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
Data Replicator	Combined Plotter

Data Generator and Replicator

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()
```

Data Replicator

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

```
replicator <- 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 = "Time 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 = "Time 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)
  op <- 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
      op <- op + 1
    }
    vec[pos + 1] <- key
    op <- op + 1
  }
  return (list("vec" = vec, "operations" = op))
}
```

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){
    op <- 0
    res <- c()
    while(length(left) > 0 && length(right) > 0){
      op <- op + 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)
    }
    op <- op + 1
    return (list("vec" = res, "operations" = op))
  }
```

```r
  op <- 0
  n <- length(vec)
  if(n <= 1) return (list("vec" = vec, "operations" = op))
  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)
    op <- op + left_list$operations + right_list$operations + res$operations
    return (list("vec" = res$vec, "operations" = op))
  }
}
```

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
    op <- 0
    pivot = vec[high]
    for(j in low:(high - 1)){
      op <- op + 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" = op, "pi" = i))
  }
  op <- 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

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

Proof of Concept

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

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

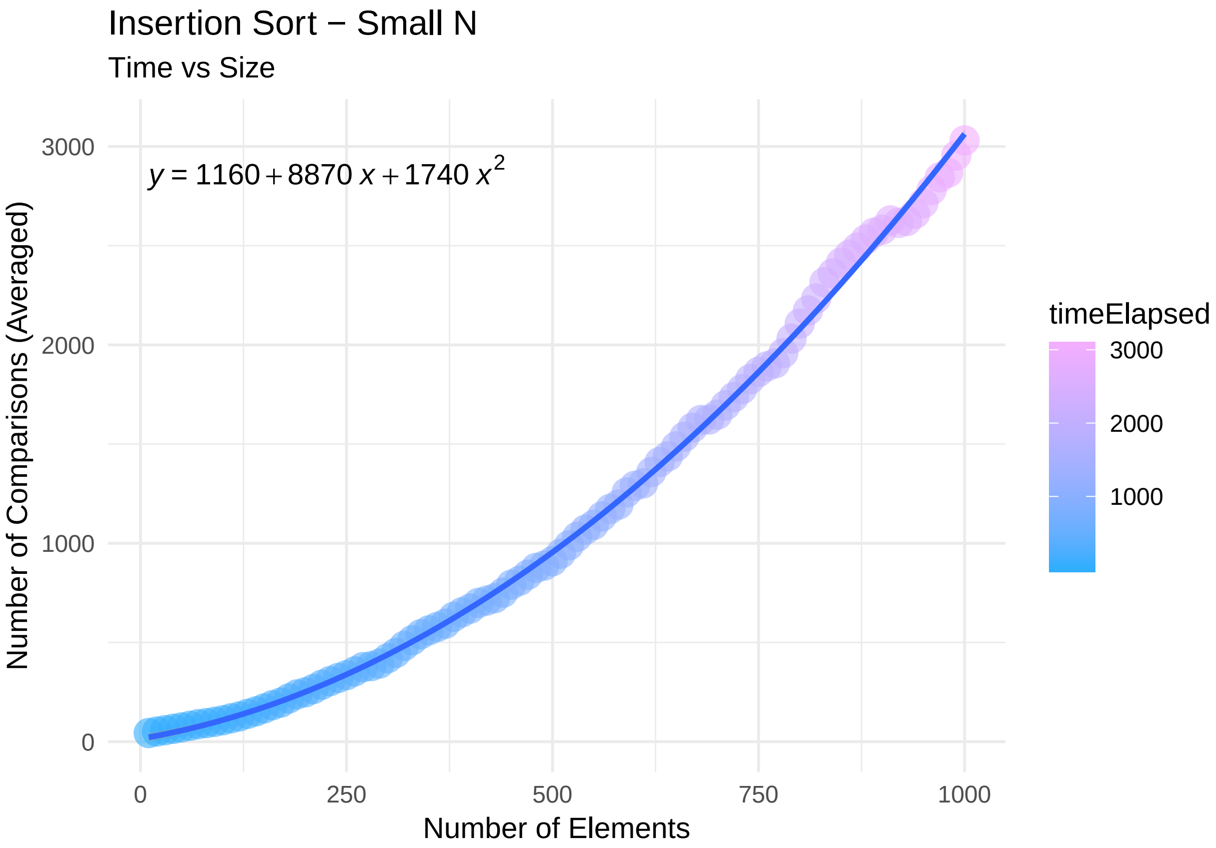
Sorting Algorithms - Plots

Individual Plots

Insertion Sort

```
isdf_small <- replicator(insertionSort)
plotter(isdf_small, "Insertion Sort - Small N")
```

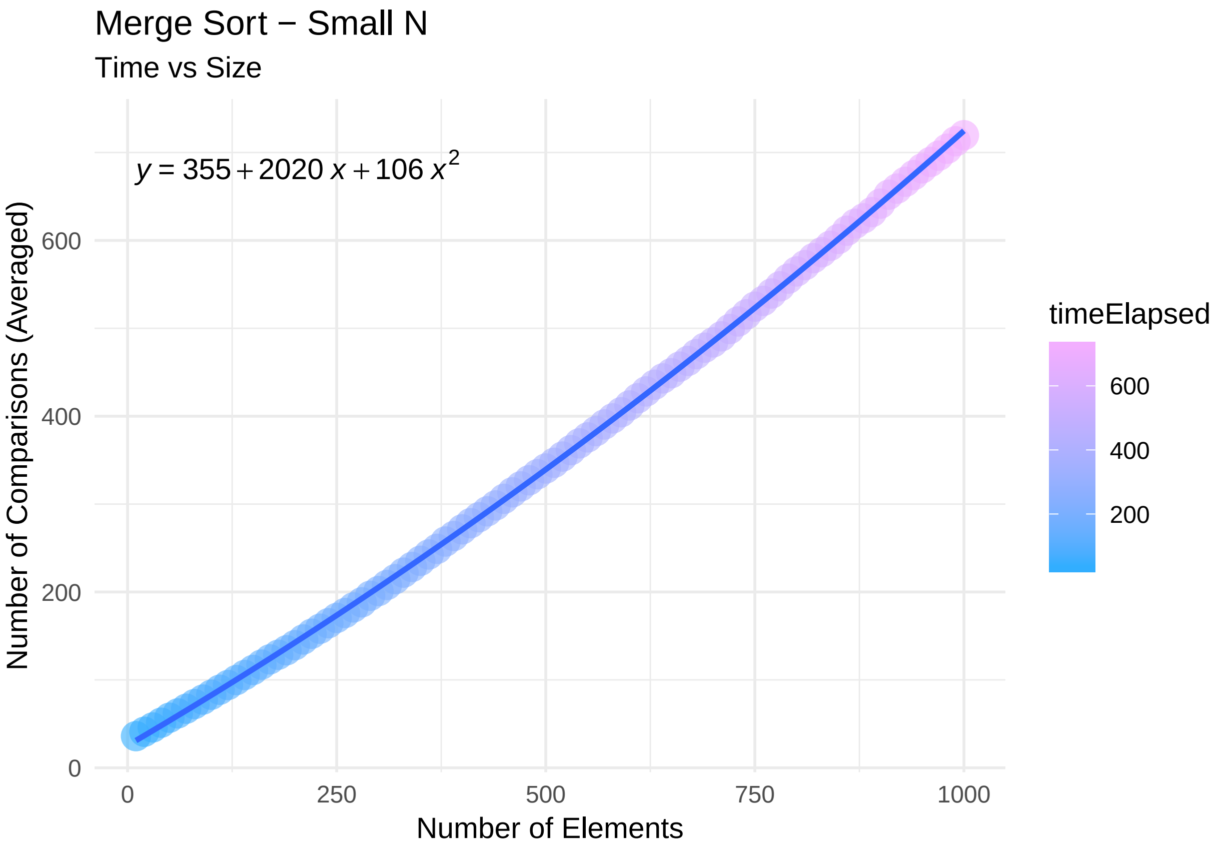
```
## Warning: Ignoring unknown parameters: rm
```



Merge Sort

```
msdf_small <- replicator(mergeSort)
plotter(msdf_small, "Merge Sort - Small N")
```

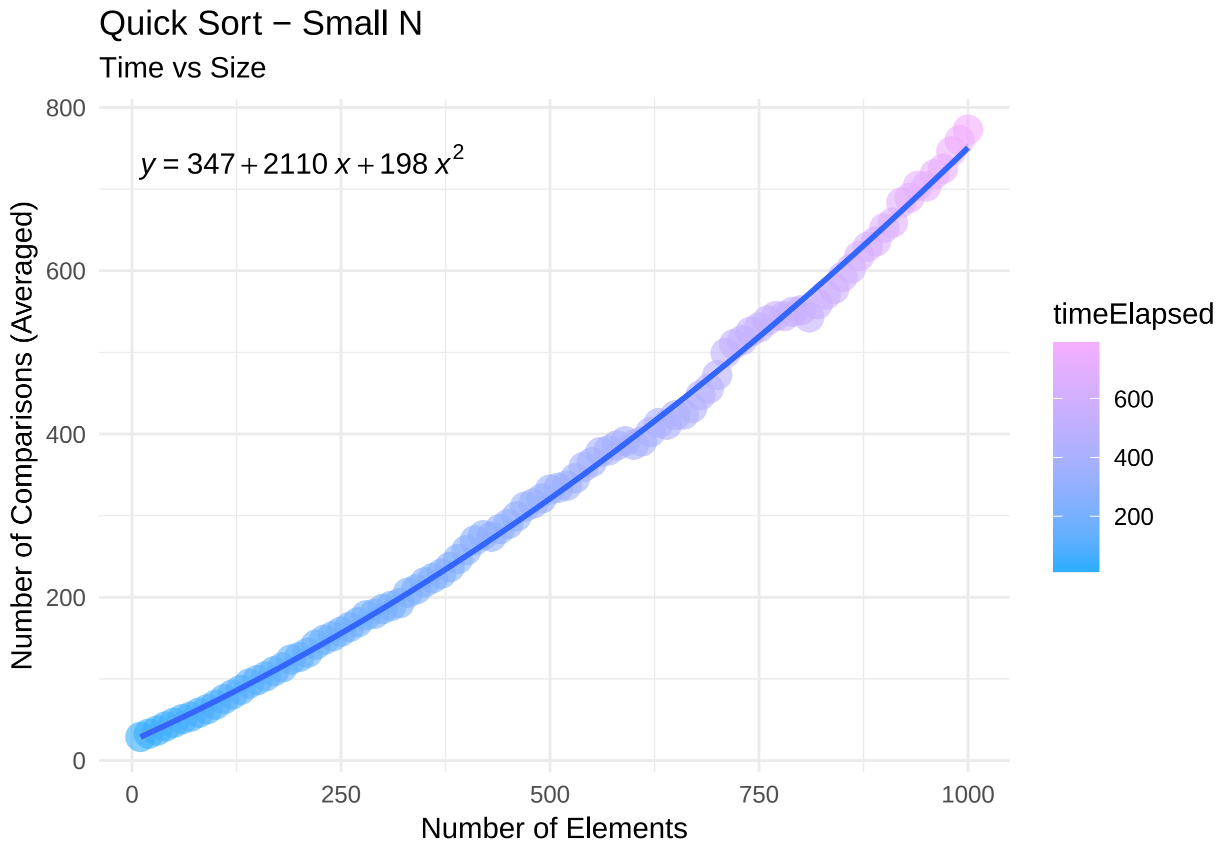
```
## Warning: Ignoring unknown parameters: rm
```



Quick Sort

```
qsdf_small <- replicator(quickSort)
plotter(qsdf_small, "Quick Sort - Small N")
```

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

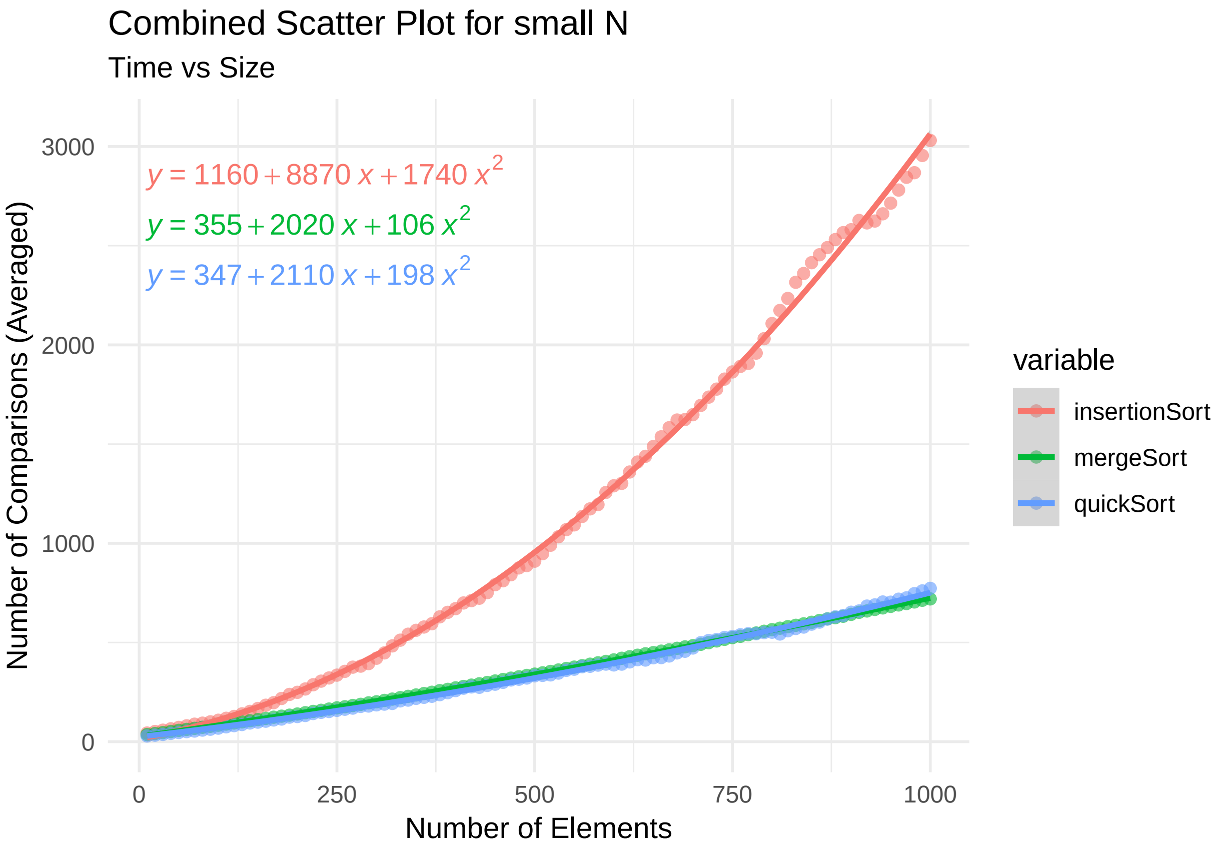


Combined Plots

```
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          43.7      36.0      28.9
## 2     20          51.4      41.0      32.8
## 3     30          57.9      45.9      36.6
## 4     40          65.0      51.3      41.7
## 5     50          72.0      57.0      46.2
## 6     60          80.0      61.9      50.6
## 7     70          88.2      67.2      53.6
## 8     80          93.5      72.5      58.5
## 9     90         100.0      77.6      62.8
## 10   100         108.1      83.1      68.3
## 11   110         117.9      88.8      75.0
## 12   120         127.2      94.2      80.9
## 13   130         140.3      99.8      86.5
## 14   140         152.6     105.6      94.2
## 15   150         167.4     111.3      98.5
## 16   160         183.7     117.4     103.2
## 17   170         196.6     123.4     109.4
## 18   180         217.8     128.6     114.1
## 19   190         238.3     133.7     123.7
## 20   200         248.8     139.5     126.8
## 21   210         265.6     146.0     132.0
## 22   220         287.0     152.5     142.0
## 23   230         305.1     158.2     147.9
## 24   240         321.2     164.5     152.2
## 25   250         335.4     170.5     157.8
## 26   260         354.4     176.1     163.5
## 27   270         375.6     182.5     169.3
## 28   280         380.8     188.3     177.8
## 29   290         393.6     195.7     179.7
## 30   300         420.7     201.1     185.7
## 31   310         447.5     208.0     189.3
## 32   320         482.8     214.6     192.8
## 33   330         511.6     222.0     205.6
## 34   340         541.7     228.5     209.6
## 35   350         561.3     235.5     218.2
## 36   360         578.0     242.7     223.5
## 37   370         594.6     249.1     228.9
## 38   380         629.2     257.4     237.0
## 39   390         651.9     263.7     246.9
## 40   400         670.6     271.3     257.6
## 41   410         699.2     278.3     269.6
## 42   420         710.9     285.1     276.0
## 43   430         723.1     291.8     274.2
## 44   440         751.1     298.5     283.6
## 45   450         791.3     306.0     289.8
## 46   460         811.1     313.4     299.0
## 47   470         841.5     320.2     311.0
## 48   480         875.9     327.1     314.8
## 49   490         887.6     334.0     320.9
## 50   500         909.8     340.5     331.9
## 51   510         947.6     347.0     334.2
## 52   520         990.2     354.1     336.7
## 53   530        1032.9     361.4     345.7
## 54   540        1068.8     369.1     359.6
## 55   550        1092.9     375.9     365.4
## 56   560        1135.2     383.0     377.7
## 57   570        1173.5     390.6     380.0
## 58   580        1195.0     397.5     386.0
## 59   590        1256.3     404.5     391.1
## 60   600        1289.8     412.2     386.9
## 61   610        1302.0     420.4     390.9
## 62   620        1359.3     428.2     402.1
## 63   630        1409.5     435.8     413.3
## 64   640        1438.4     443.0     411.5
## 65   650        1488.0     448.9     422.9
## 66   660        1536.7     456.4     423.9
## 67   670        1582.6     462.9     432.0
## 68   680        1621.3     470.6     447.4
## 69   690        1623.7     477.9     455.7
## 70   700        1648.3     484.0     472.2
## 71   710         1695.2     490.9     499.2
## 72   720         1736.8     499.2     510.2
## 73   730         1777.1     507.8     515.0
## 74   740         1827.6     515.9     525.4
## 75   750         1863.7     524.6     530.4
## 76   760         1891.6     531.2     539.0
## 77   770         1906.7     539.6     544.5
## 78   780         1958.6     547.8     544.3
## 79   790         2031.4     556.5     549.7
## 80   800         2107.7     564.7     551.6
## 81   810         2173.7     571.9     542.8
## 82   820         2234.2     579.8     558.9
## 83   830         2315.6     586.6     570.6
## 84   840         2360.0     593.9     577.8
## 85   850         2414.4     601.7     592.4
## 86   860         2454.4     611.0     602.7
## 87   870         2490.5     619.0     618.1
## 88   880         2530.9     625.1     629.4
## 89   890         2565.9     632.2     636.0
## 90   900         2580.5     641.8     652.5
## 91   910         2627.3     652.0     659.1
## 92   920         2614.9     658.9     683.5
## 93   930         2624.3     666.7     689.4
## 94   940         2660.7     674.3     704.1
## 95   950         2714.5     682.0     703.2
## 96   960         2780.5     689.3     718.2
## 97   970         2844.7     696.4     725.0
## 98   980         2868.0     704.3     746.3
## 99   990         2954.8     712.8     760.0
## 100  1000        3030.7     719.7     772.8
```

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


Combined Scatter Plot for small N
Time vs Size
y = 1160 + 8870 x + 1740 x²
y = 355 + 2020 x + 106 x²
y = 347 + 2110 x + 198 x²
Number of Comparisons (Averaged)
0
1000
2000
3000
0
250
500
750
1000
Number of Elements
variable
insertionSort
mergeSort
quickSort