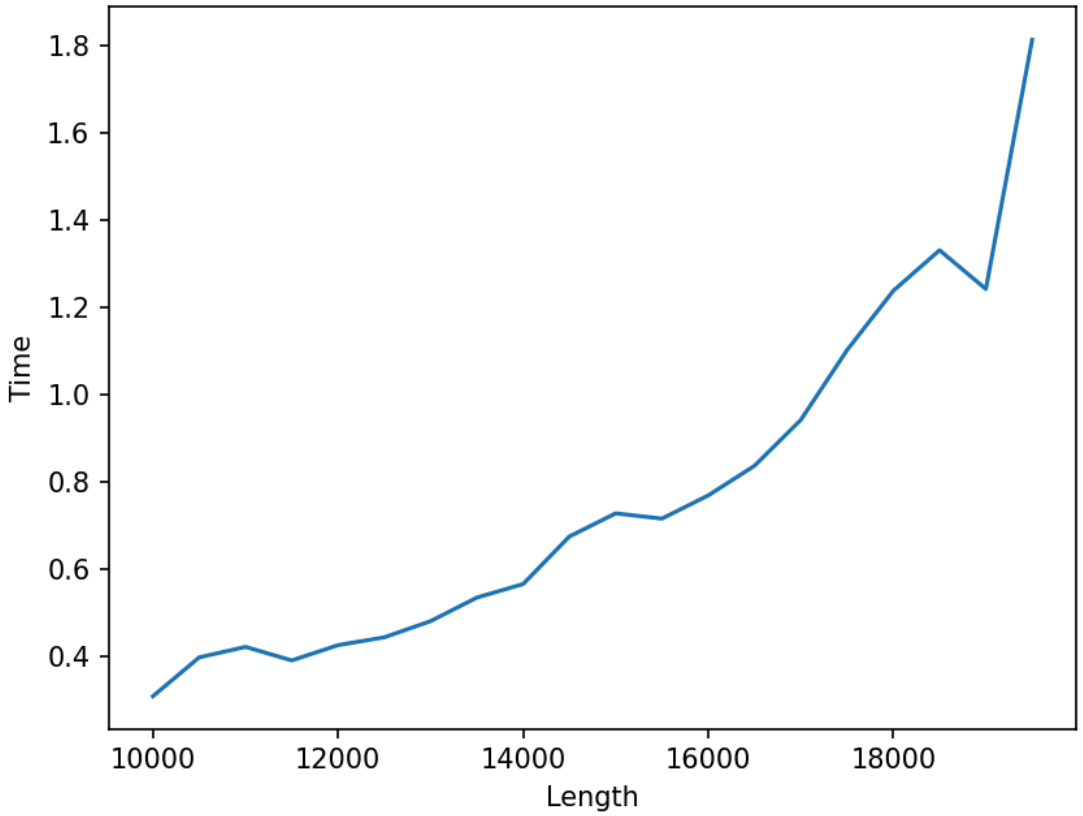
В целях подсчета времени выполнения программы в зависимости от длины динамического массива, изменим функцию main:

```
int main() {
    srand(time(NULL));

    for (int n = 10000; n < 20000; n = n + 500) {
        std::clock_t start = std::clock();

        int* array{ new int[n] };
        array = memory(n);

        fill(array, n);

        BubbleSort(array, n);


        std::clock_t end = std::clock();

        double search_time = static_cast<double>(end - start) / CLOCKS_PER_SEC;


        std::cout << n << " " << search_time << std::endl;

        dlt(array);
    }

    return 0;
```

При выполнении программы получим такие значения:

```
10000 0.31
10500 0.399
11000 0.423
11500 0.392
12000 0.427
12500 0.445
13000 0.482
13500 0.536
14000 0.567
14500 0.676
15000 0.729
15500 0.717
16000 0.77
16500 0.838
17000 0.943
17500 1.103
18000 1.239
18500 1.332
19000 1.243
19500 1.814
```

Представим таблицу значений в виде диаграммы:




## Counting Sort

В целях подсчета времени выполнения программы в зависимости от длины динамического массива, изменим функцию main:

```cpp
int main(){
    for (int n = 1000; n < 5000; n = n + 100) {
        std::clock_t start = std::clock();

        int* array{ new int[n] };
        array = memory(n);

        fill(array, n);

        countSort(array, n);

        std::clock_t end = std::clock();

        double search_time = static_cast<double>(end - start) / CLOCKS_PER_SEC;

        std::cout << n << " " << search_time << std::endl;

        dlt(array);
    }

    return 0;
}
```

При выполнении программы получим такие значения:

```
6500 0.105
7000 0.167
7500 0.175
8000 0.171
8500 0.198
9000 0.21
9500 0.253
10000 0.267
10500 0.294
11000 0.324
11500 0.366
12000 0.408
12500 0.51
13000 0.863
13500 0.532
14000 0.605
14500 0.697
15000 0.688
15500 1.233
16000 0.927
16500 1.113
17000 1.262
17500 1.116
18000 1.151
18500 1.278
19000 1.222
19500 1.373
```

Построим график: