

**DATA STRUCTURES II**

**LAB 1**

**Names:**

1- Amr Yasser Imam – 6772

2- Marwan Khaled Mohamed – 7020

3- Begad Wael – 6718

**Repository address:**

<https://github.com/AMR-21/Sorting-Algorithms>

**­­1- Time comparison**

On testing different arrays of unique integers with different size (100 – 1000 – 5000 – 10000 - 25000 – 50000 – 100000), the following results for (Quick Sort – Merge Sort – Selection Sort – Insertion Sort) was obtained as shown in TAB.1.1.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Algorithm – Size | 100 | 1000 | 5000 | 10000 | 25000 | 50000 | 100000 | Time (ms) |
| Quick Sort | 0 | 0.99 | 7.99 | 18.99 | 46 | 101.01 | 211 |
| Merge Sort | 1 | 2.99 | 14 | 33.01 | 81 | 179.99 | 365.99 |
| Selection Sort | 0 | 19 | 486.26 | 1906.08 | 12687.45 | 50194.72 | 213170 |
| Insertion Sort | 0 | 25 | 623.02 | 2726.11 | 19392.05 | 74761.92 | 322379.9 |

**TAB.1.1. Running times for different algorithms**

The following FIG.1.1. and FIG.1.2. represent the plots of the running time for each algorithm versus the array size

Chart, line chart

Description automatically generated Chart, line chart

Description automatically generated

**FIG.1.1. Plots for Merge, Quick, Selection, and Insertion sorting algorithms FIG.1.2. Plot showing Quick sort algorithm curve**

The following FIG.1.3. and FIG.1.4. are the results captured inside the program

Text

Description automatically generated Text

Description automatically generated

**FIG.1.3. Test cases (100 – 1000 – 5000 – 10000) FIG.1.4. Test cases (25000 – 50000 - 100000)**

**2- Quick Select Algorithm**

We implemented a function called quickSelect(list,k,left,right) that take the array and the order of item to be found, afterwards, we generated a random array with (name arr and size = 50).

arr = [7838, 5511, 7617, 2935, 4619, 5409, 8602, 1042, 5390, 45, 8110, 3545, 5761, 1666, 5165, 7909, 5178, 2881, 7013, 1104, 9763, 4813, 7043, 7048, 4996, 416, 7076, 5565, 4501, 4680, 5827,

     644, 4566, 3433, 8059, 7132, 780, 7587, 9155, 302, 2742, 141, 7106, 2507, 367, 2384, 6272, 323, 148, 930]

**2.1 Test case**

We tested the function by searching for the 8th smallest element by calling it with the following call quickSelect(arr,8,0,len(arr)-1). The result was 644 and to make sure it’s the correct element we sorted the array as shown in FIG.2.1.1. and indeed, we found that the 8th smallest element is 644 as shown in FIG.2.1.2..

quickSelect(arr,8,0,len(arr)-1)

Text

Description automatically generated

**FIG.2.1.1. The sorted array shows that 644 is the 8th smallest element**

Graphical user interface, text

Description automatically generated

**FIG.2.1.2. 8th smallest element in the array**

**3- Hybrid Merge and Selection Algorithm**