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.

```
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.

```
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

```
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){

  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))
  }
```

```r
  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

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

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

Quick Sort

Sorting Algorithm

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

  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){
        temp = vec[i]
        vec[i] = vec[j]
        vec[j] = temp
        i = i + 1
      }
    }
    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
```

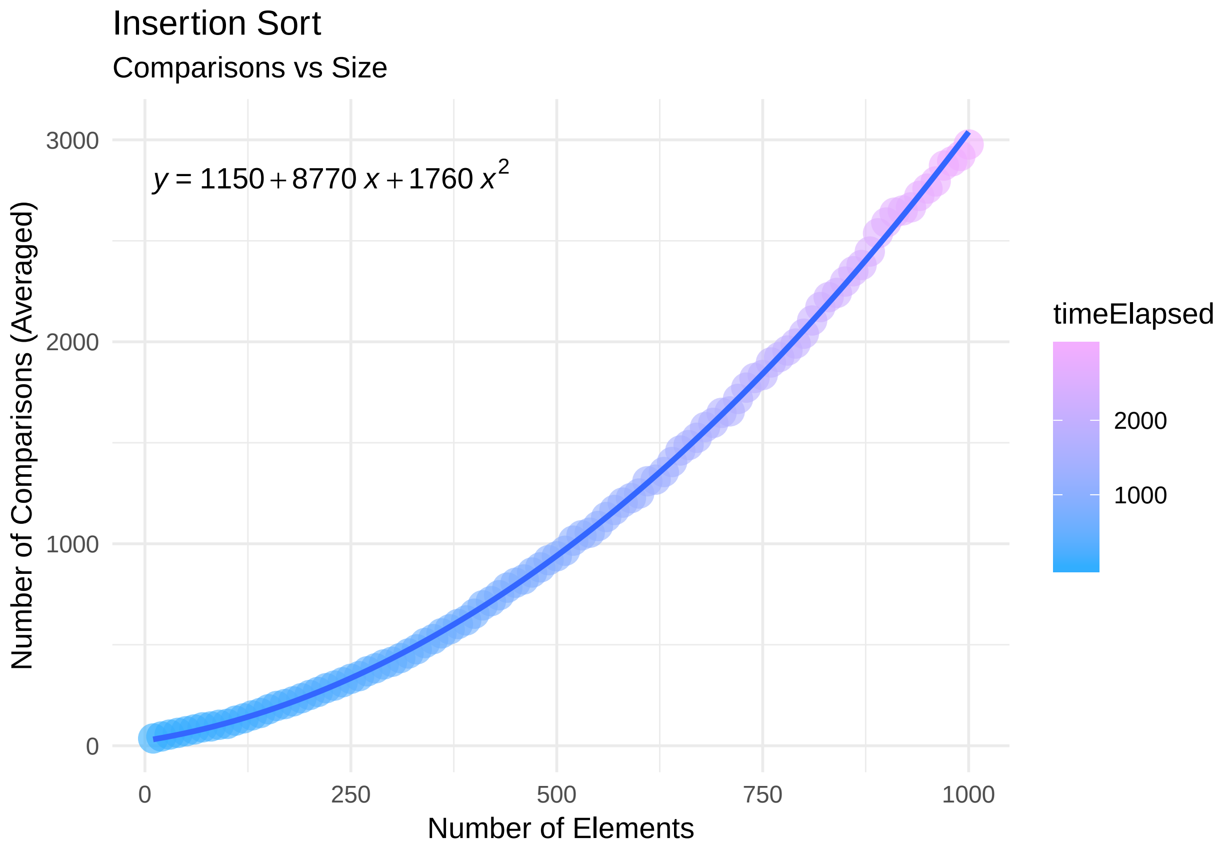
Sorting Algorithms - Plots

Individual Plots

Insertion Sort

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

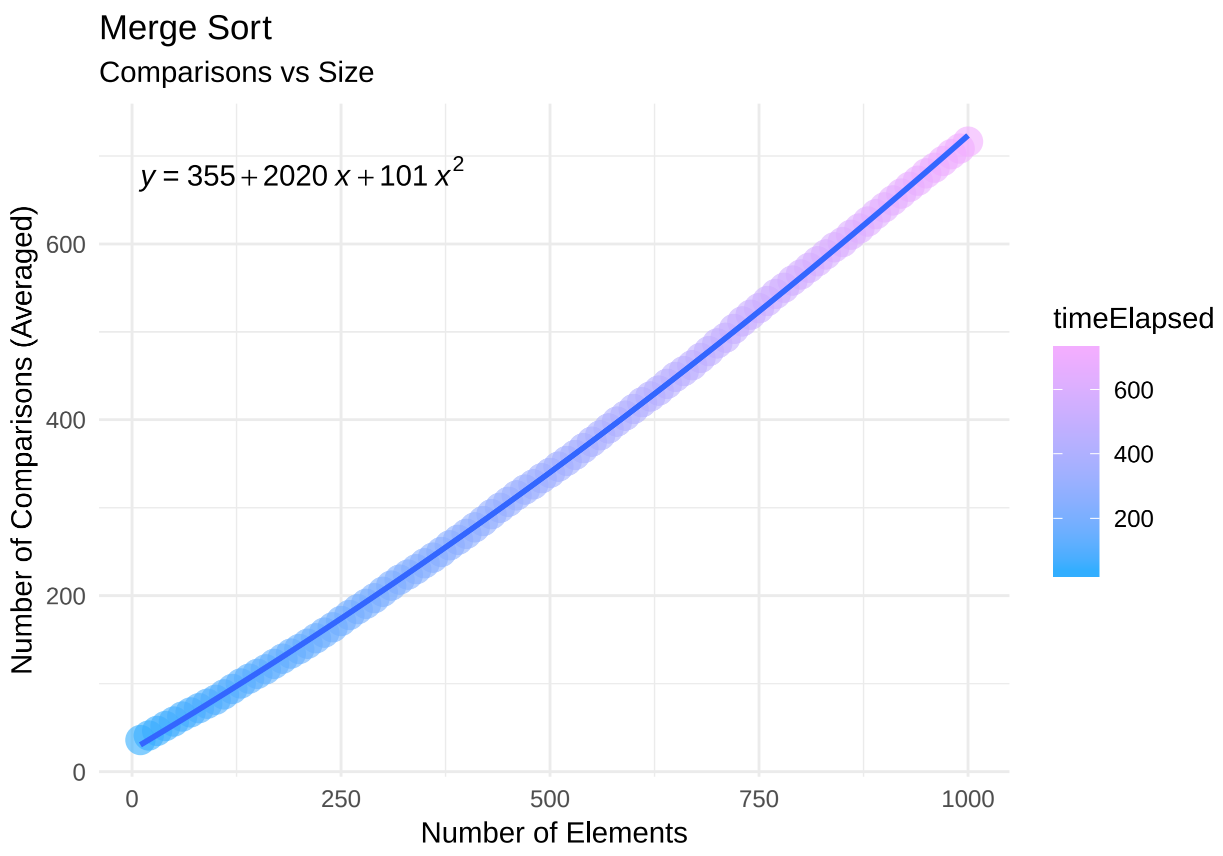
```
## Warning: Ignoring unknown parameters: rm
```



Merge Sort

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

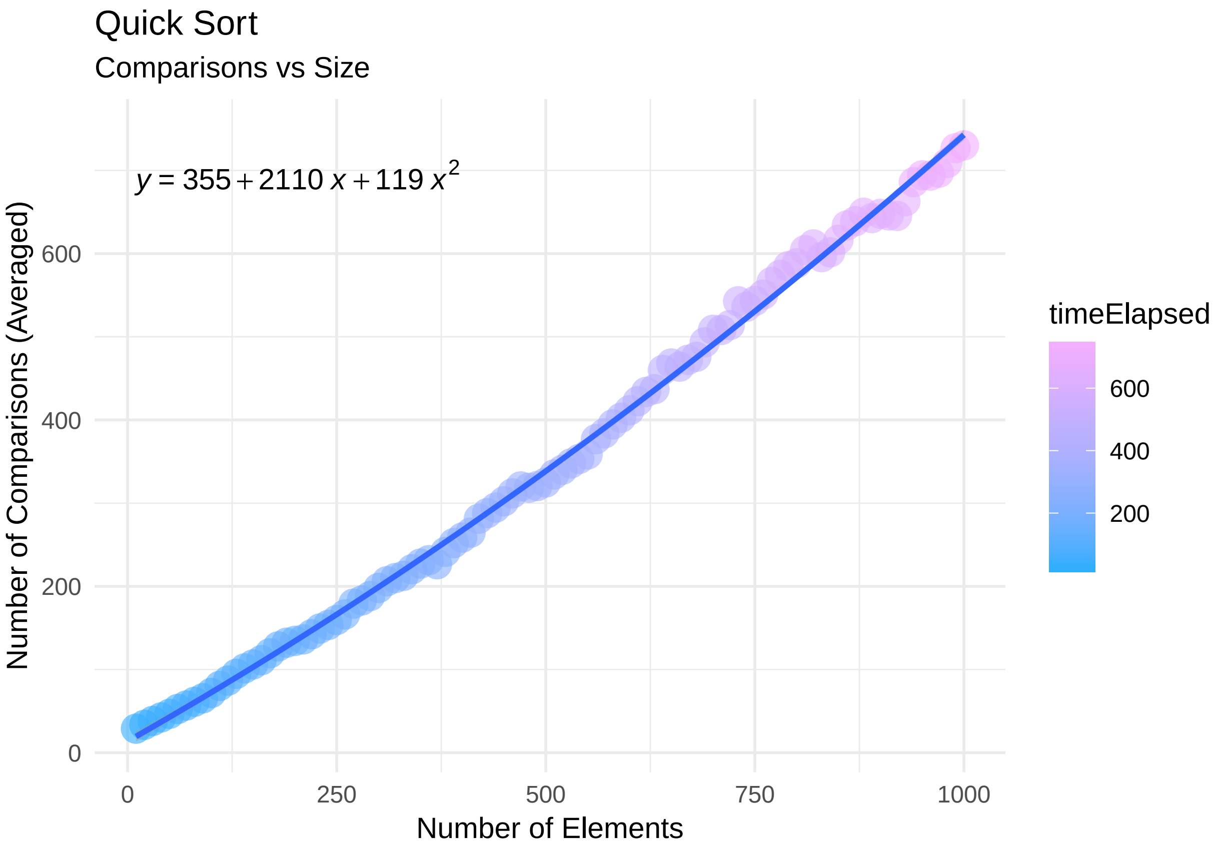
```
## Warning: Ignoring unknown parameters: rm
```



Quick Sort

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

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

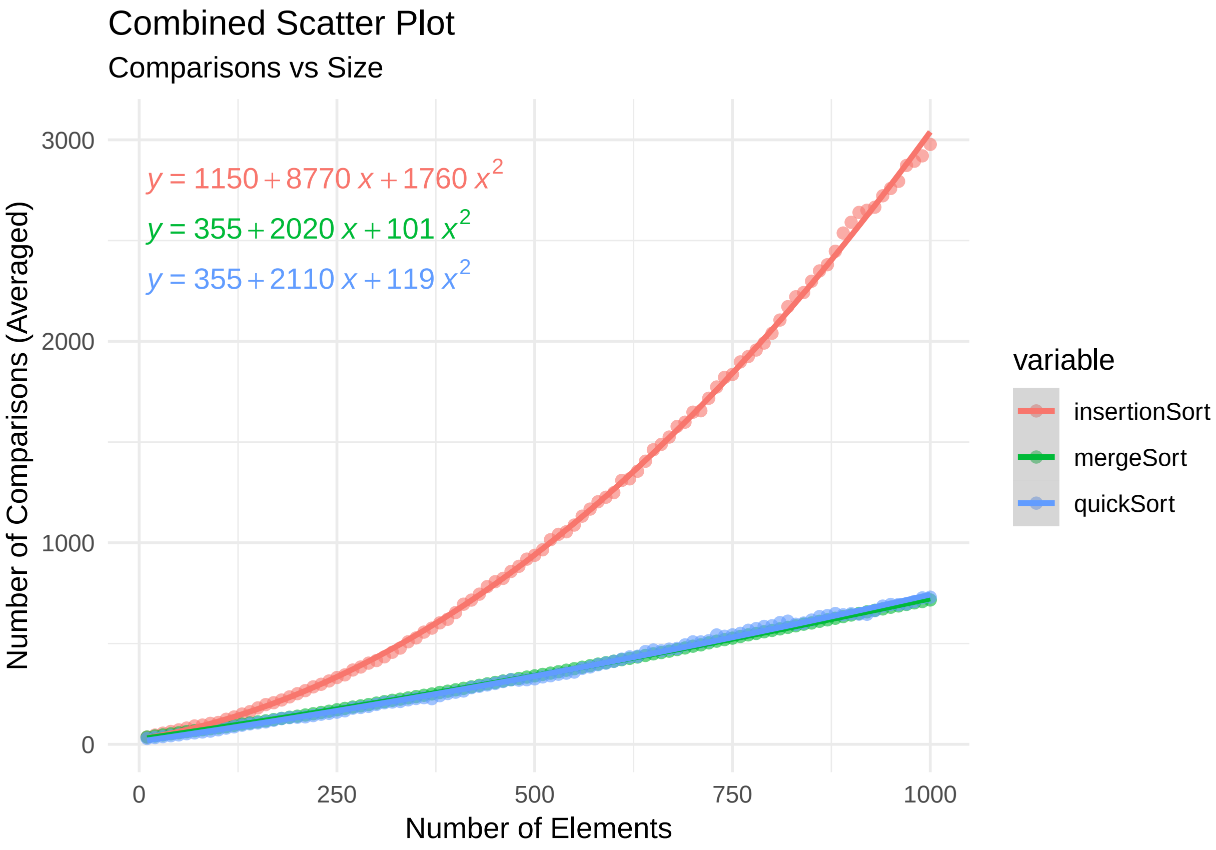


Combined Plots

```
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          36.0      35.9      28.9
## 2     20          45.8      41.0      33.5
## 3     30          55.2      46.3      38.2
## 4     40          63.7      51.8      42.4
## 5     50          71.7      56.9      46.7
## 6     60          80.7      62.6      52.6
## 7     70          91.0      67.3      56.7
## 8     80          95.5      72.1      61.2
## 9     90         103.8      77.2      65.4
## 10   100         108.3      81.7      71.8
## 11   110         124.2      87.8      80.3
## 12   120         135.6      94.0      86.7
## 13   130         150.2     100.2      94.8
## 14   140         162.1     105.7     101.4
## 15   150         180.6     111.2     106.1
## 16   160         197.0     116.3     111.2
## 17   170         206.3     122.5     119.5
## 18   180         219.7     128.4     127.8
## 19   190         235.3     134.2     132.2
## 20   200         251.2     139.4     134.4
## 21   210         266.3     145.4     136.2
## 22   220         285.0     151.6     142.4
## 23   230         297.8     158.0     149.3
## 24   240         314.3     164.1     153.6
## 25   250         331.3     171.3     159.5
## 26   260         344.5     178.1     166.2
## 27   270         368.6     184.7     179.2
## 28   280         383.5     190.8     183.0
## 29   290         402.8     197.2     188.3
## 30   300         415.6     204.5     198.0
## 31   310         434.0     211.5     206.1
## 32   320         456.8     218.3     209.9
## 33   330         477.5     224.0     212.6
## 34   340         508.6     230.2     220.5
## 35   350         527.9     236.9     227.3
## 36   360         556.7     243.3     231.4
## 37   370         576.5     249.8     226.6
## 38   380         602.6     257.3     241.4
## 39   390         620.5     263.6     252.0
## 40   400         653.7     270.4     258.6
## 41   410         695.2     277.7     264.5
## 42   420         715.7     284.9     281.3
## 43   430         745.2     292.7     288.0
## 44   440         782.8     300.0     294.6
## 45   450         807.0     306.8     302.2
## 46   460         823.5     314.1     311.7
## 47   470         857.7     320.7     320.1
## 48   480         883.2     326.6     318.5
## 49   490         918.9     333.6     320.6
## 50   500         938.0     339.7     324.8
## 51   510         965.3     346.8     334.5
## 52   520        1015.4     353.3     340.2
## 53   530        1042.0     360.1     347.8
## 54   540        1054.7     367.8     353.2
## 55   550        1088.0     375.2     358.5
## 56   560        1132.2     382.4     376.9
## 57   570        1167.3     390.2     383.9
## 58   580        1203.7     398.0     394.7
## 59   590        1226.0     404.8     402.7
## 60   600        1248.8     412.4     411.6
## 61   610        1309.6     420.0     422.6
## 62   620        1317.5     426.6     433.7
## 63   630        1355.3     433.3     437.0
## 64   640        1404.8     441.0     460.1
## 65   650        1461.7     448.9     467.9
## 66   660        1488.5     455.4     464.1
## 67   670        1524.8     462.1     472.3
## 68   680        1577.7     470.4     475.9
## 69   690        1598.6     478.1     493.3
## 70   700        1648.2     486.9     508.4
## 71   710        1655.5     493.4     508.2
## 72   720        1717.3     503.4     514.0
## 73   730        1773.2     511.9     542.7
## 74   740        1821.0     519.6     536.1
## 75   750        1835.7     527.0     543.1
## 76   760        1897.8     535.1     550.7
## 77   770        1924.4     543.2     565.9
## 78   780        1956.8     550.2     574.4
## 79   790        1991.3     558.4     584.9
## 80   800        2040.0     565.3     588.3
## 81   810        2105.5     572.9     604.2
## 82   820        2171.9     580.3     611.1
## 83   830        2221.0     588.1     595.7
## 84   840        2242.1     596.2     601.6
## 85   850        2298.2     602.1     616.6
## 86   860        2349.3     610.5     633.9
## 87   870        2380.1     617.6     638.8
## 88   880        2446.9     625.5     649.1
## 89   890        2537.3     633.9     642.7
## 90   900        2590.7     641.6     647.8
## 91   910        2639.7     649.5     645.9
## 92   920        2650.5     657.3     645.3
## 93   930        2665.7     665.0     663.0
## 94   940        2722.2     671.9     685.9
## 95   950        2758.6     680.3     694.1
## 96   960        2794.0     686.7     693.8
## 97   970        2872.7     694.3     697.2
## 98   980        2893.7     702.1     708.9
## 99   990        2920.4     708.5     726.7
## 100 1000        2977.2     716.4     730.1
```

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


Combined Scatter Plot
Comparisons vs Size
$y = 1150 + 8770\,x + 1760\,x^2$
$y = 355 + 2020\,x + 101\,x^2$
$y = 355 + 2110\,x + 119\,x^2$
Number of Comparisons (Averaged)
0
1000
2000
3000
0
250
500
750
1000
Number of Elements
variable
insertionSort
mergeSort
quickSort