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]

```r
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

```r
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){

  #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
```

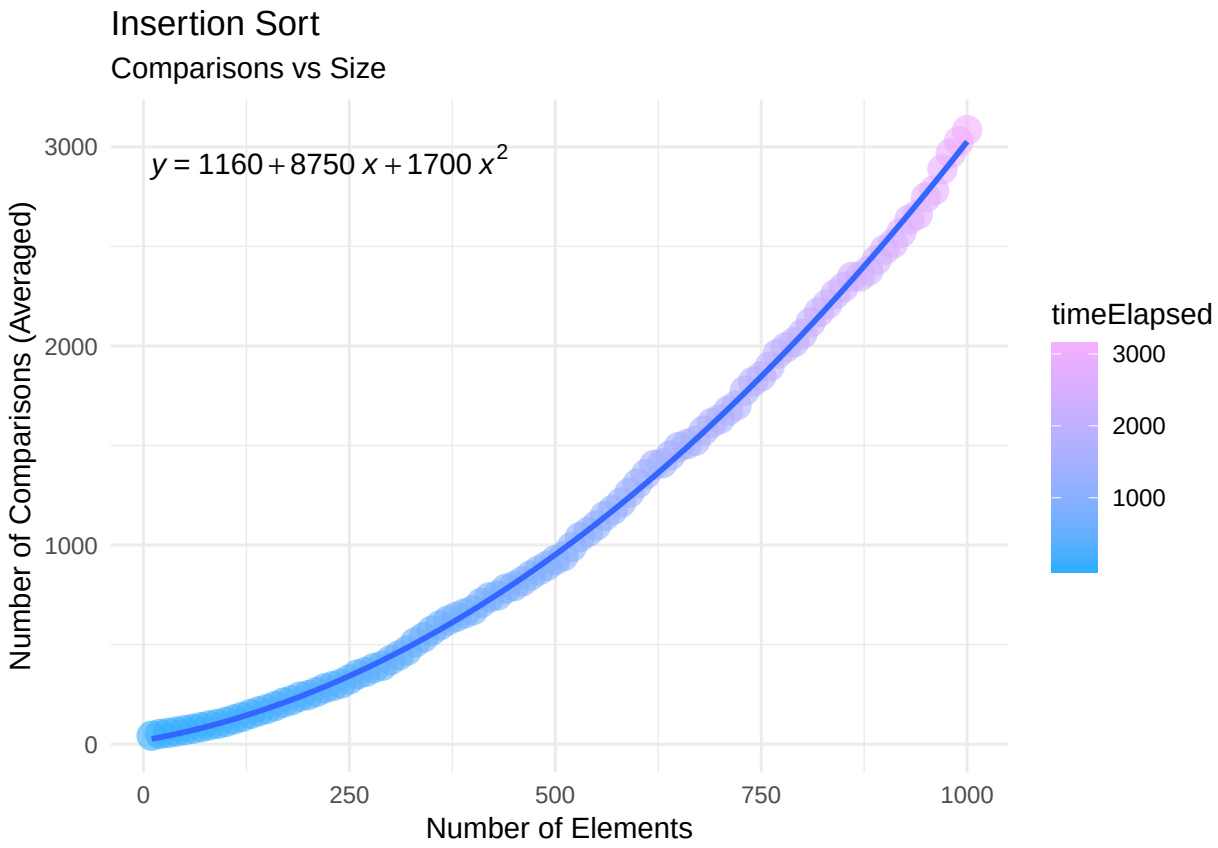
Sorting Algorithms - Plots

Individual Plots

Insertion Sort

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

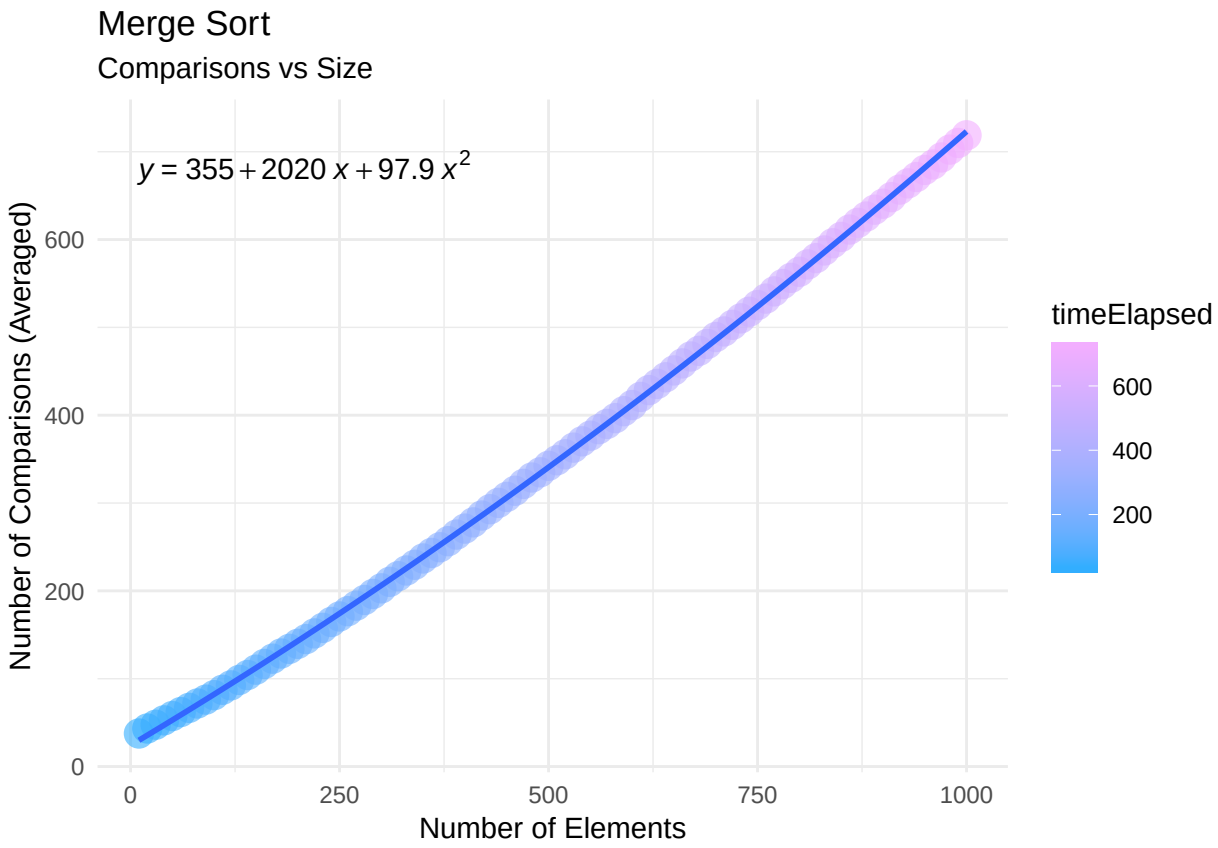
```
## Warning: Ignoring unknown parameters: rm
```



Merge Sort

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

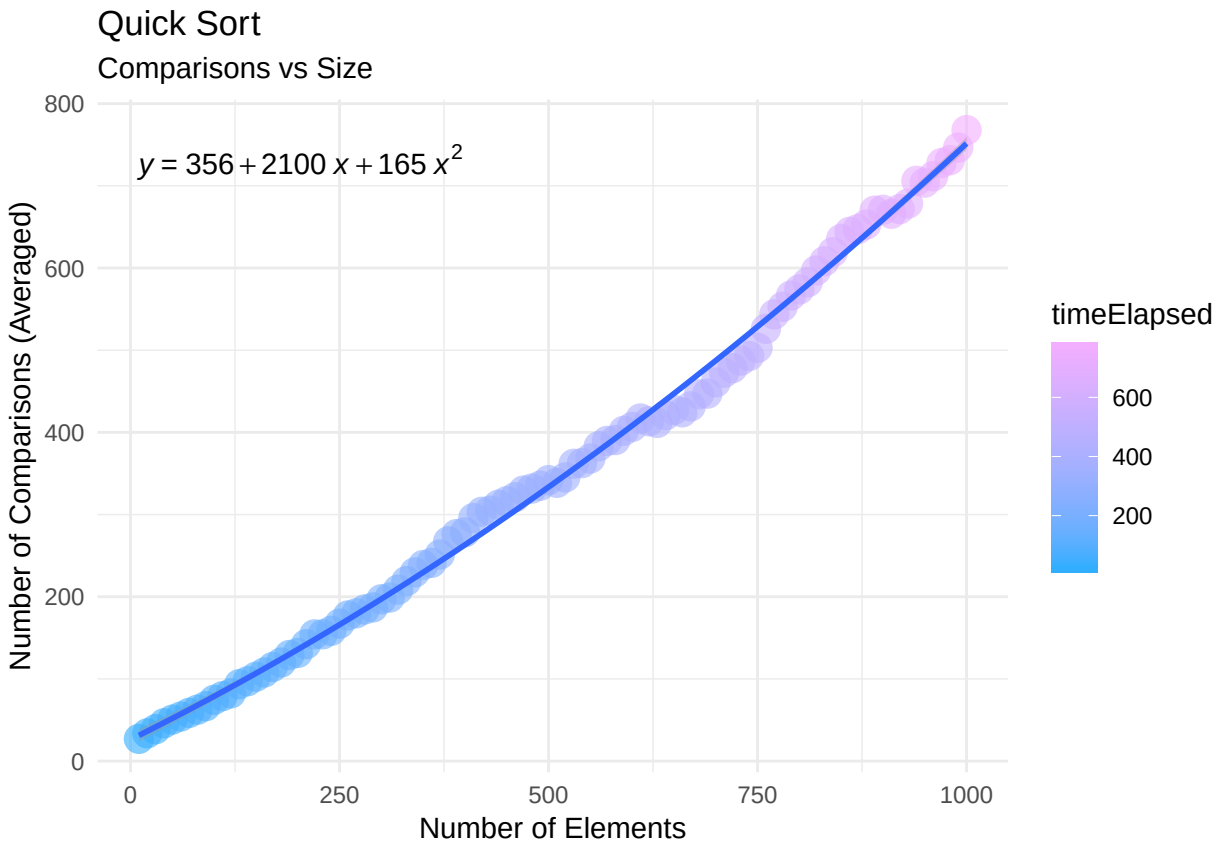
```
## Warning: Ignoring unknown parameters: rm
```



Quick Sort

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

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



Combined Plots

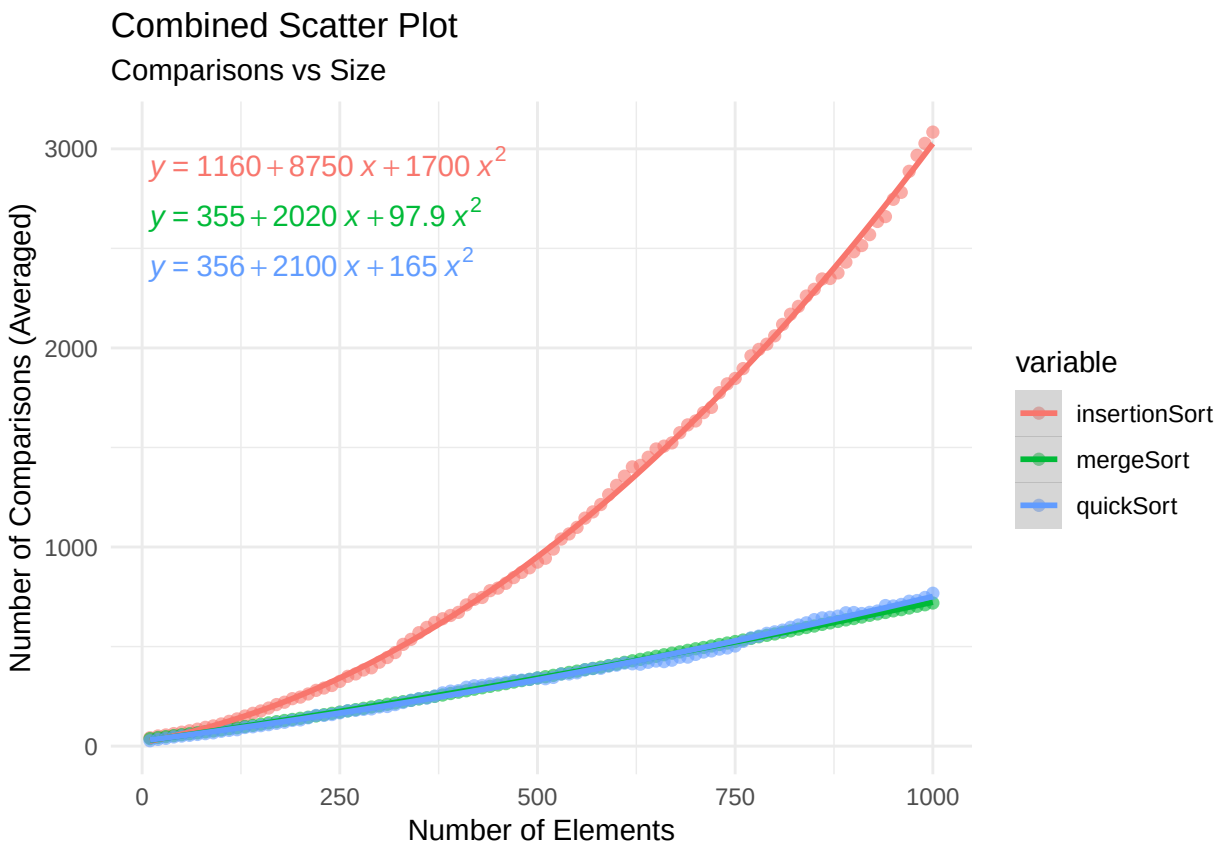
```
df_small <- data.frame(ele = msdf_small[[1]],
                       insertionSort = isdf_small[[2]],
                       mergeSort = msdf_small[[2]],
                       quickSort = qsdf_small[[2]])
df_small
```

```
##       ele insertionSort mergeSort quickSort
## 1      10          42.3      37.6      27.1
## 2      20          51.1      43.3      33.8
## 3      30          56.2      47.6      38.8
## 4      40          63.0      52.6      46.0
## 5      50          70.0      57.5      50.5
## 6      60          77.2      62.0      54.2
## 7      70          85.1      66.8      58.7
## 8      80          94.3      71.8      62.6
## 9      90         101.6      76.2      66.8
## 10    100         111.3      81.3      74.6
## 11    110         124.4      87.4      79.0
## 12    120         136.6      92.4      82.4
## 13    130         151.3      98.6      93.6
## 14    140         164.8     104.2      97.2
## 15    150         176.7     110.3     102.7
## 16    160         191.2     116.8     107.8
```

```
## 17   170         208.7     122.8     114.5
## 18   180         220.5     128.7     119.9
## 19   190         238.4     134.0     129.1
## 20   200         246.2     139.7     131.6
## 21   210         261.9     145.1     142.2
## 22   220         280.1     151.6     154.2
## 23   230         291.8     157.9     154.3
## 24   240         304.5     164.6     158.5
## 25   250         325.0     170.9     167.3
## 26   260         350.4     176.9     177.2
## 27   270         362.7     183.2     179.9
## 28   280         382.4     189.7     184.5
## 29   290         393.4     196.5     187.0
## 30   300         421.5     202.9     196.9
## 31   310         445.1     210.3     199.0
## 32   320         470.0     216.7     208.6
## 33   330         511.0     223.2     219.1
## 34   340         536.2     229.7     229.6
## 35   350         569.7     236.9     238.6
## 36   360         596.4     243.2     241.2
## 37   370         621.4     249.7     251.4
## 38   380         639.8     256.6     267.4
## 39   390         656.2     264.1     276.2
## 40   400         671.8     270.9     278.4
## 41   410         709.2     278.0     295.7
## 42   420         737.1     285.8     303.3
## 43   430         746.7     293.0     304.8
## 44   440         780.7     300.2     312.0
## 45   450         793.6     307.0     316.3
## 46   460         817.7     314.0     321.1
## 47   470         846.6     321.8     329.2
## 48   480         872.6     329.1     331.6
## 49   490         895.7     335.0     335.1
## 50   500         924.2     342.4     341.9
## 51   510         943.6     348.7     338.6
## 52   520         989.7     355.5     345.2
## 53   530        1040.0     363.1     361.7
## 54   540        1065.9     370.3     362.6
## 55   550        1098.9     376.7     368.3
## 56   560        1145.1     384.1     383.1
## 57   570        1176.4     390.1     389.6
## 58   580        1213.4     396.7     390.7
## 59   590        1262.7     404.6     401.9
## 60   600        1309.7     411.6     406.6
## 61   610        1356.0     420.8     416.9
## 62   620        1402.6     428.6     413.5
## 63   630        1410.1     435.7     411.5
## 64   640        1450.8     443.2     420.8
## 65   650        1492.4     451.0     427.3
## 66   660        1506.7     459.0     424.4
## 67   670        1523.1     466.9     431.6
## 68   680        1574.3     473.3     446.4
## 69   690        1613.4     481.2     447.1
## 70   700        1633.3     489.0     460.8
```

```
## 71   710        1675.1     495.4     472.5
## 72   720        1701.5     502.9     478.1
## 73   730        1775.3     510.3     487.3
## 74   740        1819.7     517.8     492.9
## 75   750        1846.7     525.4     502.9
## 76   760        1896.2     532.8     526.0
## 77   770        1960.2     541.0     543.6
## 78   780        1992.6     549.3     552.8
## 79   790        2018.8     556.4     566.5
## 80   800        2060.8     563.3     574.0
## 81   810        2118.2     571.7     582.7
## 82   820        2169.2     578.9     596.9
## 83   830        2208.6     587.6     608.2
## 84   840        2260.6     595.8     619.2
## 85   850        2294.4     603.0     635.4
## 86   860        2346.5     611.4     644.0
## 87   870        2348.9     619.0     647.3
## 88   880        2376.9     626.2     652.9
## 89   890        2430.9     634.0     669.8
## 90   900        2483.2     640.8     670.9
## 91   910        2514.3     648.3     666.0
## 92   920        2569.0     656.9     671.7
## 93   930        2635.2     664.0     678.4
## 94   940        2659.7     671.2     706.1
## 95   950        2746.5     679.3     703.9
## 96   960        2781.5     685.5     711.6
## 97   970        2887.1     693.5     727.7
## 98   980        2967.4     702.4     731.2
## 99   990        3027.4     710.0     746.6
## 100 1000        3084.0     718.7     767.8
```

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


Combined Scatter Plot
Comparisons vs Size
$y = 1160 + 8750\,x + 1700\,x^2$
$y = 355 + 2020\,x + 97.9\,x^2$
$y = 356 + 2100\,x + 165\,x^2$
Number of Comparisons (Averaged)
3000
2000
1000
0
0
250
500
750
1000
Number of Elements
variable
insertionSort
mergeSort
quickSort