Data structure project

Mahmoud Abdalla Mohasseb Abdelrahim

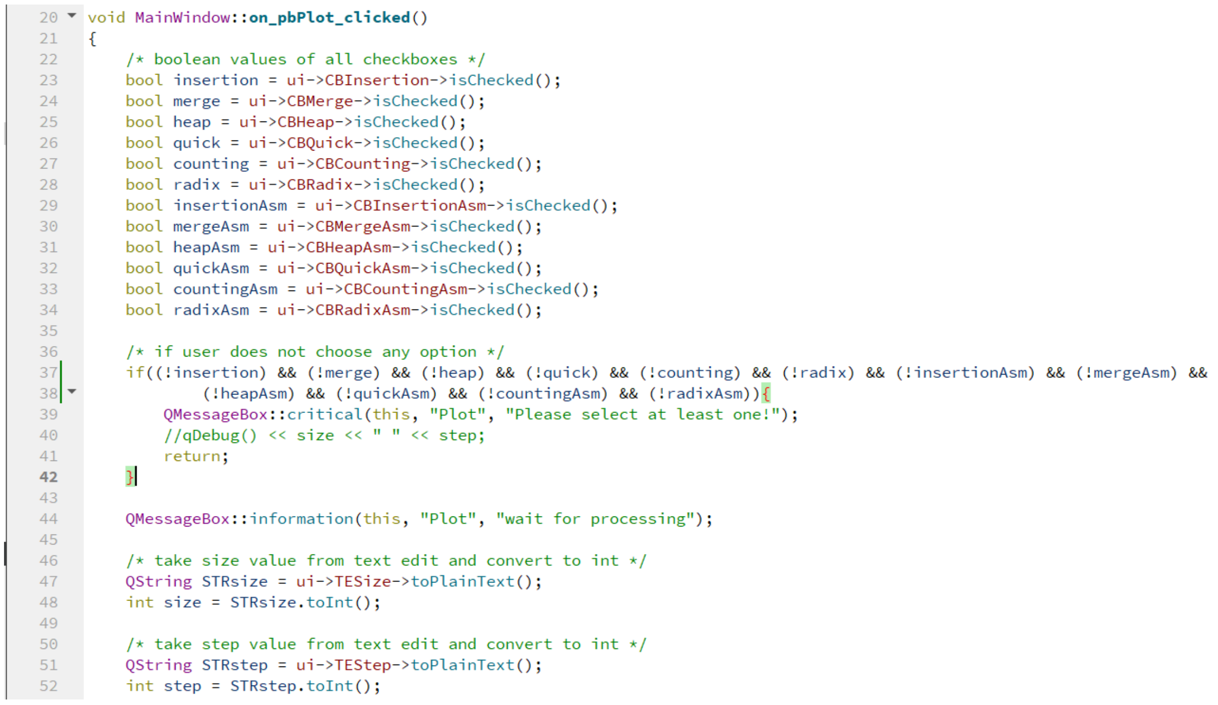
20P2787

Introduction:

This project explains steps sort take to sort numbers. Steps differ based on size and sort type. I explained Insertion, Merge, Heap, Quick, Counting and Radix sort and there asymptotic notations**. I used C++ and Qt** for making project

Explanation of Code:

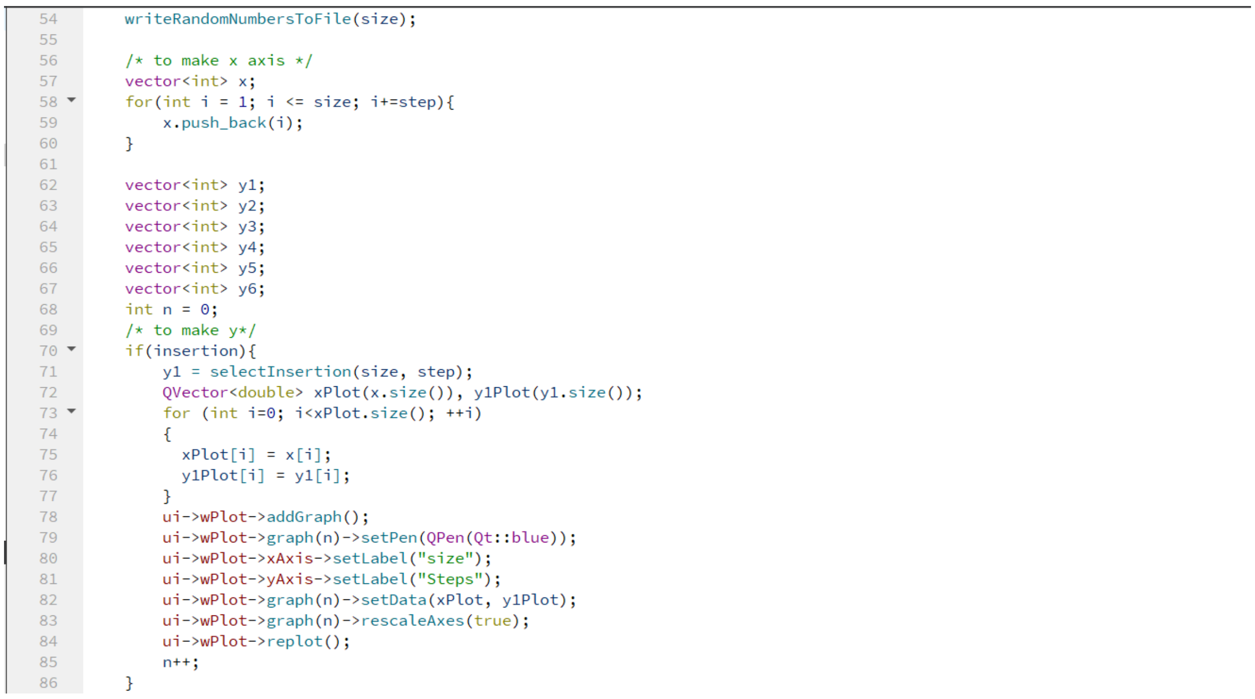
We will go through project step by step and explain things



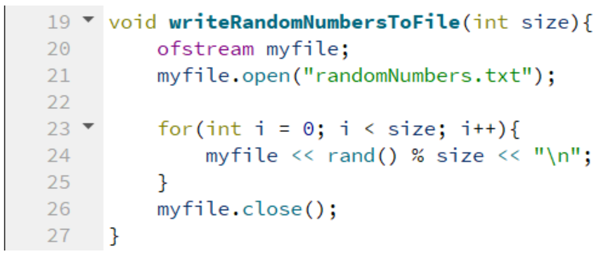
```cpp
void MainWindow::on_pbPlot_clicked()
{
    /* boolean values of all checkboxes */
    bool insertion = ui->CBInsertion->isChecked();
    bool merge = ui->CBMerge->isChecked();
    bool heap = ui->CBHeap->isChecked();
    bool quick = ui->CBQuick->isChecked();
    bool counting = ui->CBCounting->isChecked();
    bool radix = ui->CBRadix->isChecked();
    bool insertionAsm = ui->CBInsertionAsm->isChecked();
    bool mergeAsm = ui->CBMergeAsm->isChecked();
    bool heapAsm = ui->CBHeapAsm->isChecked();
    bool quickAsm = ui->CBQuickAsm->isChecked();
    bool countingAsm = ui->CBCountingAsm->isChecked();
    bool radixAsm = ui->CBRadixAsm->isChecked();

    /* if user does not choose any option */
    if((!insertion) && (!merge) && (!heap) && (!quick) && (!counting) && (!radix) && (!insertionAsm) && (!mergeAsm) &&
            (!heapAsm) && (!quickAsm) && (!countingAsm) && (!radixAsm)){
        QMessageBox::critical(this, "Plot", "Please select at least one!");
        //qDebug() << size << " " << step;
        return;
    }

    QMessageBox::information(this, "Plot", "wait for processing");

    /* take size value from text edit and convert to int */
    QString STRsize = ui->TESize->toPlainText();
    int size = STRsize.toInt();

    /* take step value from text edit and convert to int */
    QString STRstep = ui->TEStep->toPlainText();
    int step = STRstep.toInt();
```

This function is called when button (plot) clicked.

- we check for check boxes to which sort and asymptotic selected, then check if no check box is selected message will appear to tell user to select at least one. If user select at least one check box then message wait for processing appear to wait until plotting.
- taking value of size and step from text edit and convert them to int.

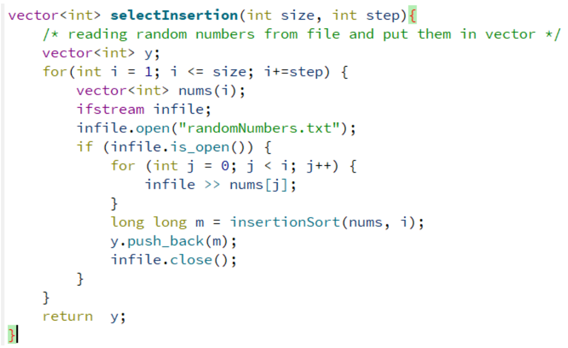


```cpp
    writeRandomNumbersToFile(size);

    /* to make x axis */
    vector<int> x;
    for(int i = 1; i <= size; i+=step){
        x.push_back(i);
    }

    vector<int> y1;
    vector<int> y2;
    vector<int> y3;
    vector<int> y4;
    vector<int> y5;
    vector<int> y6;
    int n = 0;
    /* to make y*/
    if(insertion){
        y1 = selectInsertion(size, step);
        QVector<double> xPlot(x.size()), y1Plot(y1.size());
        for (int i=0; i<xPlot.size(); ++i)
        {
          xPlot[i] = x[i];
          y1Plot[i] = y1[i];
        }
        ui->wPlot->addGraph();
        ui->wPlot->graph(n)->setPen(QPen(Qt::blue));
        ui->wPlot->xAxis->setLabel("size");
        ui->wPlot->yAxis->setLabel("Steps");
        ui->wPlot->graph(n)->setData(xPlot, y1Plot);
        ui->wPlot->graph(n)->rescaleAxes(true);
        ui->wPlot->replot();
        n++;
    }
```



```cpp
void writeRandomNumbersToFile(int size){
    ofstream myfile;
    myfile.open("randomNumbers.txt");

    for(int i = 0; i < size; i++){
        myfile << rand() % size << "\n";
    }
    myfile.close();
}
```

- we call writeRandomNumbersToFile(size), it takes size and write random numbers all of them smaller than size. It writes nth numbers equals to size.
- Making vector called x its value is size of different array sorted (ex: if size = 10000, step = 50 -> x contains 1, 51, 101,…., 9951 to help us in plotting
- Making 6 vectors to store m steps taken at each size
- Here, we check for insertion check box if selected, we call selectInsertion which takes size and steps and return vector contains steps taken to sort different sizes of vectors.
- In selectInsertion we make vectors of numbers in different sizes and write numbers to vector and give vector to function that return steps token to sort vector and push steps in vector y that we will return it in last of function



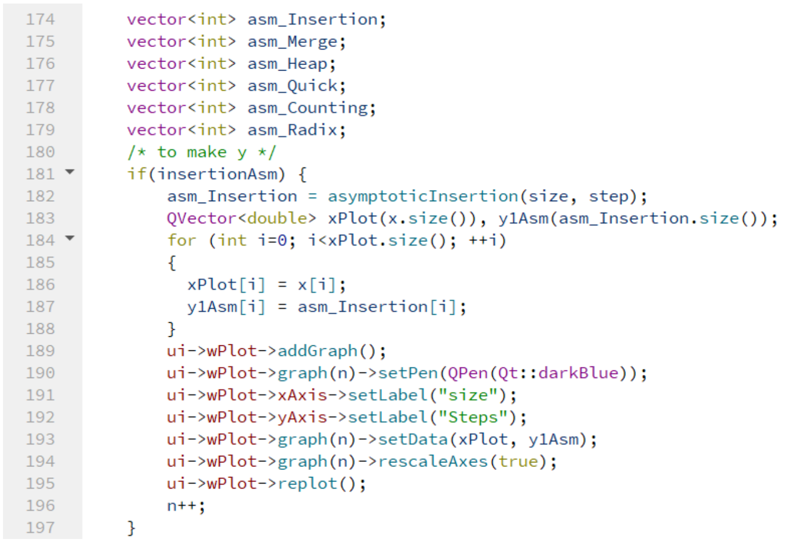
```cpp
vector<int> selectInsertion(int size, int step){
    /* reading random numbers from file and put them in vector */
    vector<int> y;
    for(int i = 1; i <= size; i+=step) {
        vector<int> nums(i);
        ifstream infile;
        infile.open("randomNumbers.txt");
        if (infile.is_open()) {
            for (int j = 0; j < i; j++) {
                infile >> nums[j];
            }
            long long m = insertionSort(nums, i);
            y.push_back(m);
            infile.close();
        }
    }
    return  y;
}
```

- Code of insertionSort in algorithms.c and I use the same function used in lap
- After selectInsertion() we make Qvector of x and y (x contains sizes, y contains steps of sizes) and put vector to Qvector because function responsible to plot in QT takes QVector of doubles as parameter.

- We have variable n so every graph we need to plot has its value in insertion its value is zero then after plotting we will increment by one
- addGraph(n) used to add graph and set color of pen here in insertion is blue
- make labels of x and y axis and plot.
- rescaleAxes(true) is used to make graph contains all points.

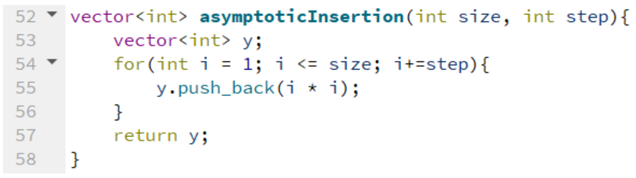
That steps will be repeated with merge, heap, quick, counting and radix sort

After checking for check boxes of sort and plot them we check for asymptotic notations



```
    vector<int> asm_Insertion;
    vector<int> asm_Merge;
    vector<int> asm_Heap;
    vector<int> asm_Quick;
    vector<int> asm_Counting;
    vector<int> asm_Radix;
    /* to make y */
    if(insertionAsm) {
        asm_Insertion = asymptoticInsertion(size, step);
        QVector<double> xPlot(x.size()), y1Asm(asm_Insertion.size());
        for (int i=0; i<xPlot.size(); ++i)
        {
          xPlot[i] = x[i];
          y1Asm[i] = asm_Insertion[i];
        }
        ui->wPlot->addGraph();
        ui->wPlot->graph(n)->setPen(QPen(Qt::darkBlue));
        ui->wPlot->xAxis->setLabel("size");
        ui->wPlot->yAxis->setLabel("Steps");
        ui->wPlot->graph(n)->setData(xPlot, y1Asm);
        ui->wPlot->graph(n)->rescaleAxes(true);
        ui->wPlot->replot();
        n++;
    }
```

- we make six vectors to store asymptotic value of each sort
- we check for insertionAsm, if checked then we will call asymptoticInsertion(size, step)



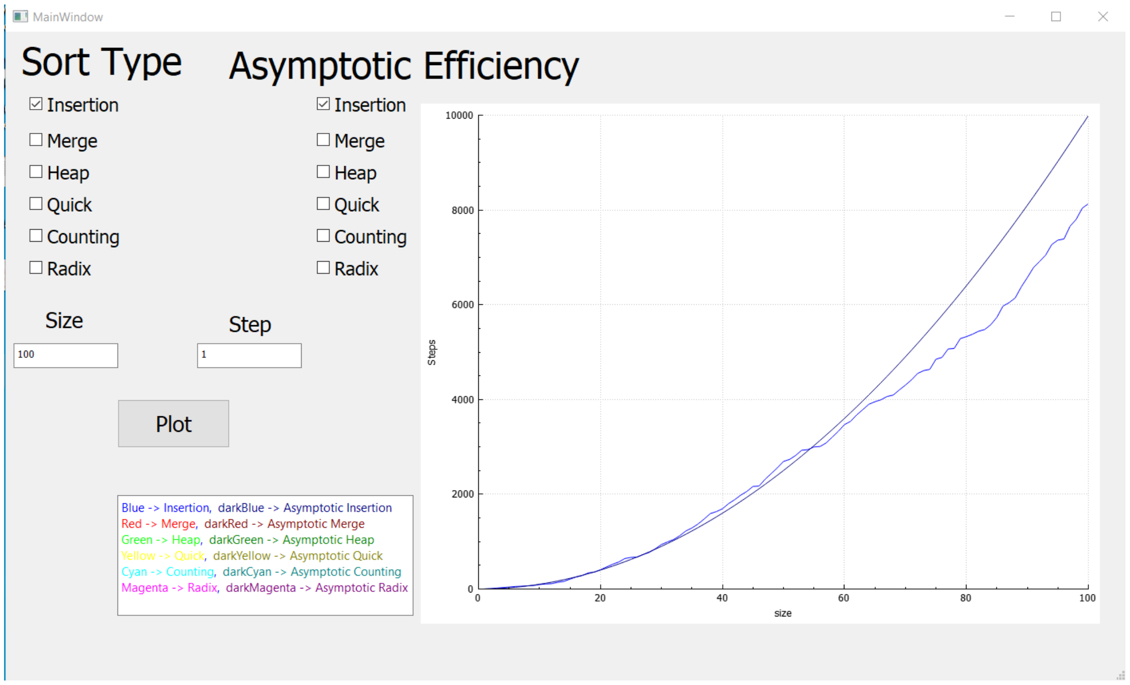
```
vector<int> asymptoticInsertion(int size, int step){
    vector<int> y;
    for(int i = 1; i <= size; i+=step){
        y.push_back(i * i);
    }
    return y;
}
```

- We make vector y to sore value of each size = size * size and return vector y
- Then we will repeat same steps in insertion for plotting graph

That steps will be repeated for merge, heap, quick, counting and radix sort asymptotic.

Screen shots of program:

Comparison between actual steps of sort and its asymptotic notation



Sort Type: [✓ Insertion] Merge, Heap, Quick, Counting, Radix

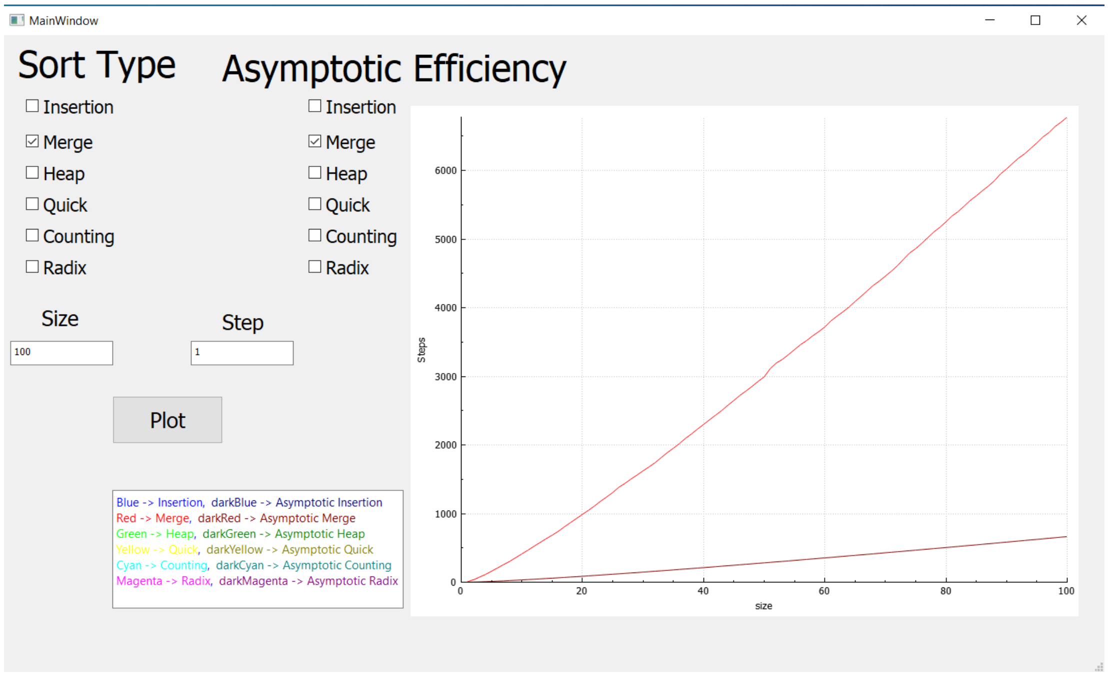
Asymptotic Efficiency: [✓ Insertion] Merge, Heap, Quick, Counting, Radix

Size: 100

Step: 1

Blue -> Insertion, darkBlue -> Asymptotic Insertion
Red -> Merge, darkRed -> Asymptotic Merge
Green -> Heap, darkGreen -> Asymptotic Heap
Yellow -> Quick, darkYellow -> Asymptotic Quick
Cyan -> Counting, darkCyan -> Asymptotic Counting
Magenta -> Radix, darkMagenta -> Asymptotic Radix

Here we compare between asymptotic notation of Insertion sort(n * n) and actual steps taken by insertion sort to sort numbers



Sort Type: Insertion, [✓ Merge] Heap, Quick, Counting, Radix

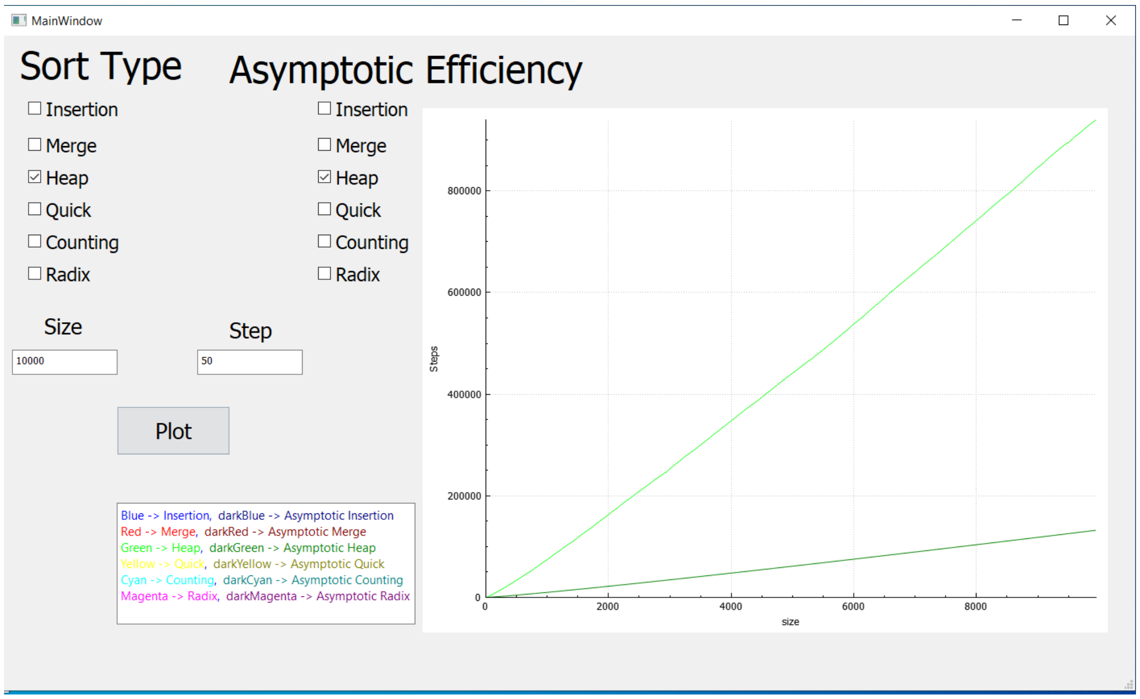
Asymptotic Efficiency: Insertion, [✓ Merge] Heap, Quick, Counting, Radix

Size: 100

Step: 1

Blue -> Insertion, darkBlue -> Asymptotic Insertion
Red -> Merge, darkRed -> Asymptotic Merge
Green -> Heap, darkGreen -> Asymptotic Heap
Yellow -> Quick, darkYellow -> Asymptotic Quick
Cyan -> Counting, darkCyan -> Asymptotic Counting
Magenta -> Radix, darkMagenta -> Asymptotic Radix

Here we compare between asymptotic notation of Merge sort(n * log n) and actual steps taken by Merge sort to sort numbers



Sort Type

☐ Insertion
☐ Merge
☑ Heap
☐ Quick
☐ Counting
☐ Radix

Asymptotic Efficiency

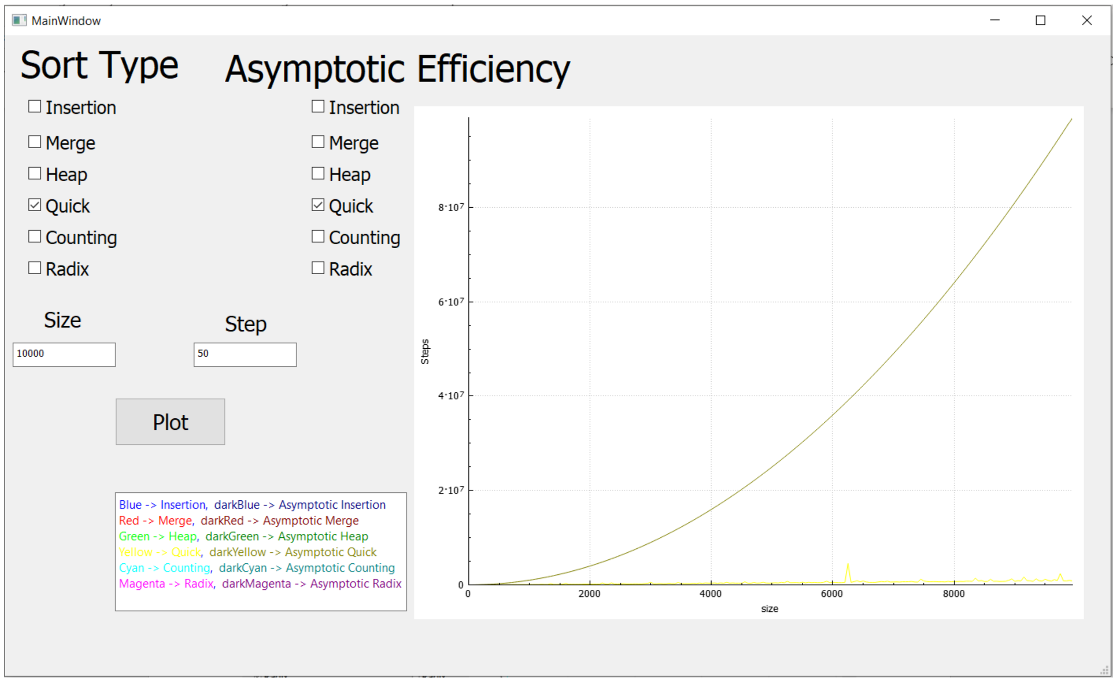
☐ Insertion
☐ Merge
☑ Heap
☐ Quick
☐ Counting
☐ Radix

Size: 10000

Step: 50

Blue -> Insertion, darkBlue -> Asymptotic Insertion
Red -> Merge, darkRed -> Asymptotic Merge
Green -> Heap, darkGreen -> Asymptotic Heap
Yellow -> Quick, darkYellow -> Asymptotic Quick
Cyan -> Counting, darkCyan -> Asymptotic Counting
Magenta -> Radix, darkMagenta -> Asymptotic Radix

Here we compare between asymptotic notation of heap sort(n * log n) and actual steps taken by heap sort to sort numbers



Sort Type

☐ Insertion
☐ Merge
☐ Heap
☑ Quick
☐ Counting
☐ Radix

Asymptotic Efficiency

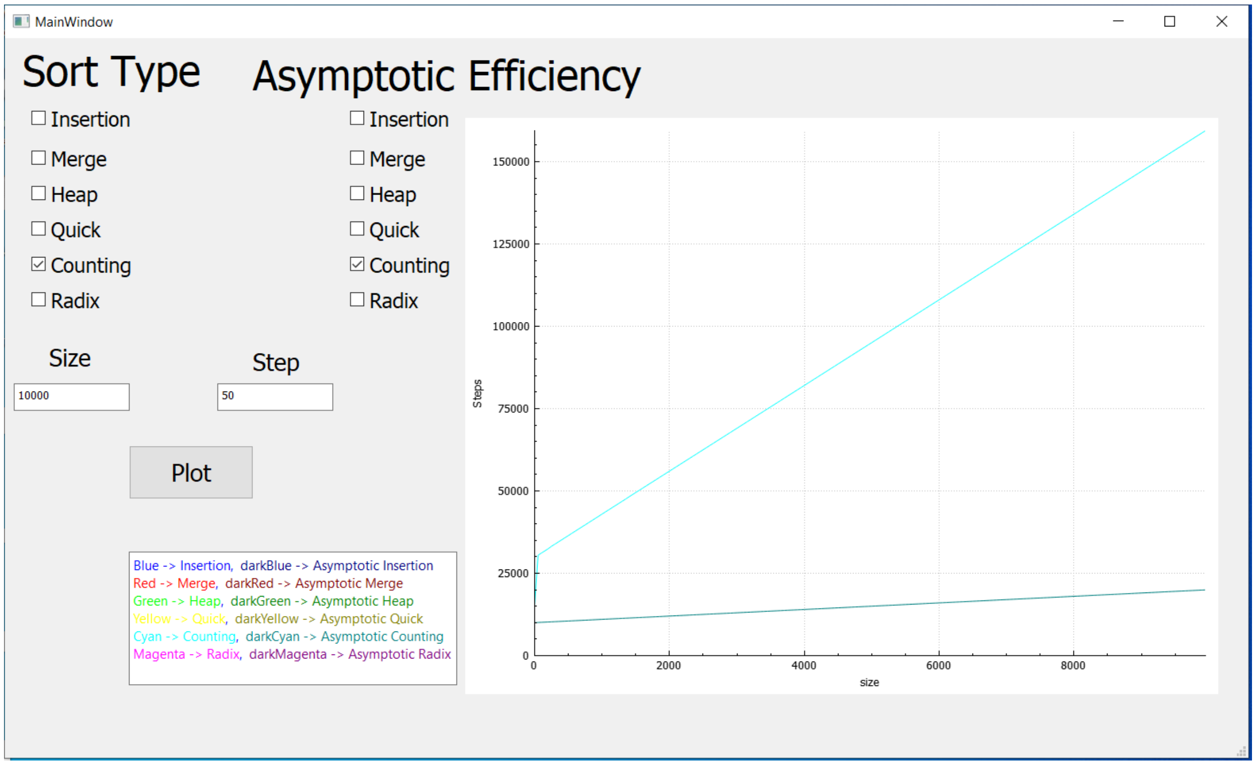
☐ Insertion
☐ Merge
☐ Heap
☑ Quick
☐ Counting
☐ Radix

Size: 10000

Step: 50

Blue -> Insertion, darkBlue -> Asymptotic Insertion
Red -> Merge, darkRed -> Asymptotic Merge
Green -> Heap, darkGreen -> Asymptotic Heap
Yellow -> Quick, darkYellow -> Asymptotic Quick
Cyan -> Counting, darkCyan -> Asymptotic Counting
Magenta -> Radix, darkMagenta -> Asymptotic Radix

Here we compare between asymptotic notation of quick sort(n * n) and actual steps taken by quick sort to sort numbers



Sort Type

- ☐ Insertion
- ☐ Merge
- ☐ Heap
- ☐ Quick
- ☑ Counting
- ☐ Radix

Asymptotic Efficiency

- ☐ Insertion
- ☐ Merge
- ☐ Heap
- ☐ Quick
- ☑ Counting
- ☐ Radix

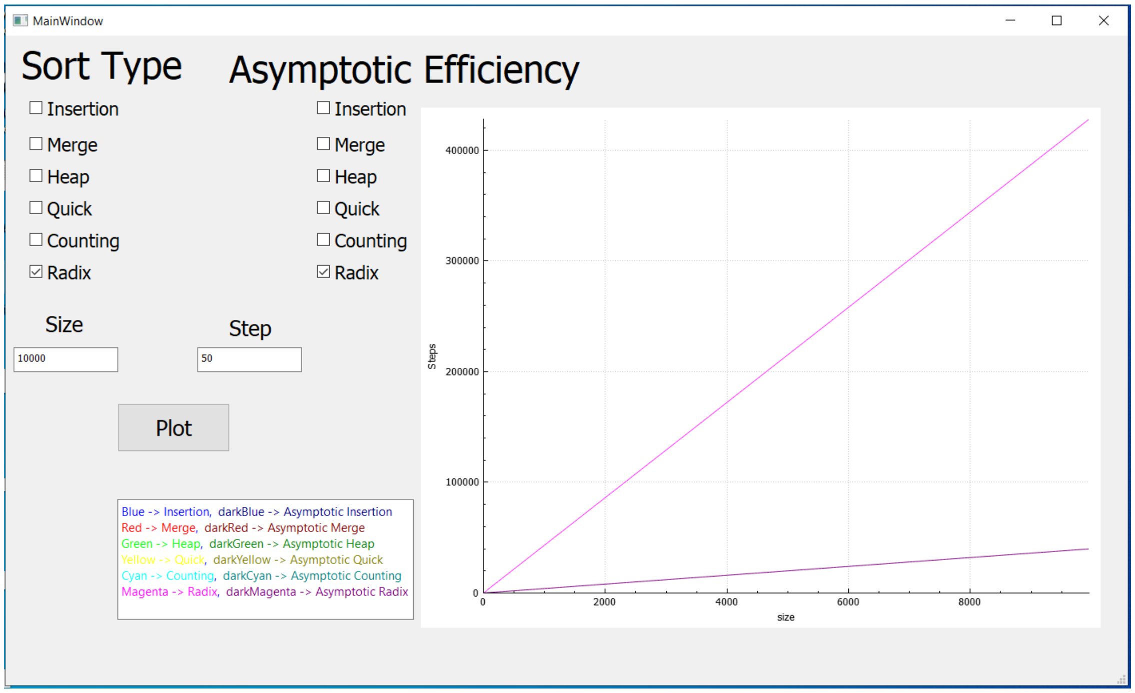
Size: 10000

Step: 50

Plot

Blue -> Insertion, darkBlue -> Asymptotic Insertion
Red -> Merge, darkRed -> Asymptotic Merge
Green -> Heap, darkGreen -> Asymptotic Heap
Yellow -> Quick, darkYellow -> Asymptotic Quick
Cyan -> Counting, darkCyan -> Asymptotic Counting
Magenta -> Radix, darkMagenta -> Asymptotic Radix

Here we compare between asymptotic notation of counting sort(n + k) and actual steps taken by counting sort to sort numbers



Sort Type

- ☐ Insertion
- ☐ Merge
- ☐ Heap
- ☐ Quick
- ☐ Counting
- ☑ Radix

Asymptotic Efficiency

- ☐ Insertion
- ☐ Merge
- ☐ Heap
- ☐ Quick
- ☐ Counting
- ☑ Radix

Size: 10000

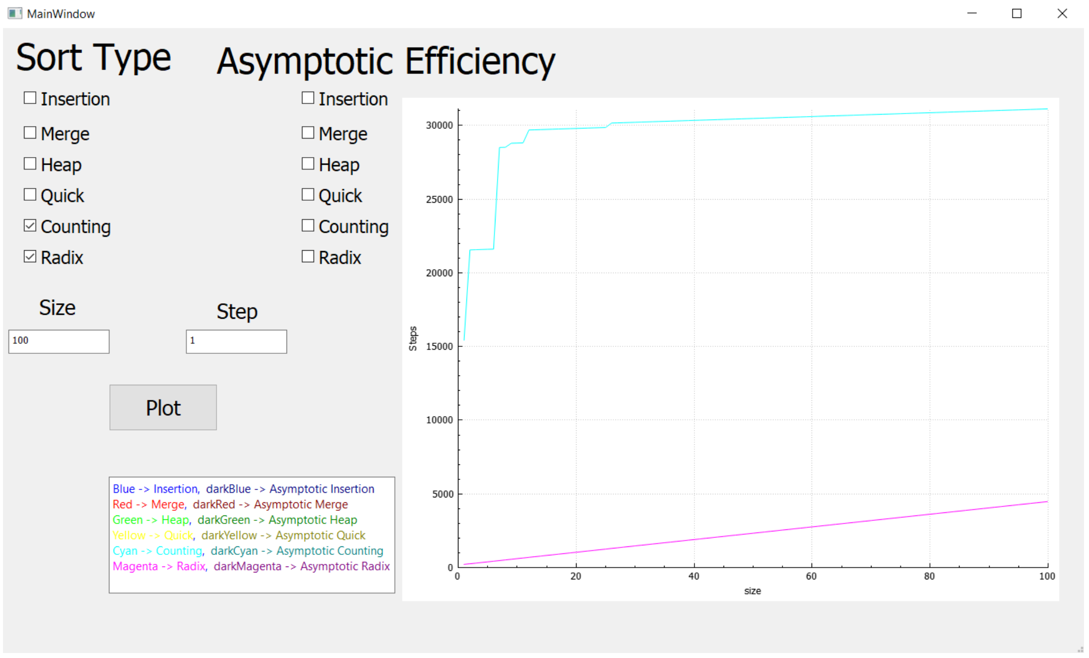
Step: 50

Plot

Blue -> Insertion, darkBlue -> Asymptotic Insertion
Red -> Merge, darkRed -> Asymptotic Merge
Green -> Heap, darkGreen -> Asymptotic Heap
Yellow -> Quick, darkYellow -> Asymptotic Quick
Cyan -> Counting, darkCyan -> Asymptotic Counting
Magenta -> Radix, darkMagenta -> Asymptotic Radix

Here we compare between asymptotic notation of radix sort(d* (n + k)) and actual steps taken by radix sort to sort numbers

Comparison between two sort (steps)



Sort Type

☐ Insertion
☐ Merge
☐ Heap
☐ Quick
☑ Counting
☑ Radix

Asymptotic Efficiency

☐ Insertion
☐ Merge
☐ Heap
☐ Quick
☐ Counting
☐ Radix

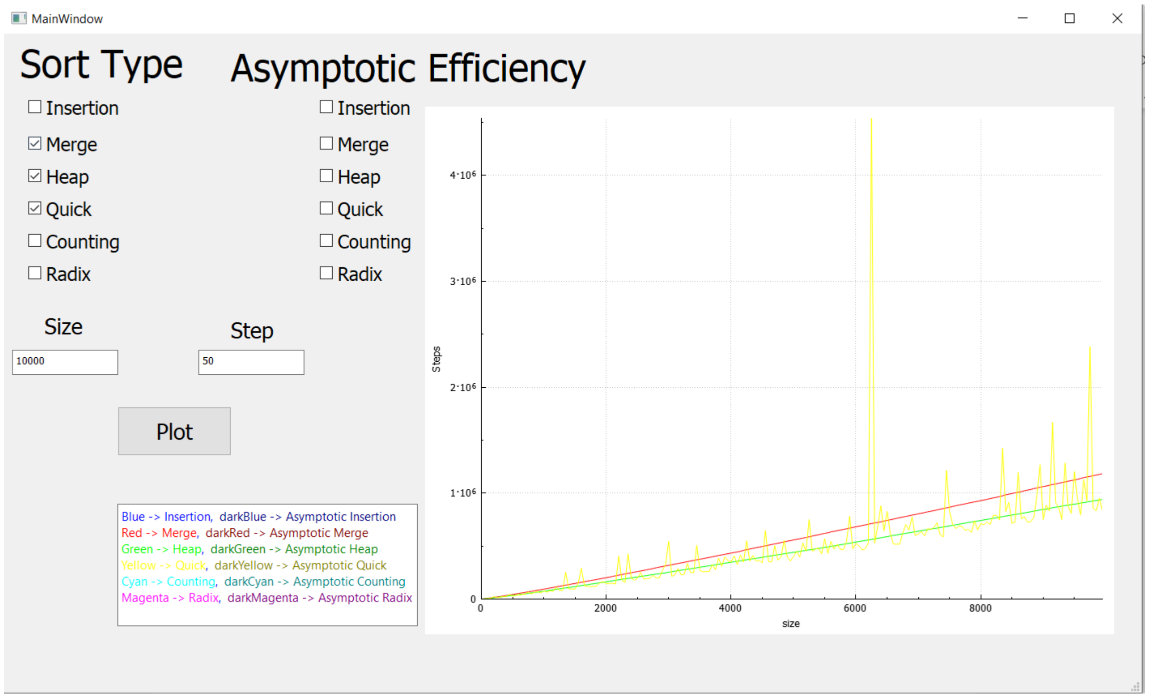
Size: 100

Step: 1

Plot

Blue -> Insertion, darkBlue -> Asymptotic Insertion
Red -> Merge, darkRed -> Asymptotic Merge
Green -> Heap, darkGreen -> Asymptotic Heap
Yellow -> Quick, darkYellow -> Asymptotic Quick
Cyan -> Counting, darkCyan -> Asymptotic Counting
Magenta -> Radix, darkMagenta -> Asymptotic Radix

Comparison between counting and radix (actual steps) which radix takes less steps than counting sort



Sort Type

☐ Insertion
☑ Merge
☑ Heap
☑ Quick
☐ Counting
☐ Radix

Asymptotic Efficiency

☐ Insertion
☐ Merge
☐ Heap
☐ Quick
☐ Counting
☐ Radix

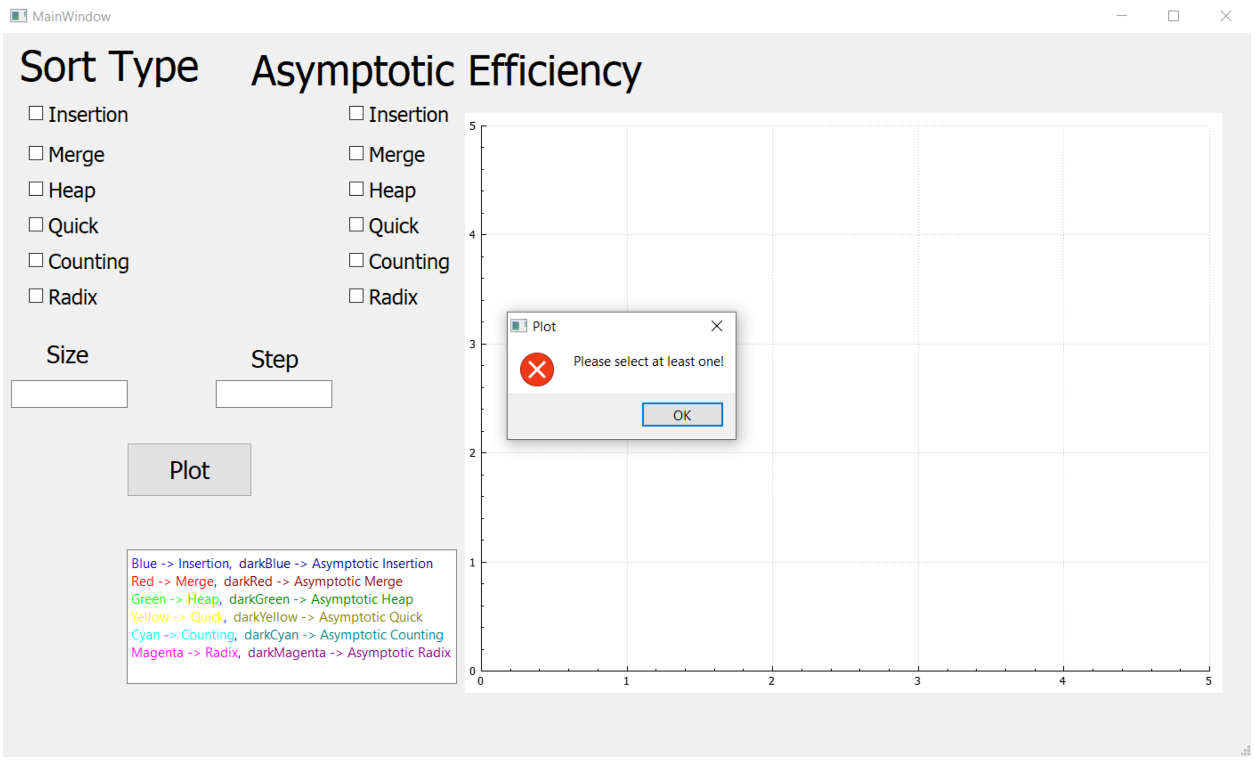
Size: 10000

Step: 50

Plot

Blue -> Insertion, darkBlue -> Asymptotic Insertion
Red -> Merge, darkRed -> Asymptotic Merge
Green -> Heap, darkGreen -> Asymptotic Heap
Yellow -> Quick, darkYellow -> Asymptotic Quick
Cyan -> Counting, darkCyan -> Asymptotic Counting
Magenta -> Radix, darkMagenta -> Asymptotic Radix

Comparison between merge, heap and quick sort (actual steps) which heap takes steps less than merge. Quick sort sometimes less than merge and heap and sometimes much higher than merge and heap, that depending on pivot.



Sort Type: ☐ Insertion ☐ Merge ☐ Heap ☐ Quick ☐ Counting ☐ Radix

Asymptotic Efficiency: ☐ Insertion ☐ Merge ☐ Heap ☐ Quick ☐ Counting ☐ Radix

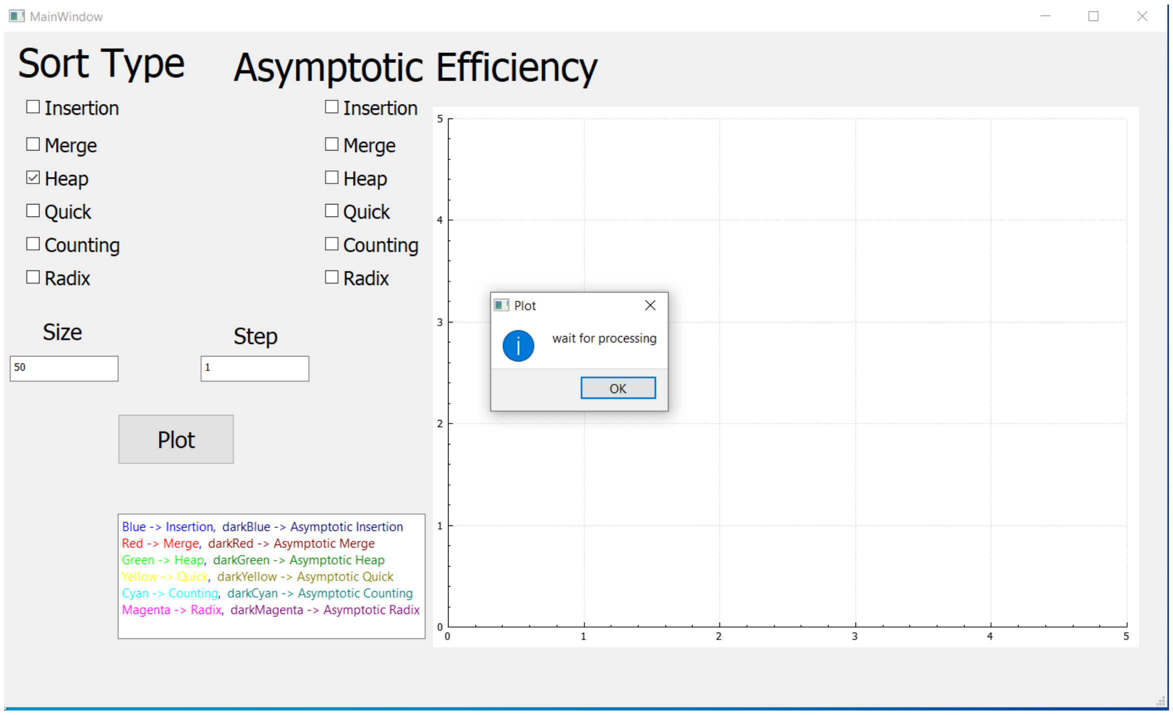
Size:

Step:

Plot: Please select at least one!

Blue -> Insertion, darkBlue -> Asymptotic Insertion
Red -> Merge, darkRed -> Asymptotic Merge
Green -> Heap, darkGreen -> Asymptotic Heap
Yellow -> Quick, darkYellow -> Asymptotic Quick
Cyan -> Counting, darkCyan -> Asymptotic Counting
Magenta -> Radix, darkMagenta -> Asymptotic Radix

Here user does not select any type of sort to compare, so that message appears



Sort Type: ☐ Insertion ☐ Merge ☑ Heap ☐ Quick ☐ Counting ☐ Radix

Asymptotic Efficiency: ☐ Insertion ☐ Merge ☐ Heap ☐ Quick ☐ Counting ☐ Radix

Size: 50

Step: 1

Plot: wait for processing

Blue -> Insertion, darkBlue -> Asymptotic Insertion
Red -> Merge, darkRed -> Asymptotic Merge
Green -> Heap, darkGreen -> Asymptotic Heap
Yellow -> Quick, darkYellow -> Asymptotic Quick
Cyan -> Counting, darkCyan -> Asymptotic Counting
Magenta -> Radix, darkMagenta -> Asymptotic Radix

This message appears after user select type of sort to inform him to wait until plotting because in higher size takes bigger time