Comparison Analysis for Sorting Algorithms

Samyak Ahuja

Overview

Sorting Algorithms chosen for analysis are :

- Insertion Sort
- Merge Sort
- Quick Sort

Helper Functions

Helper functions are used for two purposes:

Data Processing	Plotting
Data Generator	Individual Plotter
Comparison Finder	Combined Plotter

Data Generator and Comparison Finder

Data Generator

About

Objective : To formulate a dataset that is same for all the Sorting functions

Input :

- n which is the maximum number of elements in the set
- sep which is the separator by which the number of elements in the set are increased

Output : Dataset on which Sorting is done. The format of the dataset is explained below.

Dataset Structure

Dataset is a list with the following elements by row.

1	2	…	10
X-1,1	X-1,2	…	X-1,10
X-2,1	X-2,2	…	X-2,10
…	…	…	…
X-n,1	X-n,2	…	X-n,10

X-i,j is an array with a number Xi from 0 to 100

[X1, X2, …, Xi]

```
dataSetGenerator <- function(n = 1000, sep = 10){
    ele <- seq(from = 0, to = n, by = sep)
    ele <- ele[-1]
    data <- list()
    for(j in ele){
      iterator <- j / sep
      repeated <- list()
      for(i in 1:10){
        repeated <- c(repeated, list(sample(x = 1:100, size = j, replace = TRUE)))
      }
      data <- c(data, repeated)
    }
    return (data)
}

dataSet <- dataSetGenerator()
```

Comparison Finder

About

Objective : To output the average number of comparisons used for each row in the dataset given the sorting algorithm

Input :

- func which is the sorting function to use on the dataset
- n which is the maximum number of elements in the set
- sep which is the separator by which the number of elements in the set are increased

Output : a Data-Frame (Matrix) that has two Columns :

- ele which is the number of elements in an array given for Sorting.
- timeElapsed which is the **average** number of comparisons used for that sorting algorithm

```
comp_find <- function(func, n = 1000, sep = 10){
  ele <- seq(from = 0, to = n, by = sep)
  ele <- ele[-1]
  timeElapsed <- c()
  for(j in ele){
    op <- 0
    iterator <- j / sep
    for(i in 1:10){
         op = op + func(dataSet[[iterator + i]])$operations
    }
    #taking average over 10 examples of same size
    op = op / 10
    timeElapsed <- c(timeElapsed, op)
  }
  return (data.frame(ele,timeElapsed))
}
```

Plotting

Individual Plotter

plotter function creates a Comparisons vs Elements plot for each sorting algorithm separately.

The Fitting is done using a polynomial curve of degree 2.

```r
plotter <- function(df, df_title){
  ggplot(df, aes(ele, timeElapsed, color = timeElapsed)) +
    geom_point(shape = 16, size = 5, show.legend = FALSE, alpha = 0.6) +
    stat_smooth(method="lm", formula=y~poly(x,2), rm = FALSE) +
    theme_minimal() +
    labs(subtitle = "Comparisons vs Size",
       y = "Number of Comparisons (Averaged)",
       x = "Number of Elements",
       title = df_title) +
    scale_color_gradient(low = "#32aeff", high = "#f2aeff") +
    stat_poly_eq(parse=T, aes(label = ..eq.label..), formula=y~poly(x,2))
}
```

Combined Plotter

The comb_plotter function creates a combined Comparisons vs Elements plot for all the sorting algorithms.

The Fitting is done using a polynomial curve of degree 2.

```r
comb_plotter <- function(df, df_title){
  ggplot(df, aes(ele, value, col = variable)) +
  geom_point(shape = 16, size = 2, alpha = 0.6) +
  stat_smooth(method="lm", formula=y~poly(x,2)) +
  theme_minimal() +
  labs(subtitle = "Comparisons vs Size",
       y = "Number of Comparisons (Averaged)",
       x = "Number of Elements",
       title = df_title) +
  stat_poly_eq(parse=T, aes(label = ..eq.label..), formula=y~poly(x,2))
}
```

Sorting Function - Implementation

Insertion Sort

Sorting Algorithm

```r
insertionSort <- function(vec){
  n <- length(vec)
  comparisons <- 0
  for(i in 2:n){
    key <- vec[i]
    pos <- i - 1
    while(pos > 0 && vec[pos] > key){
      vec[pos + 1] = vec[pos]
      pos = pos - 1
      comparisons <- comparisons + 1
    }
    vec[pos + 1] <- key
    comparisons <- comparisons + 1
  }
  return (list("vec" = vec, "operations" = comparisons))
}
```

Proof of concept

```
insertionSort(c(12,-22,13,2,-33,2))
```

```
## $vec
## [1] -33 -22   2   2  12  13
##
## $operations
## [1] 14
```

Merge Sort

Sorting Algorithm

```
mergeSort <- function(vec){

  #merges the two sorted halves left and right to get a sorted list
  mergeTwo <- function(left,right){
    comparisons <- 1
    res <- c()
    while(length(left) > 0 && length(right) > 0){
      comparisons <- comparisons + 1
      if(left[1] <= right[1]){
        res <- c(res,left[1])
        left <- left[-1]
      }else{
        res <- c(res,right[1])
        right <- right[-1]
      }
    }
    if(length(left) > 0){
      res <- c(res,left)
    }
    if(length(right) > 0){
      res <- c(res,right)
    }
    return (list("vec" = res, "operations" = comparisons))
  }
  comparisons <- 0
  n <- length(vec)

  if(n <= 1) return (list("vec" = vec, "operations" = comparisons))
  else{
    middle <- length(vec) %/% 2 #integer division

    left_list <- mergeSort(vec[1:middle])
    right_list <- mergeSort(vec[(middle + 1):n])
    left <- left_list$vec
    right <- right_list$vec

    res <- mergeTwo(left,right)
    comparisons <- left_list$operations + right_list$operations + res$operations
    return (list("vec" = res$vec, "operations" = comparisons))
  }
}
```

Proof of Concept

```
mergeSort(c(12,-22,13,2,-33,2))
```

```
## $vec
## [1] -33 -22   2   2  12  13
##
## $operations
## [1] 15
```

Quick Sort

Sorting Algorithm

```
quickSort <- function(vec, low = 1, high = length(vec)){

  #partitioning vector around pivot(chosen as last element of the vector)
  partition <- function(vec, low, high){
    i = low
    comparisons <- 0
    pivot = vec[high]

    for(j in low:(high - 1)){
      comparisons <- comparisons + 1
      if(vec[j] <= pivot){
        #swap v[i] and v[j]
        temp = vec[i]
        vec[i] = vec[j]
        vec[j] = temp
        i = i + 1
      }
    }
    #swap pivot with v[i]
    temp = vec[i]
    vec[i] = vec[high]
    vec[high] = temp

    return (list("vec" = vec, "operations" = comparisons, "pi" = i))
  }

  comparisons <- 0
  if(low < high){
    pi_list = partition(vec, low, high)
    vec <- pi_list$vec
    pi <- pi_list$pi

    left_list <- quickSort(vec, low, pi - 1)
    vec <- left_list$vec

    right_list <- quickSort(vec, pi + 1, high)
    vec <- right_list$vec

    comparisons <- left_list$operations + right_list$operations + pi_list$operations
    return (list("vec" = vec, "operations" = comparisons))
  }else{
    return (list("vec" = vec, "operations" = comparisons))
  }
}
```

Proof of Concept

```
quickSort(c(12,-22,13,2,-33,2))
```

```
## $vec
## [1] -33 -22   2   2  12  13
##
## $operations
## [1] 9
```

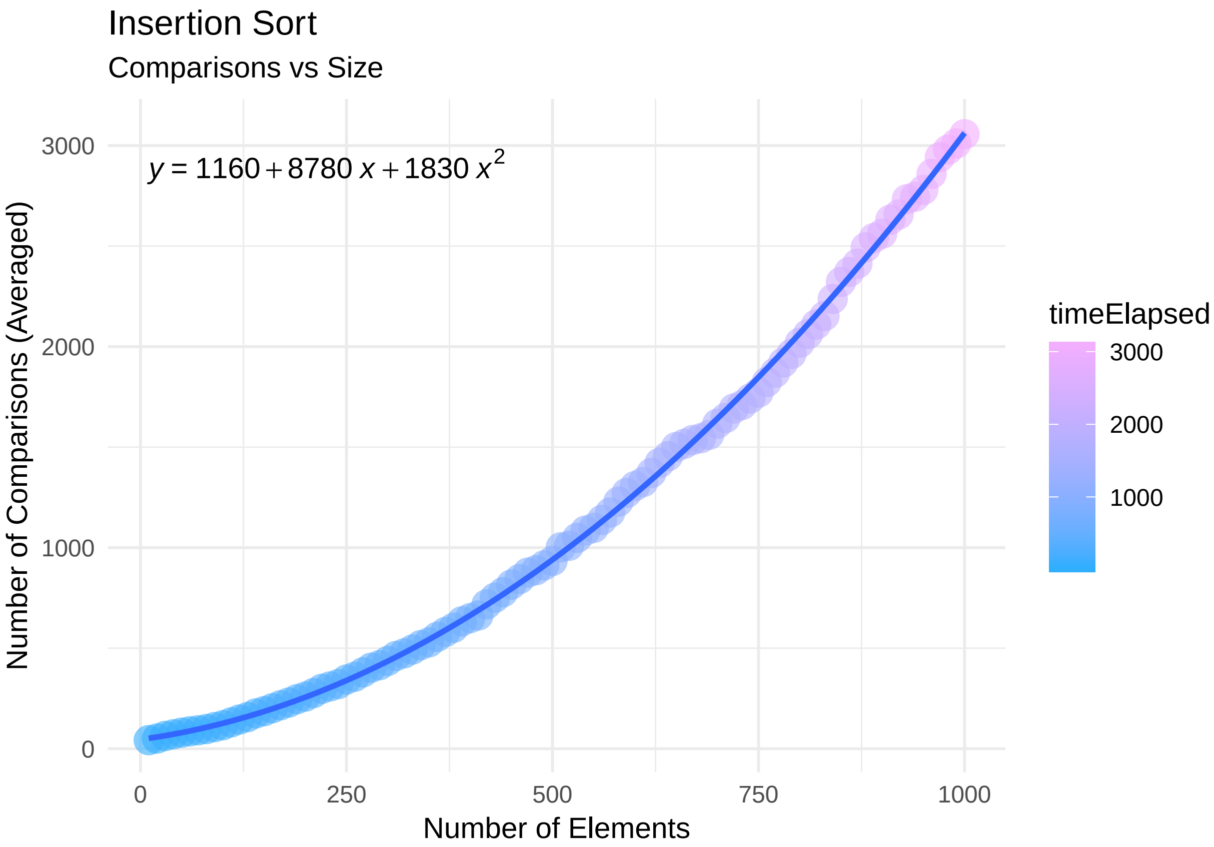
Sorting Algorithms - Plots

Individual Plots

Insertion Sort

```
isdf_small <- comp_find(insertionSort)
plotter(isdf_small, "Insertion Sort")
```

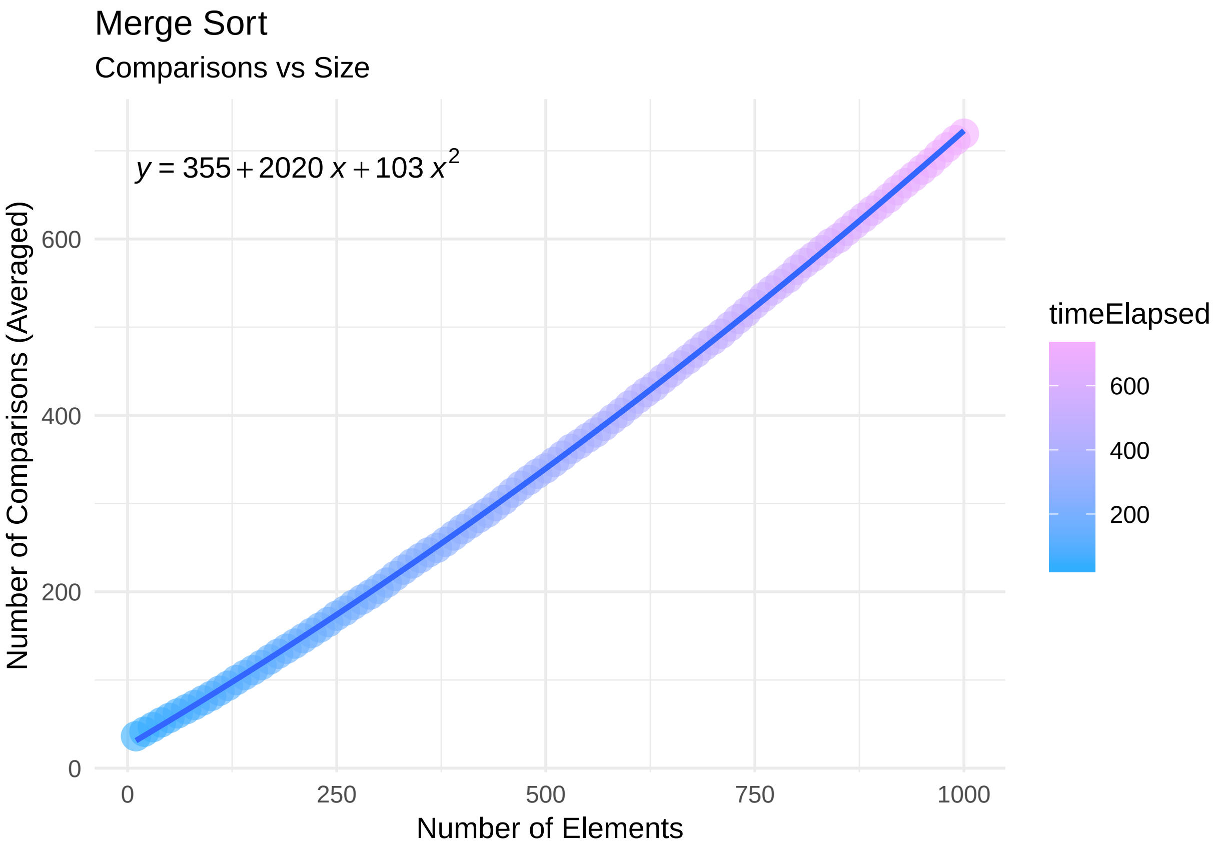
```
## Warning: Ignoring unknown parameters: rm
```



Merge Sort

```
msdf_small <- comp_find(mergeSort)
plotter(msdf_small, "Merge Sort")
```

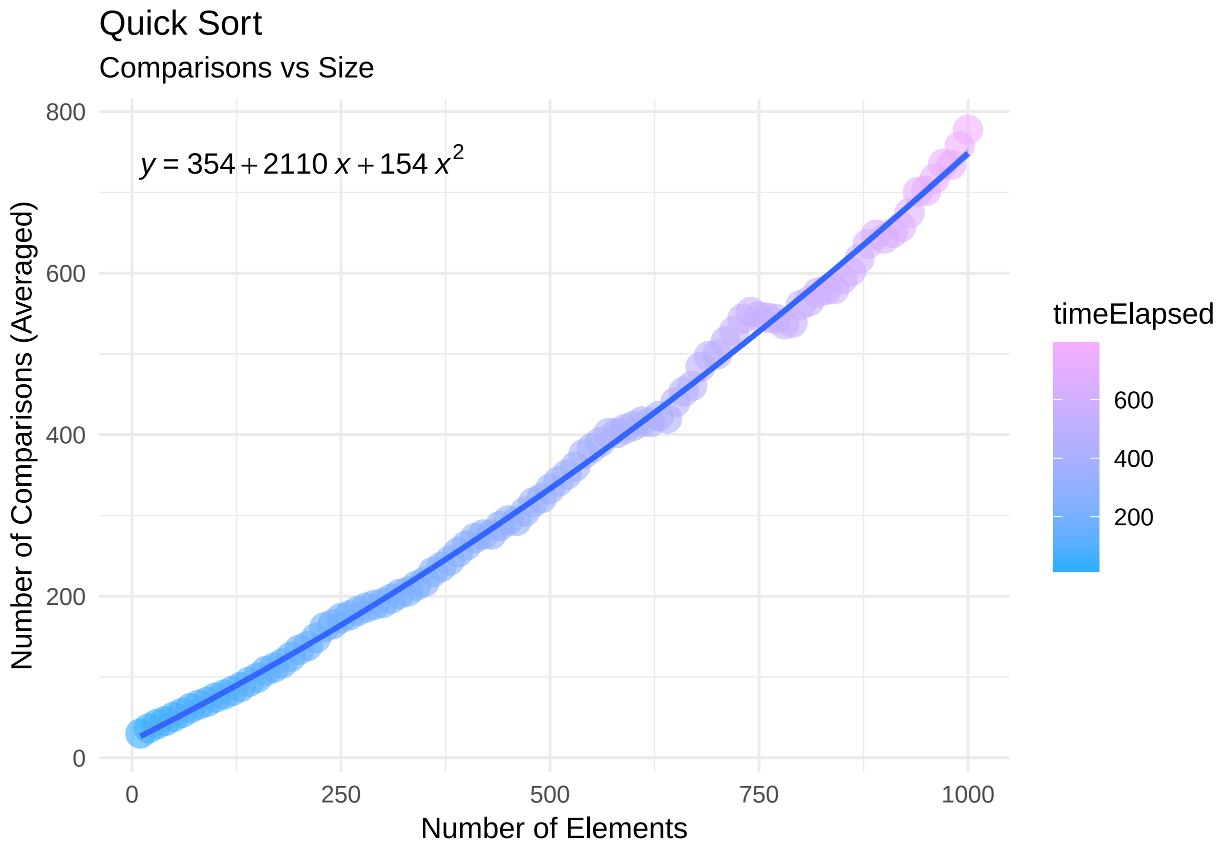
```
## Warning: Ignoring unknown parameters: rm
```



Quick Sort

```
qsdf_small <- comp_find(quickSort)
plotter(qsdf_small, "Quick Sort")
```

```
## Warning: Ignoring unknown parameters: rm
```

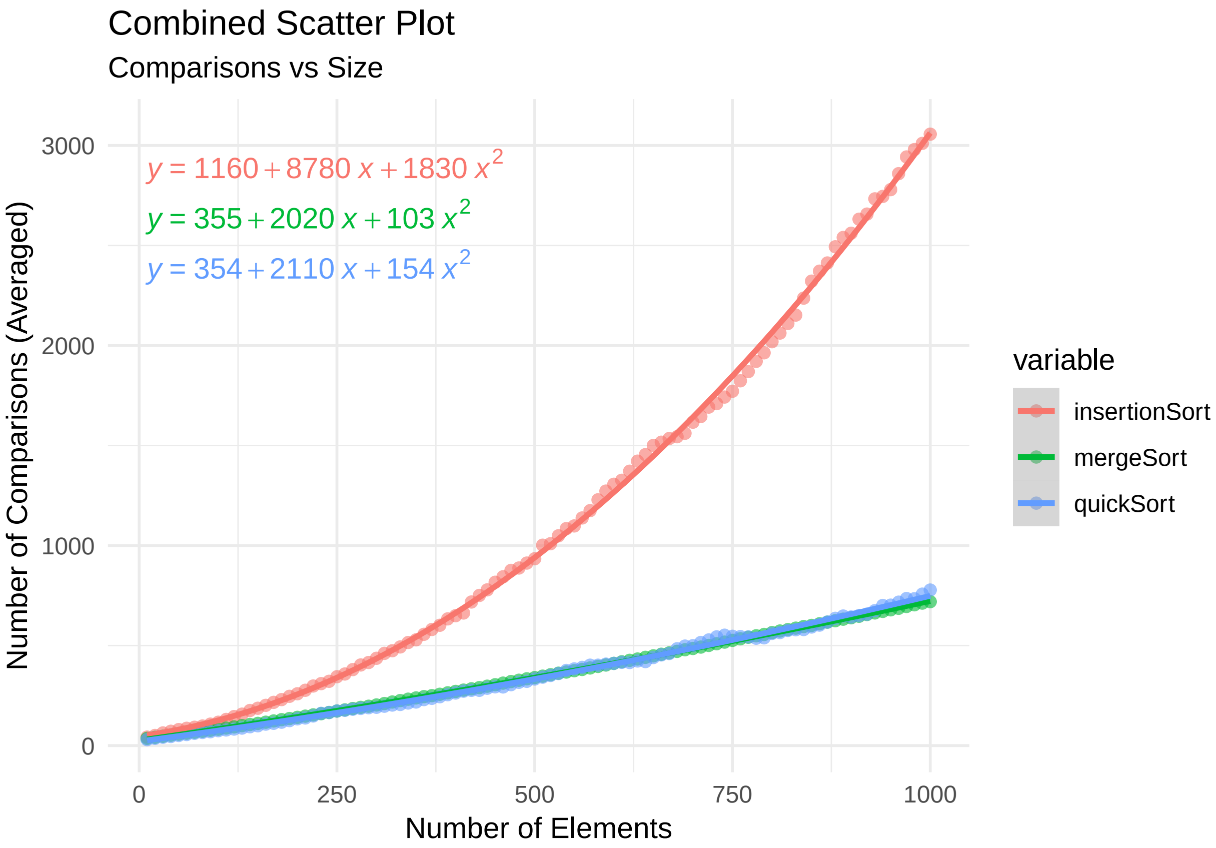


Combined Plots

```
df_small <- data.frame(ele = msdf_small[[1]],
                 insertionSort = isdf_small[[2]],
                 mergeSort = msdf_small[[2]],
                 quickSort = qsdf_small[[2]])
df_small
```

```
##      ele insertionSort mergeSort quickSort
## 1     10          42.6      36.2      30.0
## 2     20          50.7      41.3      36.4
## 3     30          63.4      46.5      41.7
## 4     40          71.7      51.9      45.5
## 5     50          80.1      56.9      50.7
## 6     60          86.6      62.1      55.5
## 7     70          91.7      66.8      61.4
## 8     80          97.9      71.6      65.7
## 9     90         107.4      76.8      69.3
## 10   100         116.9      81.9      74.5
## 11   110         130.9      87.8      78.2
## 12   120         145.1      93.4      82.7
## 13   130         157.3      99.9      87.6
## 14   140         175.5     105.6      94.1
## 15   150         186.7     111.1      99.1
## 16   160         201.3     117.0     107.4
## 17   170         216.2     123.3     111.5
## 18   180         229.9     129.7     117.0
## 19   190         246.4     135.4     125.0
## 20   200         259.1     141.2     133.8
## 21   210         276.6     147.3     138.2
## 22   220         297.9     153.5     148.2
## 23   230         309.8     159.5     161.3
## 24   240         321.4     165.7     165.3
## 25   250         344.3     172.9     172.8
## 26   260         358.0     178.5     176.4
## 27   270         379.9     185.3     181.7
## 28   280         403.1     191.2     185.9
## 29   290         415.4     196.9     189.3
## 30   300         436.5     203.2     191.6
## 31   310         461.1     210.4     196.9
## 32   320         474.3     217.9     202.9
## 33   330         493.2     225.1     205.8
## 34   340         515.3     232.1     213.0
## 35   350         528.8     238.3     217.9
## 36   360         556.4     244.7     230.1
## 37   370         580.5     249.8     236.3
## 38   380         601.5     256.7     244.2
## 39   390         633.5     263.8     254.5
## 40   400         649.4     270.9     262.8
## 41   410         662.9     277.4     272.0
## 42   420         717.9     283.8     276.4
## 43   430         750.7     289.8     276.4
## 44   440         778.5     297.1     286.9
## 45   450         816.6     304.2     293.8
## 46   460         843.9     312.4     293.3
## 47   470         875.8     320.2     304.6
## 48   480         887.8     326.8     316.3
## 49   490         912.5     333.9     321.3
## 50   500         934.1     339.9     332.9
## 51   510        1001.8     347.2     342.1
## 52   520        1009.3     354.3     350.6
## 53   530        1048.8     362.0     360.3
## 54   540        1084.9     367.7     375.8
## 55   550        1098.2     374.5     383.4
## 56   560        1138.0     380.7     390.9
## 57   570        1174.6     388.1     402.0
## 58   580        1228.8     395.9     402.0
## 59   590        1272.2     402.9     407.3
## 60   600        1306.4     411.1     411.1
## 61   610        1326.3     418.9     416.4
## 62   620        1371.6     426.3     415.5
## 63   630        1421.9     432.9     423.1
## 64   640        1454.6     441.3     420.0
## 65   650        1500.4     448.7     440.3
## 66   660        1516.6     456.7     453.4
## 67   670        1535.1     463.5     460.4
## 68   680        1544.2     471.1     484.1
## 69   690        1561.3     479.2     497.5
## 70   700        1616.6     485.5     499.5
## 71   710         1645.0     492.6     515.6
## 72   720         1692.1     500.8     528.2
## 73   730         1709.4     509.3     543.5
## 74   740         1742.3     517.3     552.0
## 75   750         1771.5     526.2     546.5
## 76   760         1823.7     534.0     545.0
## 77   770         1869.7     541.6     543.6
## 78   780         1920.8     549.0     536.7
## 79   790         1963.6     555.7     538.8
## 80   800         2019.1     565.0     560.8
## 81   810         2062.1     572.9     565.1
## 82   820         2110.0     579.8     575.8
## 83   830         2152.1     587.2     579.1
## 84   840         2236.9     595.1     580.2
## 85   850         2321.7     600.9     592.9
## 86   860         2371.6     609.1     602.3
## 87   870         2412.6     617.1     617.7
## 88   880         2493.6     624.3     636.2
## 89   890         2540.2     632.0     647.7
## 90   900         2561.2     639.1     642.9
## 91   910         2631.4     646.5     649.8
## 92   920         2657.0     655.3     657.2
## 93   930         2733.1     663.3     674.6
## 94   940         2745.4     670.9     700.6
## 95   950         2779.3     678.7     702.1
## 96   960         2859.1     686.4     717.1
## 97   970         2942.6     695.5     735.2
## 98   980         2979.4     704.0     734.2
## 99   990         3010.5     712.2     757.2
## 100  1000        3056.4     719.5     777.7
```

```
df_small <- melt(df_small, id.vars = "ele")
comb_plotter(df_small, "Combined Scatter Plot")
```


Combined Scatter Plot
Comparisons vs Size
$y = 1160 + 8780\,x + 1830\,x^2$
$y = 355 + 2020\,x + 103\,x^2$
$y = 354 + 2110\,x + 154\,x^2$
Number of Comparisons (Averaged)
3000
2000
1000
0
0
250
500
750
1000
Number of Elements
variable
insertionSort
mergeSort
quickSort