COMP1921: PROGRAMMING PROGRAMMING PROJECT: BENCHMARKING SORTING ALGORITHMS REPORT

CONTENTS

DESIGN DOCUMENT

---

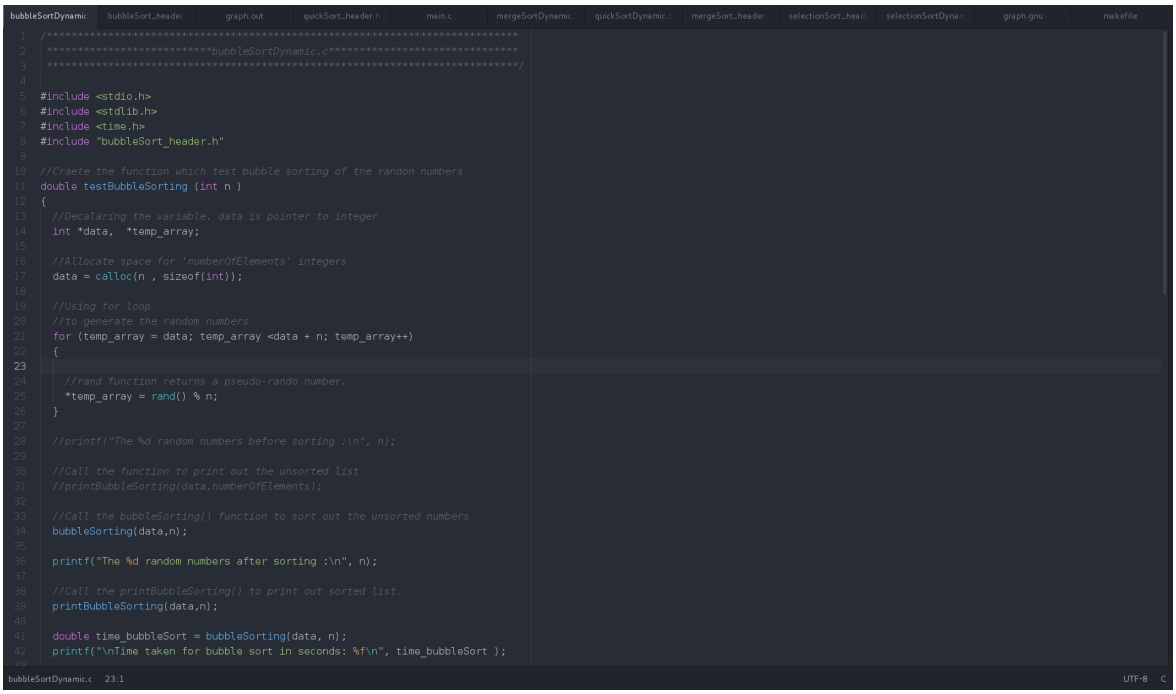
1. Beginning with the minimal stage of creating the module files, I began with the module files because it will be easier for me to see what I will define later and will work on, this will also make it much easier to create my makefile. Below I have included the module file screen shots along with the fully created makefile:

In make file, I added the modules files and header file

```
# dependencies

main.o: main.c  bubbleSort_header.h quickSort_header.h  selectionSort_header.h
bubbleSortDynamic.o:  bubbleSortDynamic.c bubbleSort_header.h
quickSortDynamic.o: quickSortDynamic.c  quickSort_header.h
selectionSortDynamic.o: selectionSortDynamic.c  selectionSort_header.h
mergeSortDynamic.o: mergeSortDynamic.c  mergeSort_header.h
```

2. The next stage I have included is my first method of sorting, minimally done, this is the bubble sort method, not comprehensively complete however it works:

bubbleSortDynamic.c



```
/*******************************************************************************
*************************bubbleSortDynamic.c*********************************
*******************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "bubbleSort_header.h"

//Craete the function which test bubble sorting of the randon numbers
double testBubbleSorting (int n )
{
  //Decalaring the variable. data is pointer to integer
  int *data,  *temp_array;

  //Allocate space for 'numberOfElements' integers
  data = calloc(n , sizeof(int));

  //Using for loop
  //to generate the random numbers
  for (temp_array = data; temp_array <data + n; temp_array++)
  {

    //rand function returns a pseudo-rando number.
    *temp_array = rand() % n;
  }

  //printf("The %d random numbers before sorting :\n", n);

  //Call the function to print out the unsorted list
  //printBubbleSorting(data,numberOfElements);

  //Call the bubbleSorting() function to sort out the unsorted numbers
  bubbleSorting(data,n);

  printf("The %d random numbers after sorting :\n", n);

  //Call the printBubbleSorting() to print out sorted list.
  printBubbleSorting(data,n);

  double time_bubbleSort = bubbleSorting(data, n);
  printf("\nTime taken for bubble sort in seconds: %f\n", time_bubbleSort );
```

bubbleSort_header.h

```
/*****************************************************************************
 ***************************bubbleSort_header.h*****************************
 *****************************************************************************/

double testBubbleSorting (int n );

void swapForBubbleSort ( int *p, int *q);

double bubbleSorting ( int *data, int numberOfElements );

void printBubbleSorting ( int *data, int numberOfElements );
```

---

3. This stage includes the quick sort stage, just like the bubble sort it works however remains fully complete, this is just for this stage as it requires for it to work, I have included this below:

quickSortDynamic.c

```
/*****************************************************************************
 ***************************quickSort_header.h******************************
 *****************************************************************************/

void swapForQuickSort(int *x,int *y);

int choose_pivot(int i,int j );

double quickSorting(int *list,int m,int n);

void printQuickSorting(int *list,const int n);

double testQuickSorting ( int n );

            while((j >= m) && (list[j] > key))
                j--;
            if( i < j)
                swapForQuickSort(&list[i],&list[j]);
        }
        /* swapForQuickSort two elements */
        swapForQuickSort(&list[m],&list[j]);

        /* recursively sort the lesser list */
        quickSorting(list,m,j-1);
        quickSorting(list,j+1,n);
    }
    end_t = clock();
    //Return the time
    return (double)(end_t - start_t)/CLOCKS_PER_SEC;
}
void printQuickSorting(int *list,int n)
{
    int i;
    for(i=0; i<n; i++)
        printf("%d\t",list[i]);
}
```

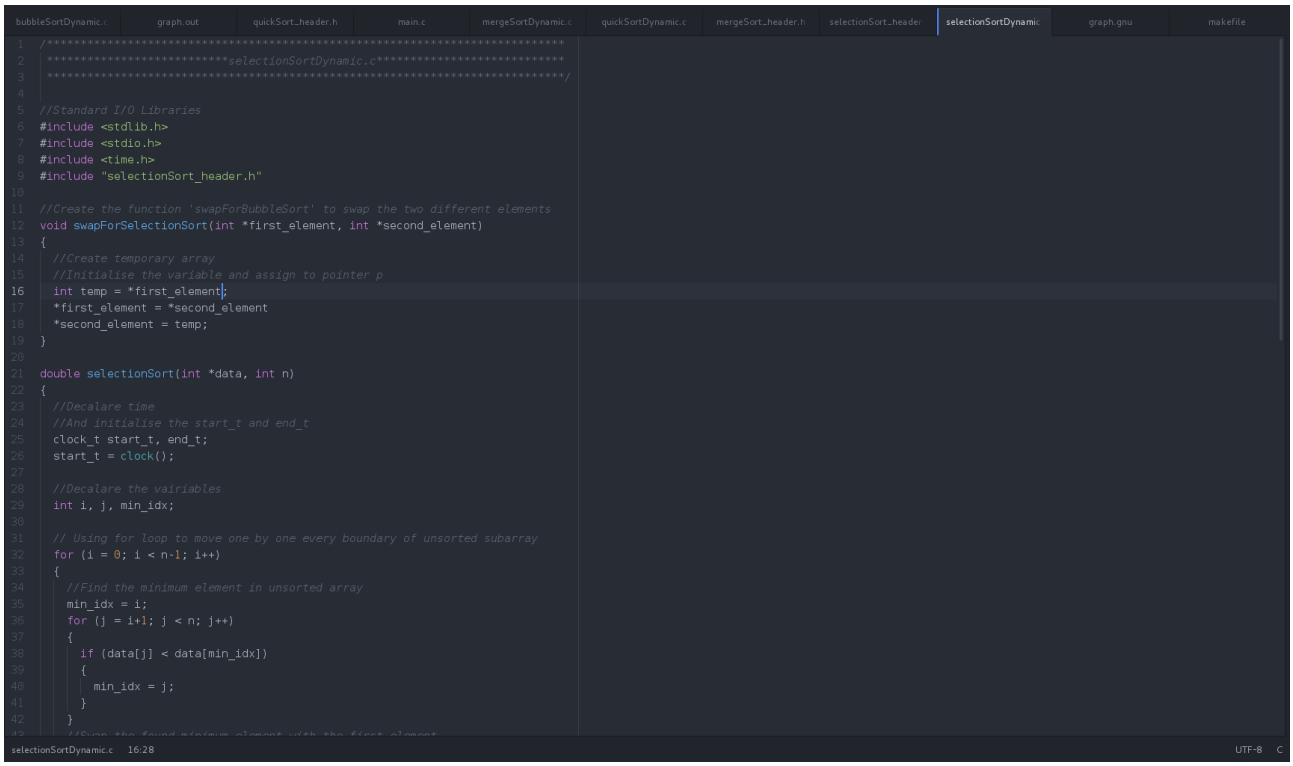
quickSort_header.h

```
/*****************************************************************************
 ***************************quickSort_header.h******************************
 *****************************************************************************/

void swapForQuickSort(int *x,int *y);

int choose_pivot(int i,int j );

double quickSorting(int *list,int m,int n);

void printQuickSorting(int *list,const int n);

double testQuickSorting ( int n );
```

---

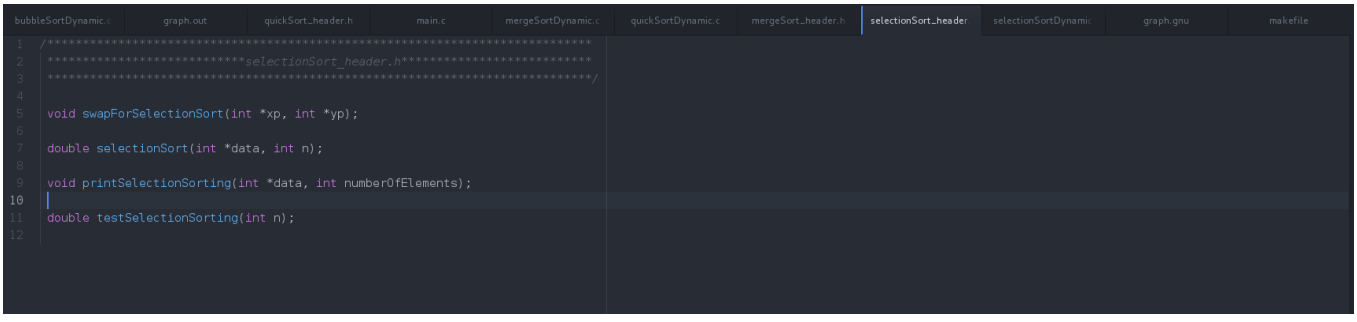
4. This stage includes my selection sorting optional sort which is the selection sort, this stage is close to completed, it works, however not comprehensively done so:

selectionSortDynamic.c



```
/*****************************************************************************
 ************************selectionSortDynamic.c***************************
 *****************************************************************************/

//Standard I/O Libraries
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include "selectionSort_header.h"

//Create the function 'swapForBubbleSort' to swap the two different elements
void swapForSelectionSort(int *first_element, int *second_element)
{
  //Create temporary array
  //Initialise the variable and assign to pointer p
  int temp = *first_element;
  *first_element = *second_element
  *second_element = temp;
}

double selectionSort(int *data, int n)
{
  //Decalare time
  //And initialise the start_t and end_t
  clock_t start_t, end_t;
  start_t = clock();

  //Decalare the vairiables
  int i, j, min_idx;

  // Using for loop to move one by one every boundary of unsorted subarray
  for (i = 0; i < n-1; i++)
  {
    //Find the minimum element in unsorted array
    min_idx = i;
    for (j = i+1; j < n; j++)
    {
      if (data[j] < data[min_idx])
      {
        min_idx = j;
      }
    }
```

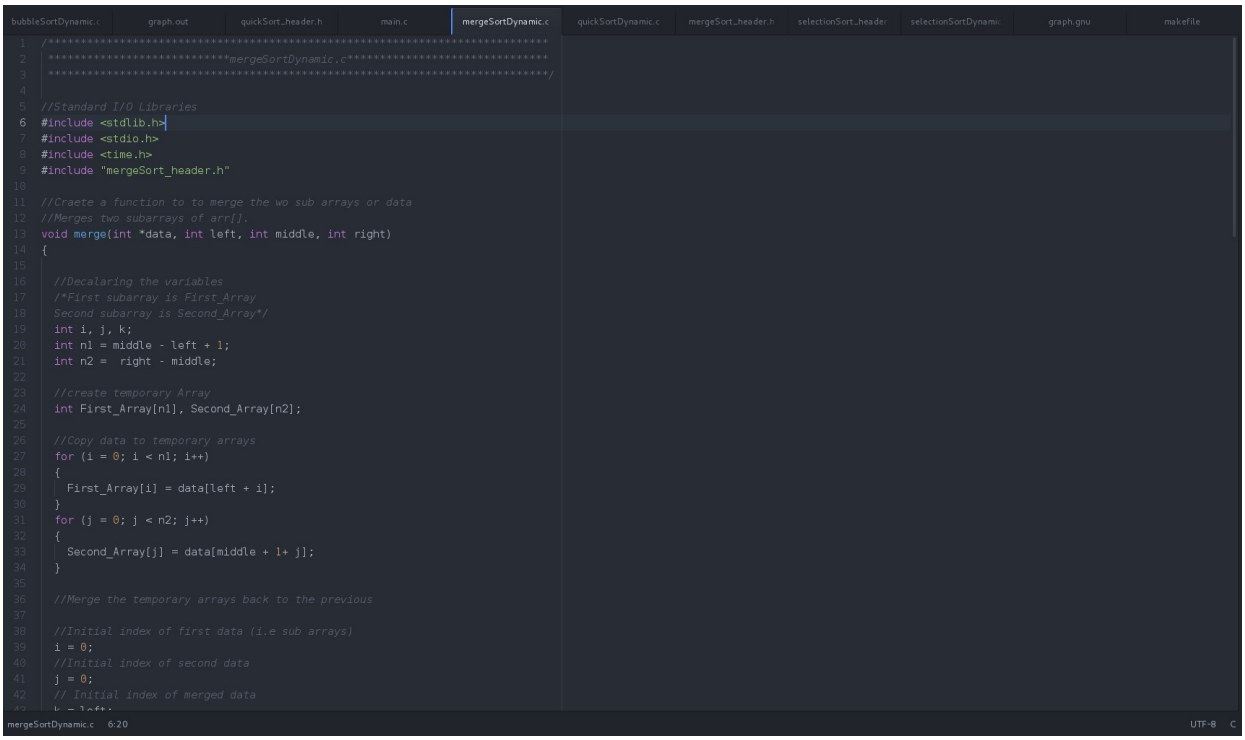
selectionSort_header.h



```
/*****************************************************************************
 ************************selectionSort_header.h***************************
 *****************************************************************************/

void swapForSelectionSort(int *xp, int *yp);

double selectionSort(int *data, int n);

void printSelectionSorting(int *data, int numberOfElements);

double testSelectionSorting(int n);
```

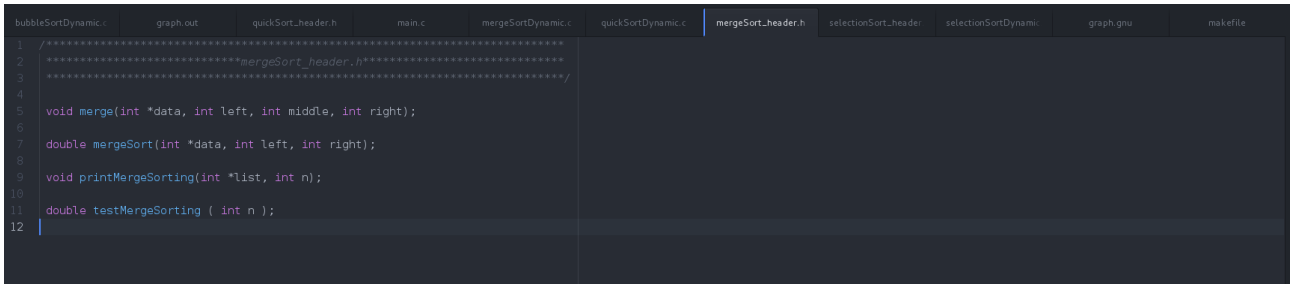
5. This stage includes my selection sorting optional sort which is the selection sort, this stage is close to completed, it works, however not comprehensively done so:

mergeSortDynamic.c



```c
/*****************************************************************************
 ***************************mergeSortDynamic.c*****************************
 *****************************************************************************/

//Standard I/O Libraries
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include "mergeSort_header.h"

//Craete a function to to merge the wo sub arrays or data
//Merges two subarrays of arr[].
void merge(int *data, int left, int middle, int right)
{

  //Decalaring the variables
  /*First subarray is First_Array
  Second subarray is Second_Array*/
  int i, j, k;
  int n1 = middle - left + 1;
  int n2 =  right - middle;

  //create temporary Array
  int First_Array[n1], Second_Array[n2];

  //Copy data to temporary arrays
  for (i = 0; i < n1; i++)
  {
    First_Array[i] = data[left + i];
  }
  for (j = 0; j < n2; j++)
  {
    Second_Array[j] = data[middle + 1+ j];
  }

  //Merge the temporary arrays back to the previous

  //Initial index of first data (i.e sub arrays)
  i = 0;
  //Initial index of second data
  j = 0;
  // Initial index of merged data
```

mergeSort_header.h



```c
/*****************************************************************************
 ***************************mergeSort_header.h*****************************
 *****************************************************************************/

void merge(int *data, int left, int middle, int right);

double mergeSort(int *data, int left, int right);

void printMergeSorting(int *list, int n);

double testMergeSorting ( int n );
```

6. This is the intermediate stage of my design, I have done the bubble sort, quick sort and selection sort to full detail, they all work and I have also began to add the commenting into my program, this also ensures that I have began to do things to near completion, below are the screen shots of all the 3 algorithms I have used for sorting. They all work and run, as compared to the previous stage, these are much more comprehensive.

main.c

7. I have no began to near the final stages of my program, this shows that I have finished commenting and now also ran tests on my program to see whether it works or not, and now that they all work and commenting is finished/near completion, I will now begin to debug my program and do tests to see if it can break to be comprehensive with it.

graph.gnu

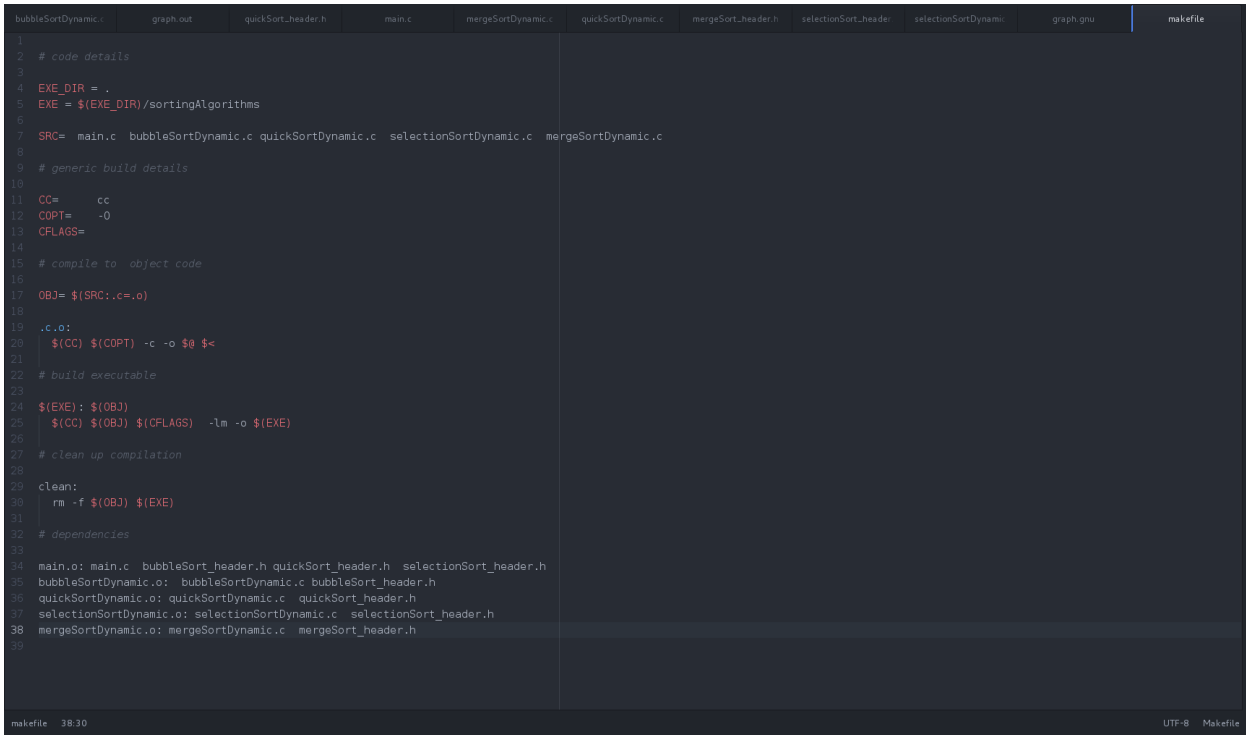
```
bubbleSortDynam | graph.out | bubbleSort_heade | graph.gnu | header.h | main.c | makefile | mergeSortDynam | mergeSort_heade | quickSortDynamic | quickSort_header | selectionSortDyn | selectionSort_hea

1  ###############################################################################
2  #                                                                             #
3  #                                GNU FILE                                     #
4  #                                                                             #
5  ###############################################################################
6
7
8
9  set title 'Graph for Sorting'    #set title
10 set xlabel 'Length'               #set x axis
11 set ylabel 'Time in seconds'      #set y axis
12 plot 'graph.out' u 1:2 w lp t 'Quick Sort', 'graph.out' u 1:3 w lp t 'Merge Sort', 'graph.out' u 1:4 w lp t 'Selection Sort', 'graph.out' u 1:5 w lp t 'Bubble Sort'
13
```

Below I have included additional screen shots for my final finished design of the program, this includes the modules and how the program is layed out, and my finalised touches that were added to it.

In main.c, I have provided the choice to select the sorting type

```
bubbleSortDynamic.c | graph.out | quickSort_header.h | main.c | mergeSortDynamic.c | quickSortDynamic.c | mergeSort_header.h | selectionSort_header | selectionSortDynamic | graph.gnu | makefile

44    //Using while loop to choose the option for sorting
45    while(1)
46    {
47      printf("\nChoose the sorting type:\n\n");
48      printf("1. Quick sort\n");
49      printf("2. Merge sort\n");
50      printf("3. Selection sort\n");
51      printf("4. Bubble sort\n");
52      printf("5. Exit\n");
53      printf("6. For another Number of Elements\n");
54
55      scanf("%d", &choice);
56      printf("Option: %d\n", choice);
57      if ( choice > 0 && choice < 6)
58      {
59        switch (choice)
60        {
61          case 1:
62            time_quickSort = testQuickSorting(n);
63          break;
64          case 2:
65            time_mergeSort = testMergeSorting(n);
66          break;
67          case 3:
68            time_quickSort = testSelectionSorting(n);
69          break;
70          case 4:
71            time_bubbleSort = testBubbleSorting(n);
72          break;
73          case 5:
74            printf("Exit!!!!!!!!!\n");
75            //system("PAUSE");
76            exit(0);
77        }
78        fprintf(fp, "%d %f %f %f\n", n, time_quickSort, time_mergeSort, time_selectionSort, time_bubbleSort );
79      }
80      else if (choice == 6)
81      {
82        printf("\nGive the number of elements : ");
83        scanf("%d", &n);
84        counter++;
85      }

main.c  21:18                                                                   UTF-8  C
```

makefile


bubbleSortDynamic.c
graph.out
quickSort_header.h
main.c
mergeSortDynamic.c
quickSortDynamic.c
mergeSort_header.h
selectionSort_header
selectionSortDynamic
graph.gnu
makefile
# code details
EXE_DIR = .
EXE = $(EXE_DIR)/sortingAlgorithms
SRC= main.c bubbleSortDynamic.c quickSortDynamic.c selectionSortDynamic.c mergeSortDynamic.c
# generic build details
CC= cc
COPT= -O
CFLAGS=
# compile to object code
OBJ= $(SRC:.c=.o)
.c.o:
$(CC) $(COPT) -c -o $@ $<
build executable
$(EXE): $(OBJ)
$(CC) $(OBJ) $(CFLAGS) -lm -o $(EXE)
clean up compilation
clean:
rm -f $(OBJ) $(EXE)
dependencies
main.o: main.c bubbleSort_header.h quickSort_header.h selectionSort_header.h
bubbleSortDynamic.o: bubbleSortDynamic.c bubbleSort_header.h
quickSortDynamic.o: quickSortDynamic.c quickSort_header.h
selectionSortDynamic.o: selectionSortDynamic.c selectionSort_header.h
mergeSortDynamic.o: mergeSortDynamic.c mergeSort_header.h
makefile 38:30
UTF-8 Makefile

PURPOSE OF THE MODULES

- **bubbleSortDynamic.c:** This is my bubble sort algorithm, in this I am using a dynamic array to sort out the random unsorted numbers.

- **bubbleSort_header.h:** Decleration of the variables that used bubbleSortDynamic.c.

- **quickSortDynamic.c**: This is my quick sort algorithm, in this I am also using dynamic array to sort out the random unsorted numbers.

- **quickSort_header.h:** This is the decleration of the variables that I used in quickSortDynamic.c

- **selectionSortDynamic.c:** This is my selection sorting alogrithm, used for the same purpose as to sort unsorted numbers in an array.

- **selectionSort_header.h:** This is the declaration of the variables being used in the selectionSortDynamic.c

- **main.c:** This is the main file, in this I am calling all the sorting functions, allowing the user choice of control to choose an sorting algorithm to sort a list of numbers.

- **graph.gnu:** Allows the plotting of the program data to the gnu plot graph, seen and represented in 2D plots.

FUNCTIONS USED IN EACH MODULE

- **bubbleSortDynamic.c:**

  - swapForBubbleSort: I have created a function void swapForBubbleSort() with two arguments, allows to swap the two different elements or arrays.

  - bubbleSorting: The function double bubbleSorting() with two arguments, allows to implement bubble sorting algorithm.

  - printBubbleSorting: The function void printBubbleSorting() with two arguments, allows to print out the unsorted random numbers and after applying the bubble sorting function, it will sort out the randomly sorted numbers. However it is not important to use this to print out the whole list in the terminal.

  - testBubbleSorting: The function double testBubbleSorting() with 1 argument, which is creating a unsorted random number, and then calling the bubble sorting function inside it, this is called to the main.c.

- **selectionSortDynamic.c:**

  - swapForSelectionSort: The function is void swapForSelectionSort() has two arguments which allow to swap between two different elements.

  - selectionSort: The function is void selectionSort() has two arguments which implements the selection sort, inside this function, swapForSelection was called into this.

  - printSelectionSorting: The function void printSelectionSorting() has 2 arguments, which allow user to print out the unsorted list and then allow it to reprint to sorted list.

  - testSelectionSorting: The function void testSelectionSorting() has 1 argument, this calls the selectionSort and this function is called inside the main.

- **quickSortDynamic.c:**

  - swapForQuickSort : The function is void swapForQuickSort() has 2 arguments which allows to swap between two different elements.

  - choosePivot : The function void choosePivot() has 2 arguments which takes last element as pivot, places the pivot element at its correct position in sorted array.

- quickSorting : The function double quickSorting() has 3 arguments which Implements the main method for quick sort. Inside this function, choosePivot() and swapForQuickSort() are used to implements the quick sorting algorithm.

- printQuickSorting : The function void printQuickSorting() has 2 arguments, which allows user to print out the unsorted list and then allow it to reprint to reprint to sorted list.

- testQuickSorting: The function void testQuickSorting() has 1 argument, this calls the quickSorting() and printQuickSorting(). This function is called inside the main file.

- **MergeSortDynamic.c :**

  - merge : The function void merge() has 3 arguments which allow to merge two sub arrays of a array. This function implements the method for merge sorting

  - mergeSort : The function double mergeSort() has 3 arguments which sort the first array and then second half and then it calls the merge() function to implement the sorting.

  - printMergeSorting : The function void printMergeSorting() has 2 arguments, which allows user to print out the unsorted list and then allow it to reprint to reprint to sorted list.

  - testMergeSorting: The function void testMergeSorting() has 1 argument, this calls the quickSorting() and printQuickSorting(). This function is called inside the main file.

- **main.c:**

  - Main calls all the functions inside it and also allows for the user input to call the specific sorting algorithms and also allows the user to then run and test the sorting algorithms through the terminal.

- **graph.gnu :**

  -graph.gnu allow to print out the result as graph on gnuplot, where in graph x-axis is Length and y-axis is Time in seconds.

The libraries I have used within this program include:

- **#include <stdio.h>** : The stdio.h header defines three variable types, several macros, and various functions for performing input and output.

- **#include <stdlib.h> :** stdlib.h is the header of the general purpose standard library of C programming language which includes functions involving memory allocation, process control, conversions and others.

- **#include <time.h>** : The time.h header defines four variable types, two macro and various functions for manipulating date and time.

- **#include "bubbleSort_header.h" :** In this header file I declared the variables which are used in bubbleSortDynamic.c

- **#include "quickSort_header.h" :** In this header file I declared the variables which are used in bubbleSortDynamic.c

- **#include "mergeSort_header.h" :** In this header file I declared the variables which are used in mergeSortDynamic.c

- **#include "selectionSort_header.h" :** In this header file I declared the variables which are used in selectionSortDynamic.c

---

External code that I have used from public online repositories includes:

References:

- http://www.geeksforgeeks.org/
- http://www.mzan.com/article/26128092-i-am-getting-a-segmentation-fault-core-dumped-error-with-dynamically-allocated.shtml
- https://www.youtube.com/watch?v=Jdtq5uKz-w4
- http://gnuplot.sourceforge.net/docs_4.2/node184.html
- http://alvinalexander.com/technology/gnuplot-charts-graphs-examples

---

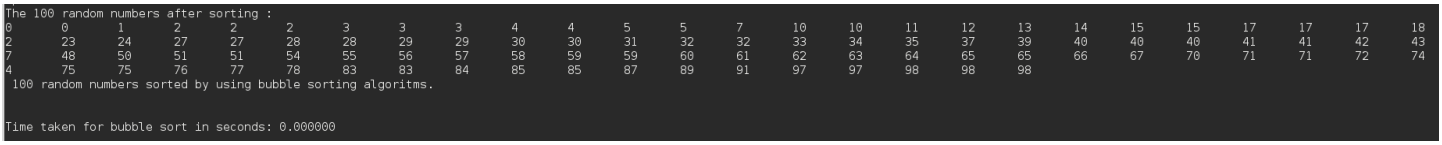
TEST PLAN

My initial thoughts on how the tests would run was, minor faults at the start however it should run fine and perfectly, I have planned to test this from the beginning as if it would work step by step, so I would be able to sort out the errors one by one, this would leave room for much less errors and help me find the errors much more efficiently and debug my code this way.

Running through the test phase one of my program this is what happened with Bubble Sort:

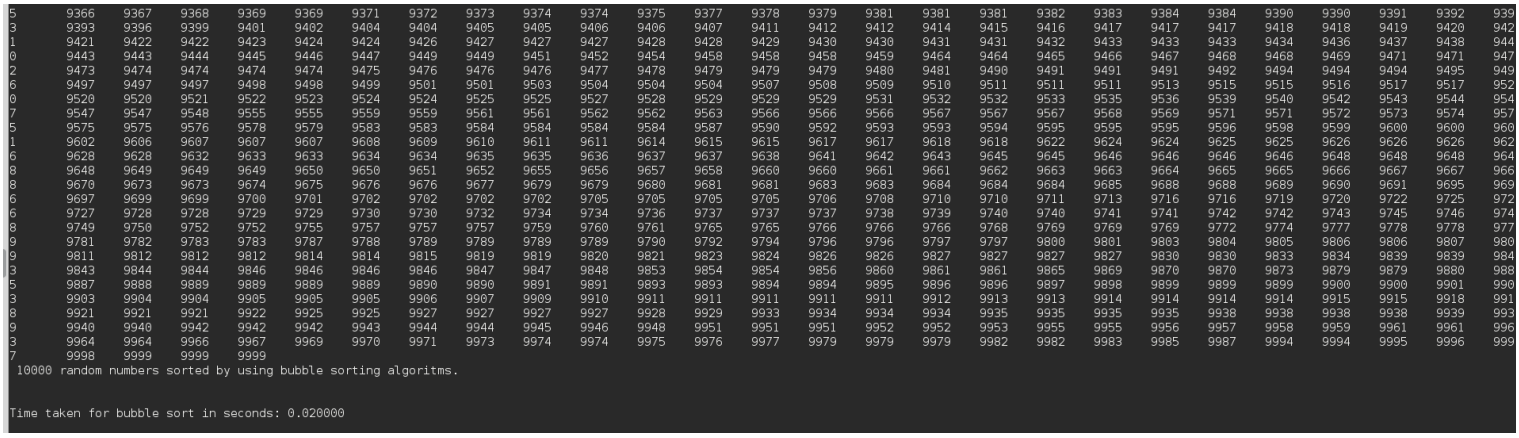
Bubble Sort 100 Words:

After execution the bubble sort algorithm displays that the time taken to process 100 random numbers was 0.000 seconds; this is also dependant on the computer and their process.



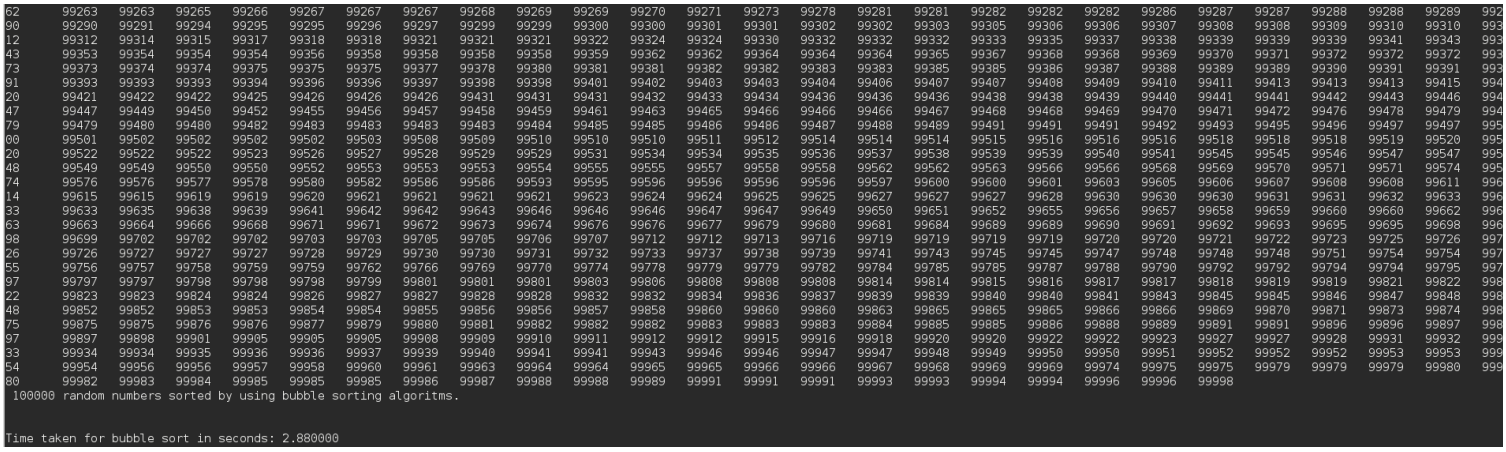
Bubble Sort 10,000 Words:

After execution the bubble sort algorithm displays that the time taken to process 10,000 random numbers was 0.02000 seconds.



Bubble Sort 100,000 Words:

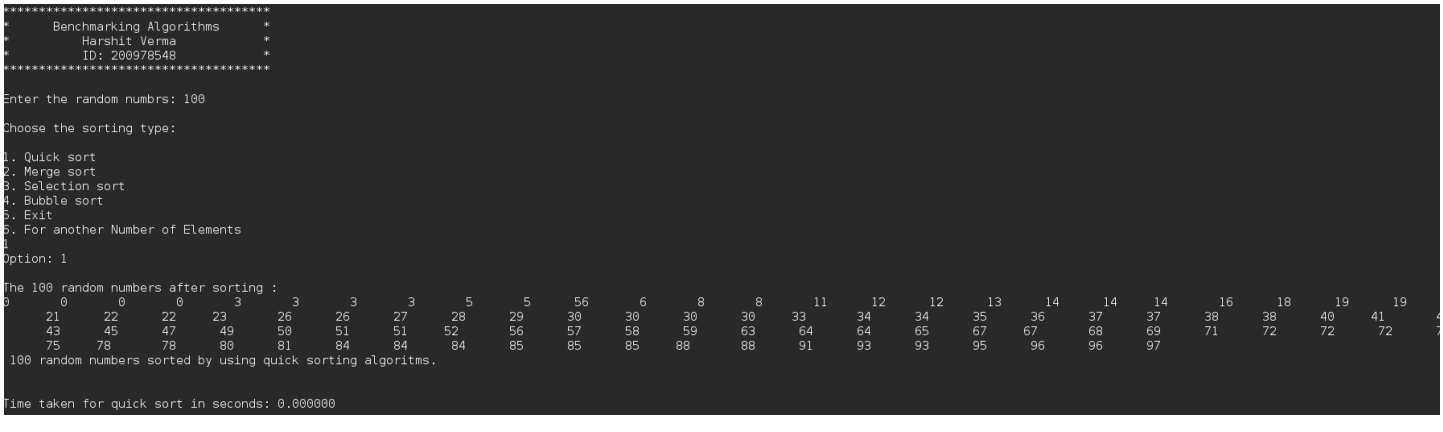
After execution the bubble sort algorithm displays that the time taken to process 10,000 random numbers was 2.8800 seconds.



Running through the test phase of the program with Quick Sort, the results are as shown

Quick Sort 100 Words:

After execution the quick sort algorithm displays that the time taken to process 100 random numbers was 0.000 seconds; this is due to the algorithm for quick sort is too quick, this is also dependant on the computer.



Quick Sort 1000 Words:

After execution the quick sort algorithm displays that the time taken to process 1000 random numbers was 0.000 seconds as same as for 100 numbers; this is due to the algorithm for quick sort is too quick, this is also dependant on the computer.

```
Option: 1

The 1000 random numbers after sorting :
0       1       2       3       3       6       6       6       7       7       7       8       9       11      12      12      13      14      14      14      15      15      16      17
8       18      20      21      22      24      26      29      30      30      31      32      32      33      34      36      38      39      39      40      41      43      46      46
8       49      51      52      53      53      53      55      58      58      60      60      61      61      61      62      62      63      64      68      68      70      70      71
6       78      79      79      81      81      81      84      85      86      87      87      88      89      89      90      91      91      92      92      93      93      97      97
01      101     102     104     106     108     109     110     113     114     115     116     119     120     120     120     121     121     122     122     123     125     125     125
131     132     133     134     136     138     139     140     141     143     144     149     149     152     152     153     153     153     154     154     154     155     155     156
158     159     161     161     163     163     166     166     166     166     167     167     167     169     169     170     171     173     173     174     174     175     176     178
180     180     181     181     182     183     184     185     186     187     189     190     191     192     193     194     194     195     195     195     196     196     198     198
205     205     206     206     206     207     207     208     209     213     213     215     215     216     216     218     218     220     221     221     222     223     224     225
229     233     234     236     236     237     238     238     238     239     239     239     241     242     242     243     245     250     251     255     255     261     263     264
270     270     272     274     275     275     275     276     276     278     279     279     283     283     284     285     286     287     288     290     290     290     295     299
300     305     307     308     308     310     310     310     314     315     315     315     317     317     317     318     322     322     322     323     328     332     334     335
344     344     349     349     351     351     352     353     353     354     354     356     356     356     358     359     359     360     360     360     361     362     363     363
367     368     371     372     373     374     374     375     376     376     376     376     379     379     381     381     382     384     386     387     389     390     392     393
394     395     397     398     398     399     399     401     405     406     409     410     410     410     410     411     412     412     412     413     414     415     416     416
424     429     430     431     431     431     432     432     436     436     437     438     438     439     440     443     444     446     447     448     449     449     451     454
456     458     458     458     458     459     460     460     460     461     461     462     463     463     463     464     464     464     465     465     465     465     467     469
471     472     473     473     474     475     475     475     476     476     477     478     478     478     479     480     482     482     483     486     487     487     488     490
493     494     495     495     495     496     500     500     502     503     504     505     506     507     510     514     514     515     515     515     516     516     516     517
525     527     530     530     532     532     535     536     536     536     536     537     537     541     542     544     545     545     546     547     548     550     550     552
556     556     557     557     558     560     560     562     562     562     563     564     564     567     568     571     571     573     574     576     576     577     577
580     581     582     583     583     584     586     586     586     587     587     587     587     588     588     588     589     589     589     591     591     591     591     591
598     599     599     599     604     604     605     606     607     607     607     610     611     612     612     614     616     617     618     618     620     620     621     621
625     626     626     627     627     630     632     633     633     633     634     635     635     636     636     637     637     637     638     638     639     639     640     641
643     644     644     645     648     648     651     651     651     651     652     654     654     655     656     657     660     660     661     661     662     663     664
667     668     671     671     671     672     672     674     676     678     679     679     679     680     681     683     683     686     687     687     690     691     692     692
697     700     700     706     706     708     709     709     710     715     715     720     721     722     722     726     726     727     729     730     730     730     731
738     739     739     741     741     742     742     743     744     744     745     745     746     747     748     748     748     748     749     751     751     751     754     755
757     758     759     760     763     764     765     766     766     767     768     770     772     773     776     777     778     778     779     781     783     783     784     784
789     790     792     793     793     794     796     801     801     802     802     803     803     806     807     808     808     810     810     811     812     812     812     813
818     819     820     821     823     823     823     824     825     826     827     827     828     829     829     829     829     830     831     831     832     832     832     834
839     839     839     840     841     841     842     842     843     843     843     843     844     845     845     846     847     849     849     852     853     854     854     856
859     863     864     864     864     865     865     865     866     868     869     869     870     871     872     873     873     874     879     879     879     880     881     881
885     886     887     887     889     890     890     891     894     894     895     896     896     897     898     900     900     901     902     904     904     905     905     908
911     911     912     913     914     915     917     917     918     921     922     923     923     926     927     927     930     933     936     938     938     939     940     940
944     945     946     950     950     951     953     954     959     959     959     960     960     960     961     961     962     963     964     965     965     970     971     971
973     974     975     977     978     979     981     981     982     982     982     982     983     984     985     986     986     987     991     992     993     993     994     995
998
 1000 random numbers sorted by using quick sorting algoritms.


Time taken for quick sort in seconds: 0.000000
```

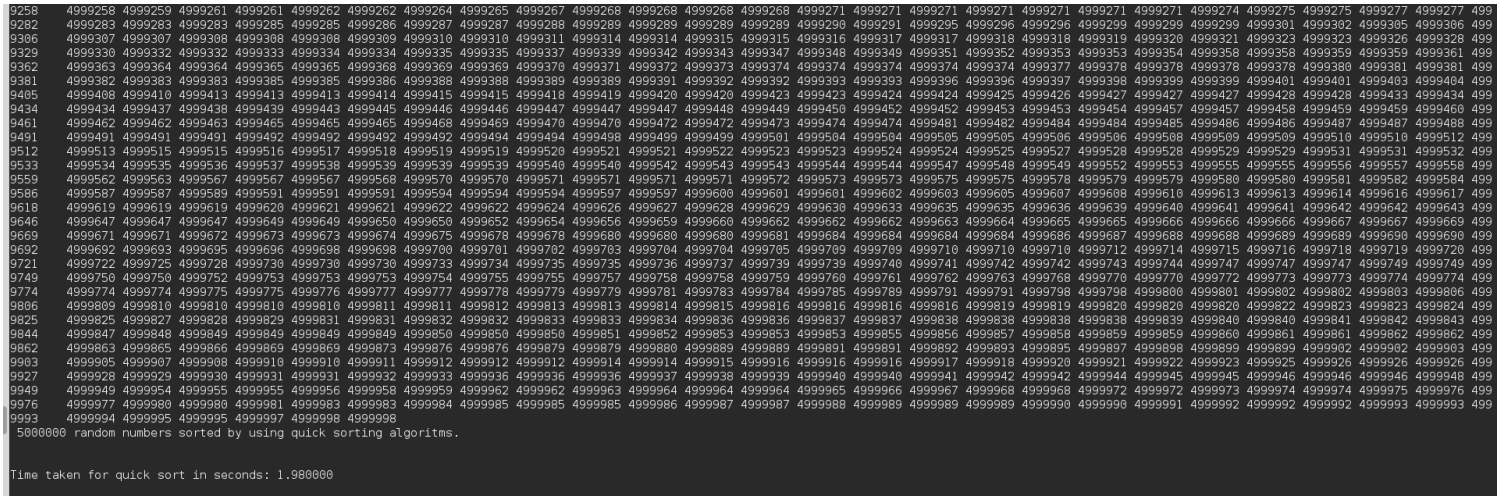
Quick Sort 1,000,000 Words:

After execution the quick sort algorithm displays that the time taken to process 1,000,000 random numbers was 0.39000 seconds.

```
378     999378  999379  999380  999381  999381  999383  999383  999383  999384  999384  999386  999388  999388  999388  999392  999392  999393  999394  999397  999398  999398  999398  999399  999399  999399  999
400     999400  999401  999401  999404  999405  999406  999407  999407  999408  999408  999409  999409  999411  999413  999417  999420  999420  999422  999423  999424  999427  999428  999428  999428  999429  999
430     999431  999432  999432  999432  999433  999434  999434  999436  999436  999437  999439  999440  999440  999441  999441  999442  999442  999442  999443  999445  999446  999446  999446  999447  999447  999
449     999452  999452  999454  999454  999454  999455  999456  999459  999460  999460  999461  999465  999465  999466  999466  999466  999466  999468  999468  999469  999469  999469  999470  999470  999471  999
471     999473  999474  999476  999479  999479  999479  999479  999479  999482  999484  999484  999484  999484  999485  999485  999486  999488  999489  999490  999492  999492  999492  999494  999495  999496  999
496     999498  999498  999498  999499  999501  999502  999502  999504  999504  999504  999505  999507  999509  999509  999510  999510  999511  999512  999515  999520  999521  999523  999523  999523  999523  999
523     999523  999526  999526  999527  999528  999529  999529  999530  999531  999532  999533  999533  999534  999535  999536  999536  999537  999538  999539  999539  999541  999541  999541  999543  999544  999
544     999544  999544  999546  999546  999550  999550  999552  999553  999556  999557  999558  999558  999559  999560  999564  999565  999566  999566  999570  999570  999571  999572  999573  999574  999576  999
578     999580  999580  999581  999583  999584  999584  999584  999584  999584  999585  999586  999588  999592  999594  999596  999597  999597  999598  999599  999602  999604  999604  999604  999606  999606  999
606     999607  999608  999608  999609  999609  999610  999610  999611  999612  999612  999613  999614  999615  999617  999620  999623  999623  999624  999624  999625  999627  999628  999629  999629  999631  999
631     999631  999631  999632  999632  999632  999633  999635  999635  999636  999638  999639  999640  999641  999642  999643  999644  999644  999644  999645  999649  999650  999650  999651  999651  999652  999
654     999654  999659  999660  999661  999661  999662  999662  999663  999664  999667  999667  999669  999670  999673  999674  999674  999674  999674  999677  999678  999678  999679  999679  999680  999680  999
681     999687  999688  999689  999691  999691  999695  999696  999696  999696  999696  999697  999698  999699  999703  999705  999707  999707  999708  999709  999710  999712  999712  999712  999714  999715  999
715     999715  999717  999718  999719  999720  999722  999726  999727  999728  999729  999730  999730  999730  999730  999730  999731  999733  999734  999736  999738  999740  999742  999742  999743  999744  999745  999
745     999745  999746  999747  999748  999748  999748  999749  999750  999750  999750  999752  999755  999757  999758  999761  999762  999766  999767  999767  999768  999771  999771  999773  999773  999774  999
774     999775  999775  999775  999775  999775  999776  999776  999777  999779  999779  999780  999781  999783  999784  999784  999784  999789  999789  999789  999790  999791  999792  999792  999793  999794  999
795     999796  999798  999798  999799  999799  999799  999800  999800  999801  999802  999805  999805  999806  999806  999807  999807  999808  999812  999812  999813  999813  999814  999819  999820  999820  999
821     999822  999822  999823  999824  999827  999828  999828  999829  999831  999832  999834  999835  999835  999837  999838  999839  999839  999839  999839  999839  999840  999840  999841  999841  999841  999842  999
845     999849  999850  999851  999851  999853  999855  999857  999858  999858  999859  999859  999860  999860  999861  999862  999863  999864  999865  999865  999866  999867  999868  999868  999869  999870  999
870     999870  999872  999872  999872  999873  999873  999874  999874  999876  999877  999878  999879  999880  999882  999883  999884  999886  999887  999888  999888  999888  999889  999889  999890  999890  999
893     999894  999894  999894  999899  999899  999899  999899  999900  999900  999900  999902  999903  999903  999904  999904  999906  999906  999907  999908  999908  999909  999909  999909  999910  999912  999
912     999912  999913  999913  999913  999914  999914  999915  999915  999915  999918  999920  999920  999920  999921  999921  999922  999922  999922  999923  999924  999924  999924  999925  999926  999927  999
928     999929  999932  999932  999932  999933  999934  999934  999934  999937  999939  999939  999940  999941  999942  999944  999945  999948  999949  999949  999950  999950  999952  999953  999953  999953  999
953     999953  999953  999954  999955  999955  999956  999957  999959  999960  999961  999961  999963  999964  999965  999965  999965  999967  999969  999969  999971  999972  999973  999973  999974  999974  999
974     999976  999979  999979  999980  999983  999983  999984  999984  999985  999986  999986  999986  999989  999990  999990  999991  999991  999992  999992  999992  999993  999995  999995  999996  999996  999
997     999997  999997  999998  999998
 1000000 random numbers sorted by using quick sorting algoritms.


Time taken for quick sort in seconds: 0.390000
```

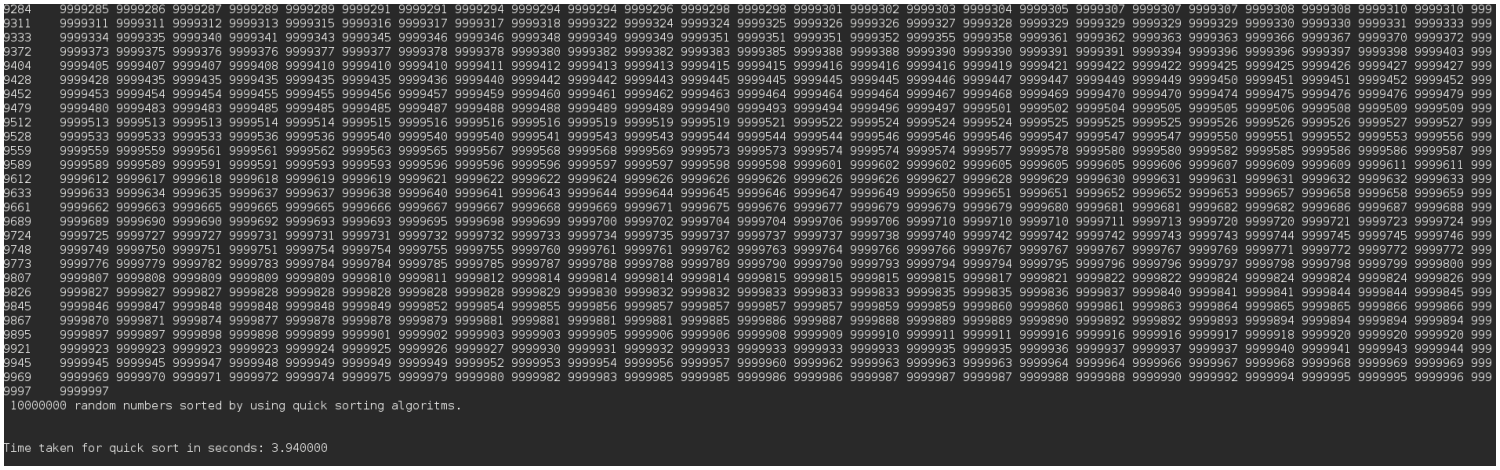
Quick Sort 5,000,000 Words:

After execution the quick sort algorithm displays that the time taken to process 1,000,000 random numbers was 1.98000 seconds.



Quick Sort 10,000,000 Words:

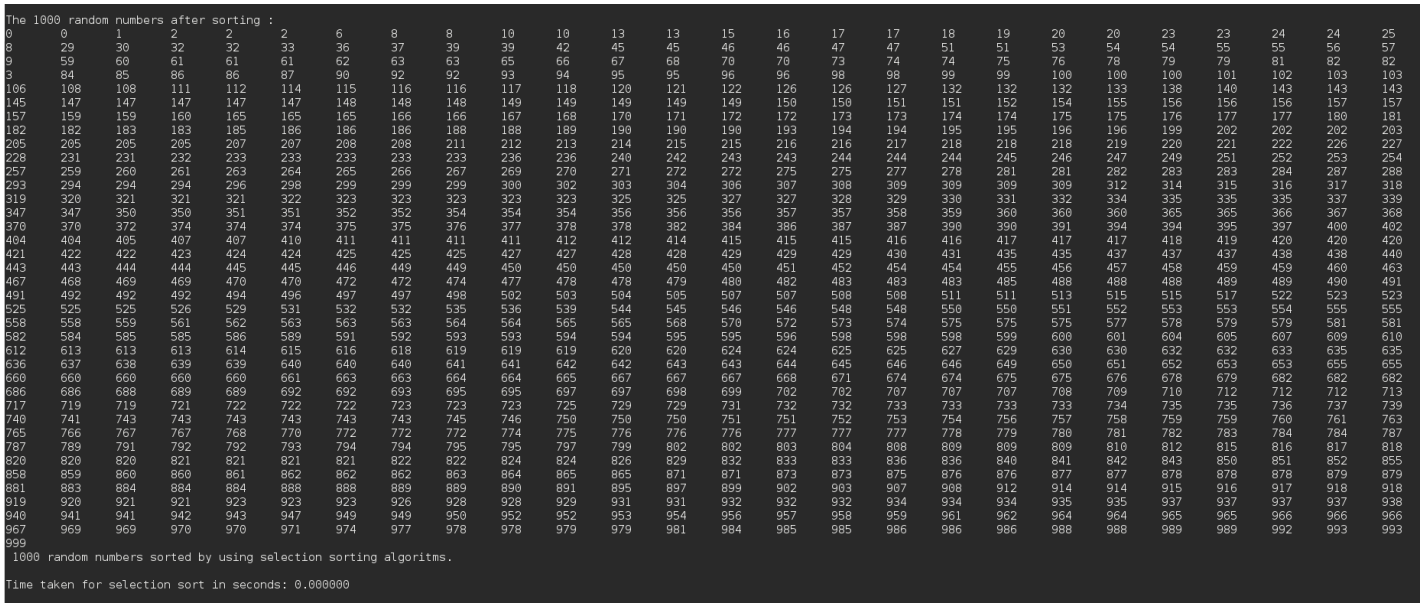
After execution the quick sort algorithm displays that the time taken to process 1,000,000 random numbers was 3.94000 seconds.



Running through the test phase of the program with Selection Sort, the results are shown below:

Selection Sort 1000 Words:

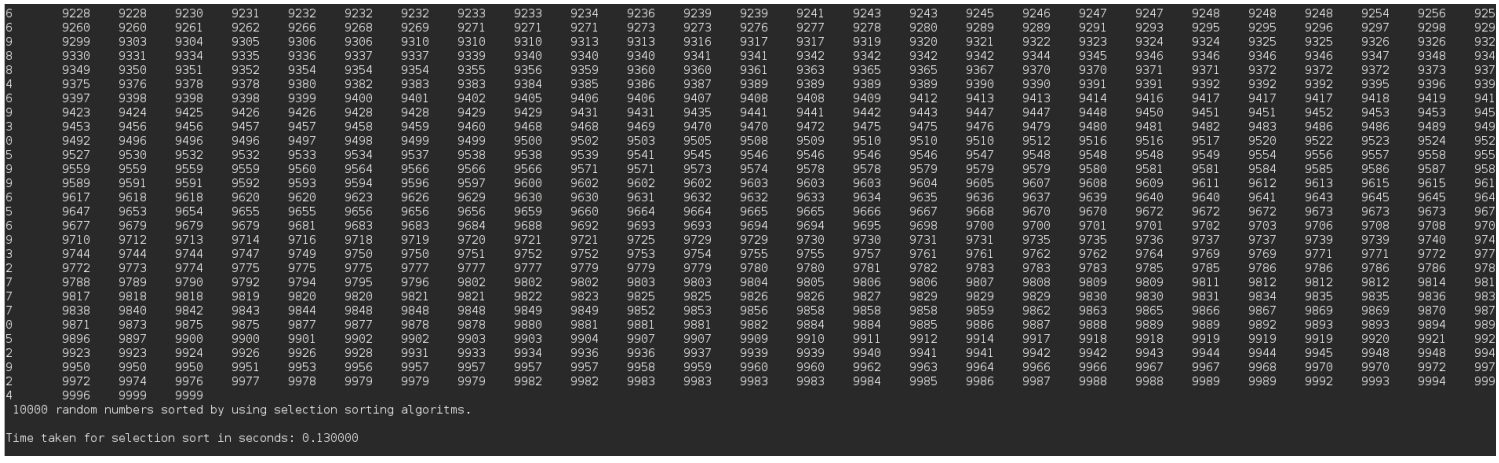
After execution the selection sort algorithm displays that the time taken to process 1000 random numbers was 0.000 seconds; this is dependant on the computer and their processors



```
The 1000 random numbers after sorting :
0       0       1       2       2       2       6       8       8       10      10      13      13      15      16      17      17      18      19      20      20      23      23      24      24      25
8       29      30      32      32      33      36      37      39      39      42      45      45      46      46      47      47      51      51      53      54      54      55      55      56      57
9       59      60      61      61      61      62      63      63      65      66      67      68      70      70      73      74      74      75      76      78      79      79      81      82      82
3       84      85      86      86      87      90      92      92      93      94      95      95      96      96      98      98      99      99      100     100     100     101     102     103     103
106     108     108     111     112     114     115     116     116     117     118     120     121     122     126     126     127     132     132     132     133     138     140     143     143     143
145     147     147     147     147     147     148     148     148     149     149     149     149     149     150     150     151     151     152     154     155     156     156     156     157     157
157     159     159     160     165     165     165     166     166     167     168     170     171     172     172     173     173     174     174     175     175     176     177     177     180     181
182     182     183     183     185     186     186     186     188     188     189     190     190     190     193     194     194     195     195     196     196     199     202     202     202     203
205     205     205     205     207     207     208     208     211     212     213     214     215     215     216     216     217     218     218     218     219     220     221     222     226     227
228     231     231     232     233     233     233     233     233     236     236     240     242     243     243     244     244     244     245     246     247     249     251     252     253     254
257     259     260     261     263     264     265     266     267     269     270     271     272     272     275     275     277     278     281     281     282     283     283     284     287     288
293     294     294     294     296     298     299     299     299     300     302     303     304     306     307     308     309     309     309     309     312     314     315     316     317     318
319     320     321     321     321     322     323     323     323     323     323     325     325     327     327     328     329     330     331     332     334     335     335     335     337     339
347     347     350     350     351     351     352     352     354     354     354     356     356     356     357     357     358     359     360     360     360     365     365     366     367     368
370     370     372     374     374     374     375     375     376     377     378     378     382     384     386     387     387     390     390     391     394     394     395     397     400     402
404     404     405     407     407     410     411     411     411     411     412     412     414     415     415     415     416     416     417     417     417     418     419     420     420     420
421     422     422     423     424     424     425     425     425     427     427     428     428     429     429     429     430     431     435     435     437     437     437     438     438     440
443     443     444     444     445     445     446     449     449     450     450     450     450     450     451     452     454     454     455     456     457     458     459     459     460     463
467     468     469     469     470     470     472     472     474     477     478     478     479     480     482     483     483     483     485     488     488     488     489     489     490     491
491     492     492     492     494     496     497     497     498     502     503     504     505     507     507     508     508     511     511     513     515     515     517     522     523     523
525     525     525     526     529     531     532     532     535     536     539     544     545     546     546     548     548     550     550     551     552     553     553     554     555     555
558     558     559     561     562     563     563     563     564     564     565     565     568     570     572     573     574     575     575     575     577     578     579     579     581     581
582     584     585     585     586     589     591     592     593     593     594     594     595     595     596     598     598     598     599     600     601     604     605     607     609     610
612     613     613     613     614     615     616     618     619     619     619     620     620     624     624     625     625     627     629     630     630     632     632     633     635     635
636     637     638     639     639     640     640     640     641     641     642     642     643     643     644     645     646     646     649     650     651     652     653     653     655     655
660     660     660     660     660     661     663     663     664     664     665     667     667     667     668     671     674     674     675     675     676     678     679     682     682     682
686     686     688     689     689     692     692     693     695     695     697     697     698     699     702     702     707     707     707     708     709     710     712     712     712     713
717     719     719     721     722     722     722     723     723     723     725     729     729     731     732     732     733     733     733     733     734     735     735     736     737     739
740     741     743     743     743     743     743     743     745     746     750     750     750     751     751     752     753     754     756     757     758     759     759     760     761     763
765     766     767     767     768     770     772     772     772     774     775     776     776     776     777     777     777     778     779     780     781     782     783     784     784     787
787     789     791     792     792     793     794     794     795     795     797     799     802     802     803     804     808     809     809     809     810     812     815     816     817     818
820     820     820     821     821     821     821     822     822     824     824     826     829     832     833     833     836     836     840     841     842     843     850     851     852     855
858     859     860     860     861     862     862     862     863     864     865     865     871     871     873     873     875     876     876     877     877     878     878     878     879     879
881     883     884     884     884     888     888     889     889     890     891     895     897     899     902     903     907     908     912     914     914     915     916     917     918     918
919     920     921     921     923     923     923     926     928     928     929     931     931     932     932     932     934     934     934     935     935     937     937     937     937     938
940     941     941     942     943     947     949     949     950     952     952     953     954     956     957     958     959     961     962     964     964     965     965     966     966     966
967     969     969     970     970     971     974     977     978     978     979     979     981     984     985     985     986     986     986     988     988     989     989     992     993     993
999
 1000 random numbers sorted by using selection sorting algoritms.

Time taken for selection sort in seconds: 0.000000
```

Selection Sort 10,000 Words:

After execution the selection sort algorithm displays that the time taken to process 10,000 random numbers was 0.13000 seconds.



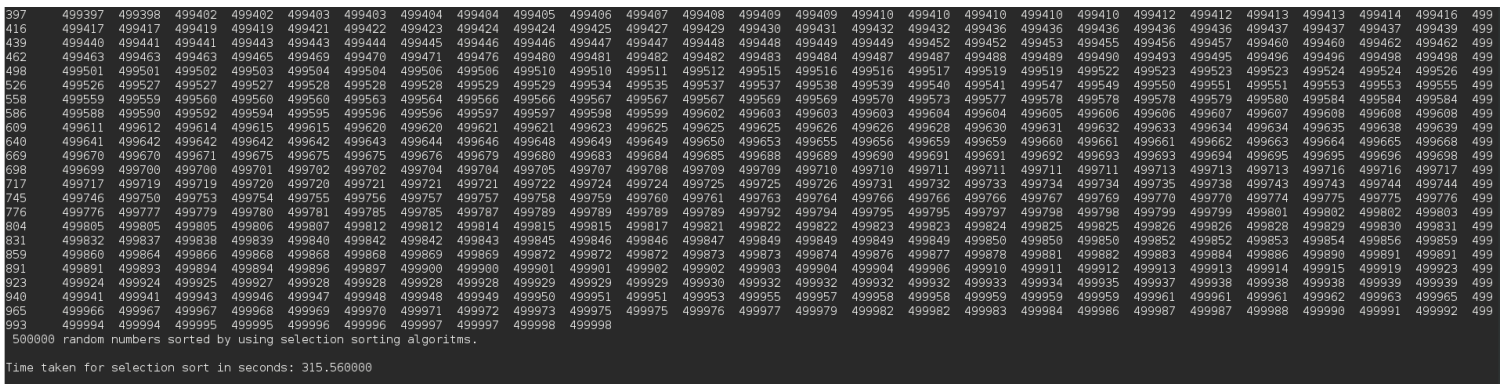
```
6       9228    9228    9230    9231    9232    9232    9232    9233    9233    9234    9236    9239    9239    9241    9243    9243    9245    9246    9247    9247    9248    9248    9248    9254    9256    925
6       9260    9260    9261    9262    9266    9268    9269    9271    9271    9271    9273    9273    9276    9277    9278    9280    9289    9289    9291    9293    9295    9295    9296    9297    9298    929
9       9299    9303    9304    9305    9306    9306    9310    9310    9310    9313    9313    9316    9317    9317    9319    9320    9321    9322    9323    9324    9324    9325    9325    9326    9326    932
8       9330    9331    9334    9335    9336    9337    9337    9339    9340    9340    9340    9341    9341    9342    9342    9342    9342    9344    9345    9346    9346    9346    9346    9347    9348    934
8       9349    9350    9351    9352    9354    9354    9354    9355    9356    9359    9360    9360    9361    9363    9365    9365    9367    9370    9370    9371    9371    9372    9372    9372    9373    937
4       9375    9376    9378    9378    9380    9382    9383    9383    9384    9385    9386    9387    9389    9389    9389    9389    9390    9390    9391    9391    9392    9392    9392    9395    9396    939
6       9397    9398    9398    9398    9399    9400    9401    9402    9405    9406    9406    9407    9408    9408    9409    9412    9413    9413    9414    9416    9417    9417    9417    9418    9419    941
9       9423    9424    9425    9426    9426    9428    9428    9429    9429    9431    9431    9435    9441    9441    9442    9443    9447    9447    9448    9450    9451    9451    9452    9453    9453    945
3       9453    9456    9456    9457    9457    9458    9459    9460    9468    9468    9469    9470    9470    9472    9475    9475    9476    9479    9480    9481    9482    9483    9486    9486    9489    949
0       9492    9496    9496    9496    9497    9498    9499    9499    9500    9502    9503    9505    9508    9509    9510    9510    9510    9512    9516    9516    9517    9520    9522    9523    9524    952
5       9527    9530    9532    9532    9533    9534    9537    9538    9538    9539    9541    9545    9546    9546    9546    9546    9547    9548    9548    9548    9549    9554    9556    9557    9558    955
9       9559    9559    9559    9559    9560    9564    9566    9566    9566    9571    9571    9573    9574    9578    9578    9579    9579    9579    9580    9581    9581    9584    9585    9586    9587    958
9       9589    9591    9591    9592    9593    9594    9596    9597    9600    9602    9602    9602    9603    9603    9603    9604    9605    9607    9608    9609    9611    9612    9613    9615    9615    961
6       9617    9618    9618    9620    9620    9623    9626    9629    9630    9630    9631    9632    9632    9633    9634    9635    9636    9637    9639    9640    9640    9641    9643    9645    9645    964
5       9647    9653    9654    9655    9655    9656    9656    9656    9659    9660    9664    9664    9665    9665    9666    9667    9668    9670    9670    9672    9672    9672    9673    9673    9673    967
6       9677    9679    9679    9679    9681    9683    9683    9684    9688    9692    9693    9693    9694    9694    9695    9698    9700    9700    9701    9701    9702    9703    9706    9708    9708    970
9       9710    9712    9713    9714    9716    9718    9719    9720    9721    9721    9725    9729    9729    9730    9730    9731    9731    9735    9735    9736    9737    9737    9739    9739    9740    974
3       9744    9744    9744    9747    9749    9750    9750    9751    9752    9752    9753    9754    9755    9755    9757    9761    9761    9762    9762    9764    9769    9769    9771    9771    9772    977
2       9772    9773    9774    9775    9775    9775    9777    9777    9777    9779    9779    9779    9780    9780    9781    9782    9783    9783    9783    9785    9785    9786    9786    9786    9786    978
7       9788    9789    9790    9792    9794    9795    9796    9802    9802    9802    9803    9803    9804    9805    9806    9806    9807    9808    9809    9809    9811    9812    9812    9812    9814    981
7       9817    9818    9818    9819    9820    9820    9821    9821    9822    9823    9825    9825    9826    9826    9827    9829    9829    9829    9830    9830    9831    9834    9835    9835    9836    983
7       9838    9840    9842    9843    9844    9848    9848    9848    9849    9849    9852    9853    9856    9858    9858    9858    9859    9862    9863    9865    9866    9867    9869    9869    9870    987
0       9871    9873    9875    9875    9877    9877    9878    9878    9880    9881    9881    9881    9882    9884    9884    9885    9886    9887    9888    9889    9889    9892    9893    9893    9894    989
5       9896    9897    9900    9900    9901    9902    9902    9903    9903    9904    9907    9907    9909    9910    9911    9912    9914    9917    9918    9918    9919    9919    9919    9920    9921    992
2       9923    9923    9924    9926    9926    9928    9931    9933    9934    9936    9936    9937    9939    9939    9940    9941    9941    9942    9942    9943    9944    9944    9945    9948    9948    994
9       9950    9950    9950    9951    9953    9956    9957    9957    9957    9957    9958    9959    9960    9960    9962    9963    9964    9966    9966    9967    9967    9968    9970    9970    9972    997
2       9972    9974    9976    9977    9978    9979    9979    9979    9982    9982    9983    9983    9983    9983    9984    9985    9986    9987    9988    9988    9989    9989    9992    9993    9994    999
4       9996    9999    9999
 10000 random numbers sorted by using selection sorting algoritms.

Time taken for selection sort in seconds: 0.130000
```

Selection Sort 100,000 Words:

After execution the selection sort algorithm displays that the time taken to process 100,000 random numbers was 12.5600 seconds.

Selection Sort 500,000 Words:

After execution the selection sort algorithm displays that the time taken to process 500,000 random numbers was 315.5600 seconds.



Running through the test phase of the program with Merge Sort, the results are shown below:

Merge Sort 1000 Words:

After execution the merge sort algorithm displays that the time taken to process 1000 random numbers was 0.000 second; this is due to the algorithm for merge sort is too quick, this is also dependant on the computer.

```
409     409     409     410     410     410     411     413     416     417     418     419     422     422     425     426     426     427     427     427     427     429     431     431     431     432     433
434     436     438     439     439     442     444     445     445     448     449     454     455     456     457     457     458     459     459     459     459     460     460     460     461     464     464
465     465     465     468     468     468     470     474     476     476     477     479     480     480     480     480     480     481     482     483     485     486     487     487     488     491     492
493     494     496     498     498     498     499     500     501     501     501     504     504     505     505     509     509     512     515     516     516     516     518     519     519     520     520
521     522     523     526     526     527     528     530     530     530     530     532     532     533     533     534     534     534     535     535     535     535     538     538     539     539     540
542     545     545     546     547     548     552     552     552     553     553     554     554     554     555     556     557     557     558     558     558     558     560     560     562     563     564
564     566     570     570     570     570     571     571     571     572     572     573     580     581     581     583     583     585     585     586     586     587     589     591     591     594     595
597     597     599     599     600     601     602     603     604     604     604     604     606     606     606     607     607     608     609     610     610     611     611     611     611     611     614
616     616     618     619     619     620     620     621     621     622     623     625     626     626     626     629     631     634     634     634     635     636     637     640     640     641     641     643
644     646     646     646     647     648     648     649     649     650     651     652     653     654     655     658     659     661     662     663     667     671     671     672     672     674     674
675     676     676     682     684     686     686     686     686     688     689     689     692     692     693     695     698     699     702     703     704     705     707     708     709     710     710
711     711     712     712     713     713     713     715     716     718     718     719     720     721     722     724     725     726     729     729     734     735     736     737     737     738     739
739     742     743     744     744     745     746     746     746     747     747     750     750     753     753     754     754     754     755     756     758     759     762     762     764     765     765
767     769     771     773     774     774     776     777     777     777     779     779     779     780     780     781     783     784     784     785     785     786     788     791     791     794     795
799     801     803     804     804     805     806     806     807     808     811     811     812     813     813     814     815     815     817     817     817     818     818     818     819     819     819
820     820     821     822     822     822     823     823     824     824     824     825     826     826     826     828     829     831     831     831     832     834     835     836     837     837     841
842     843     844     846     846     847     847     848     848     851     852     853     855     856     856     859     861     863     863     866     867     867     868     868     868     870     870
870     872     873     875     876     877     878     879     879     881     882     884     885     885     885     886     886     888     890     891     891     891     892     893     894     894     895
897     897     898     899     900     901     901     901     902     902     903     904     904     904     905     905     906     908     909     911     911     911     911     913     913     914     914
915     915     915     916     917     919     920     921     923     923     924     924     924     925     928     929     931     931     931     932     932     933     935     935     936     936     936
937     938     938     942     944     944     945     946     946     947     947     949     950     950     951     952     952     954     954     954     955     955     955     956     959     959     960
962     963     963     966     968     968     968     969     971     974     974     978     979     981     982     985     991     992     992     994     994     994     994     994     995     996     996
997
 1000 random numbers sorted by using merge sorting algoritms.


Time taken for merge sort in seconds: 0.000000
```

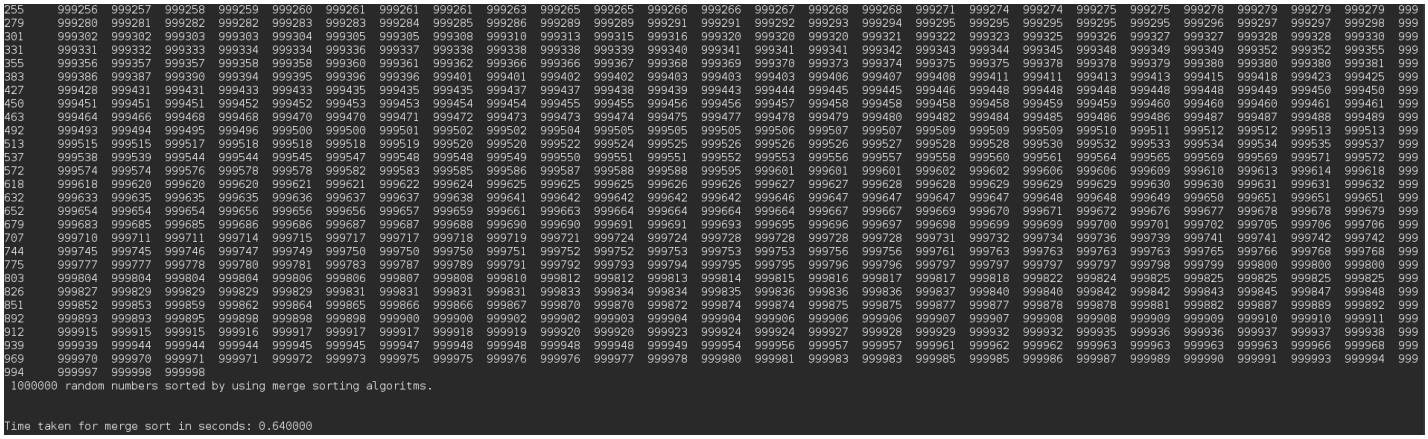
Merge Sort 100,000 Words:

After execution the merge sort algorithm displays that the time taken to process 100,000 random numbers was 0.0600 seconds.

```
46      99346   99347   99348   99348   99348   99350   99351   99351   99352   99352   99353   99354   99356   99358   99358   99359   99359   99360   99360   99362   99362   99363   99366   99366   99367   993
67      99368   99368   99369   99369   99370   99370   99371   99373   99374   99374   99376   99377   99378   99378   99379   99380   99380   99380   99384   99384   99385   99386   99387   99388   99388   993
89      99390   99390   99394   99394   99395   99396   99397   99398   99398   99398   99401   99401   99401   99402   99404   99404   99405   99408   99410   99411   99412   99412   99413   99414   99414   994
14      99415   99415   99415   99415   99416   99417   99417   99420   99420   99422   99423   99423   99423   99424   99424   99425   99425   99425   99427   99430   99430   99431   99431   99433   99435   994
37      99437   99440   99440   99442   99448   99448   99448   99449   99450   99451   99452   99452   99453   99454   99455   99455   99455   99456   99456   99458   99458   99459   99459   99460   99461   994
61      99461   99462   99464   99464   99465   99467   99468   99468   99468   99471   99471   99476   99477   99477   99479   99479   99482   99482   99482   99483   99484   99484   99487   99488   99488   994
88      99493   99493   99494   99495   99495   99496   99496   99497   99498   99498   99498   99498   99501   99501   99502   99503   99503   99506   99507   99508   99511   99511   99512   99513   99514   995
15      99517   99517   99519   99519   99520   99522   99523   99524   99524   99525   99525   99525   99526   99526   99526   99528   99529   99531   99531   99532   99533   99533   99535   99536   99537   995
38      99538   99540   99541   99544   99546   99549   99550   99551   99552   99554   99557   99558   99558   99560   99561   99562   99563   99564   99564   99566   99566   99568   99568   99569   99571   995
71      99571   99572   99572   99573   99573   99574   99575   99576   99576   99577   99578   99578   99579   99580   99580   99581   99583   99584   99585   99586   99587   99588   99588   99588   99589   995
90      99590   99592   99593   99593   99595   99596   99596   99597   99599   99601   99601   99603   99606   99607   99608   99610   99611   99612   99612   99612   99612   99615   99618   99618   99619   996
19      99622   99624   99624   99624   99626   99626   99628   99629   99629   99631   99632   99632   99636   99637   99639   99640   99643   99643   99644   99646   99646   99648   99649   99652   99652   996
54      99656   99663   99664   99664   99664   99668   99670   99670   99670   99671   99671   99672   99672   99673   99674   99674   99676   99677   99678   99678   99678   99679   99679   99680   99683   996
83      99684   99685   99685   99686   99688   99688   99689   99689   99689   99689   99691   99693   99698   99698   99700   99700   99702   99703   99704   99706   99706   99709   99709   99710   99711   997
12      99713   99715   99717   99717   99717   99718   99718   99721   99721   99722   99723   99723   99723   99725   99725   99726   99727   99730   99731   99731   99731   99732   99733   99734   99735   997
36      99739   99741   99744   99744   99746   99747   99747   99751   99752   99752   99752   99753   99754   99754   99754   99755   99757   99757   99757   99759   99759   99760   99763   99763   99764   997
66      99766   99770   99770   99771   99774   99774   99775   99779   99779   99780   99780   99781   99781   99785   99785   99786   99786   99786   99787   99789   99790   99790   99790   99791   99791   997
93      99794   99794   99795   99795   99795   99796   99797   99797   99799   99800   99800   99801   99802   99802   99803   99803   99805   99806   99807   99807   99808   99808   99809   99810   99810   998
11      99812   99812   99812   99814   99814   99815   99816   99817   99817   99818   99819   99820   99821   99822   99824   99825   99827   99829   99829   99833   99835   99835   99837   99839   99839   998
39      99841   99842   99843   99843   99843   99844   99844   99846   99847   99848   99850   99853   99855   99857   99859   99860   99861   99861   99861   99862   99863   99865   99867   99868   99869   998
71      99872   99874   99874   99874   99875   99876   99876   99877   99877   99878   99878   99880   99880   99882   99883   99883   99883   99884   99884   99885   99885   99886   99886   99887   99887   998
89      99890   99894   99895   99896   99896   99898   99899   99901   99901   99902   99903   99903   99903   99904   99906   99907   99910   99910   99911   99914   99914   99915   99915   99917   99917   999
17      99918   99918   99919   99919   99920   99922   99922   99923   99924   99925   99925   99925   99926   99928   99929   99930   99930   99931   99933   99935   99935   99936   99936   99937   99940   999
40      99942   99943   99944   99945   99945   99945   99945   99946   99946   99946   99946   99948   99949   99951   99952   99952   99954   99954   99955   99958   99958   99958   99958   99959   99959   999
60      99960   99960   99960   99962   99963   99964   99965   99966   99967   99968   99969   99969   99970   99971   99974   99974   99974   99978   99978   99978   99979   99980   99980   99982   99982   999
82      99985   99985   99986   99986   99987   99988   99992   99992   99992   99992   99992   99993   99993   99994   99995   99996   99997
 100000 random numbers sorted by using merge sorting algoritms.


Time taken for merge sort in seconds: 0.060000
```

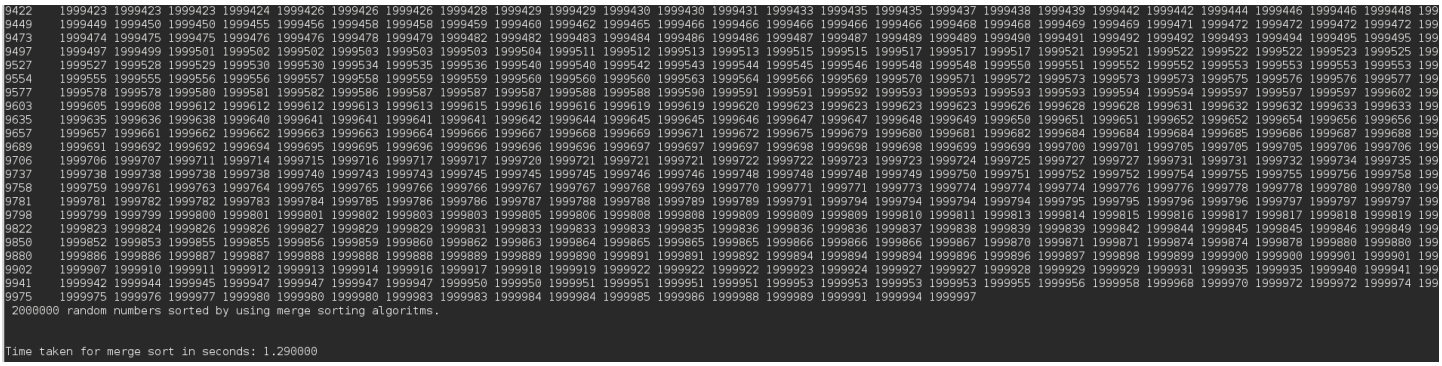
Merge Sort 1000000 Words:

After execution the merge sort algorithm displays that the time taken to process 1,000,000 random numbers was 0.64000 seconds.



Merge Sort 2000000 Words:

After execution the merge sort algorithm displays that the time taken to process 2,000,000 random numbers was 1.29000 seconds.



The Testing phases on the Gnuplot were plotted as such:

GNUPLOT TEST

Run the program for the 10000 and 50000 random numbers

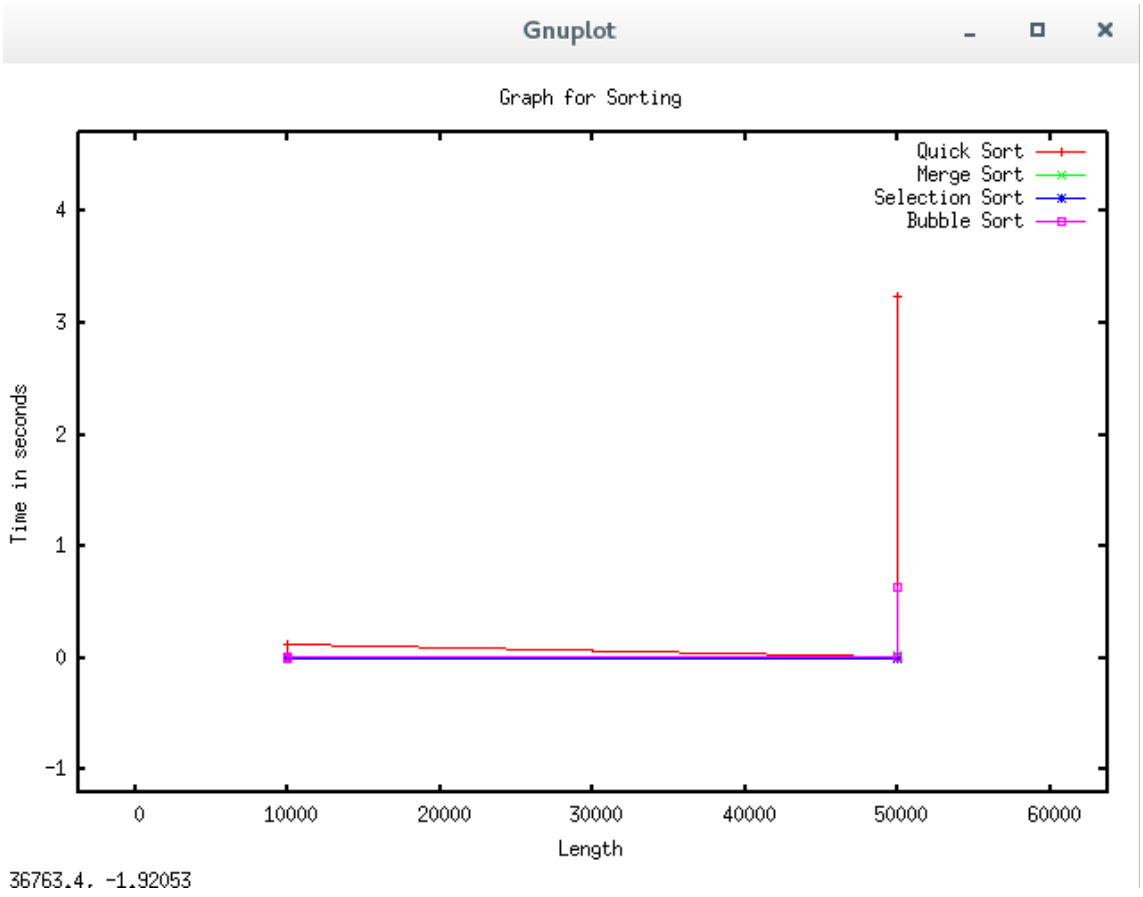
In graph, x axis is Length of the numbers and y axis is Time in seconds.

First I run the program for 10000 unsorted numbers and sorted by all sorting types i.e quick, merge, selection and bubble sorting

Time taken by quick sorting – 0.0000 seconds
Time taken by merge sorting – 0.0100 seconds
Time taken by selection sorting – 0.1200 seconds
Time taken by bubble sorting – 0.0300 seconds

And then for 50000 unsorted numbers.

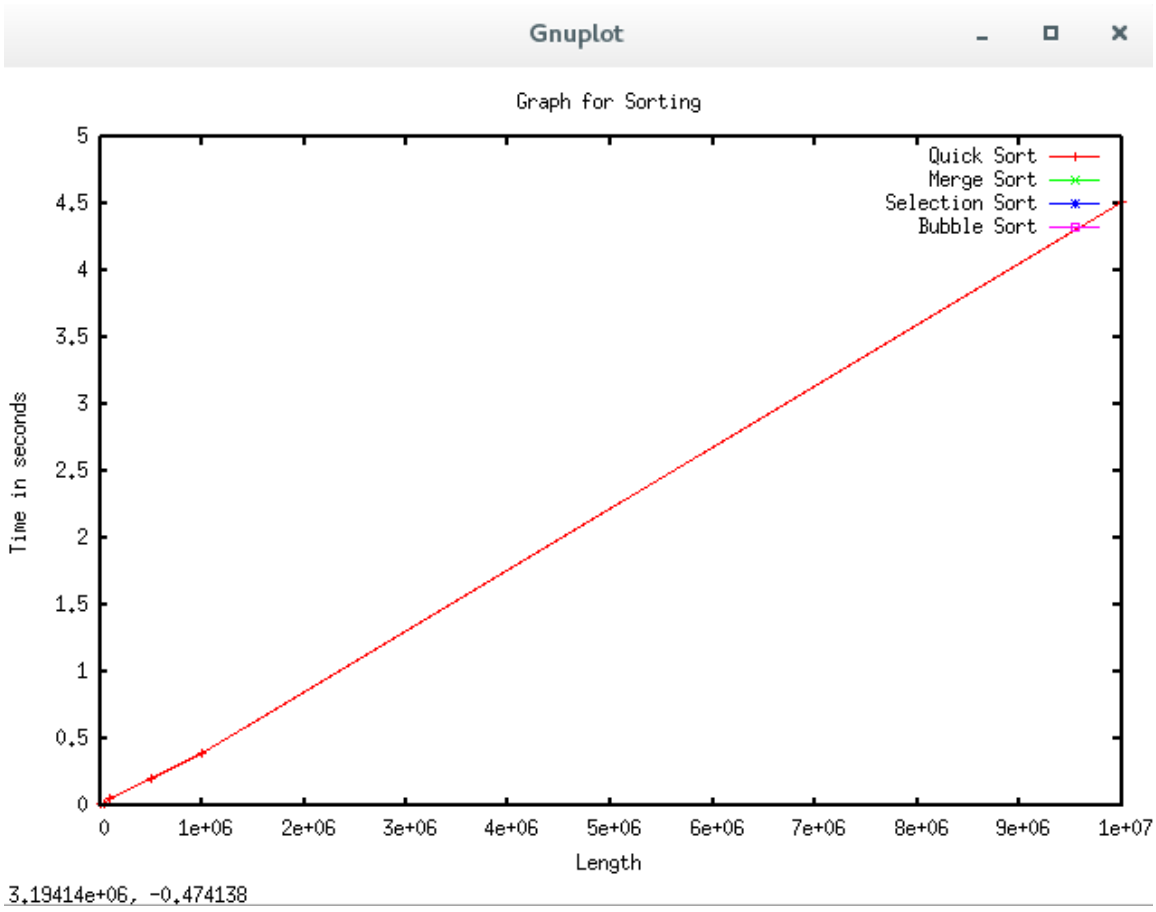
Time taken by quick sorting – 0.0200 seconds
Time taken by merge sorting – 0.0300 seconds
Time taken by selection sorting – 3.1600 seconds
Time taken by bubble sorting – 0.6200 seconds



Run the program for the 20000, 50000,100000,500000,1000000,10000000 random numbers

In graph, x axis is Length of the numbers and y axis is Time in seconds.

Time taken by quick sorting to sort 20000 numbers – 0.0100 seconds
Time taken by quick sorting to sort 50000 numbers – 0.0100 seconds
Time taken by quick sorting to sort 100000 numbers – 0.0500 seconds
Time taken by quick sorting to sort 500000 numbers – 0.2200 seconds
Time taken by quick sorting to sort 1000000 numbers – 0.3900 seconds
Time taken by quick sorting to sort 10000000 numbers – 3.9900 seconds

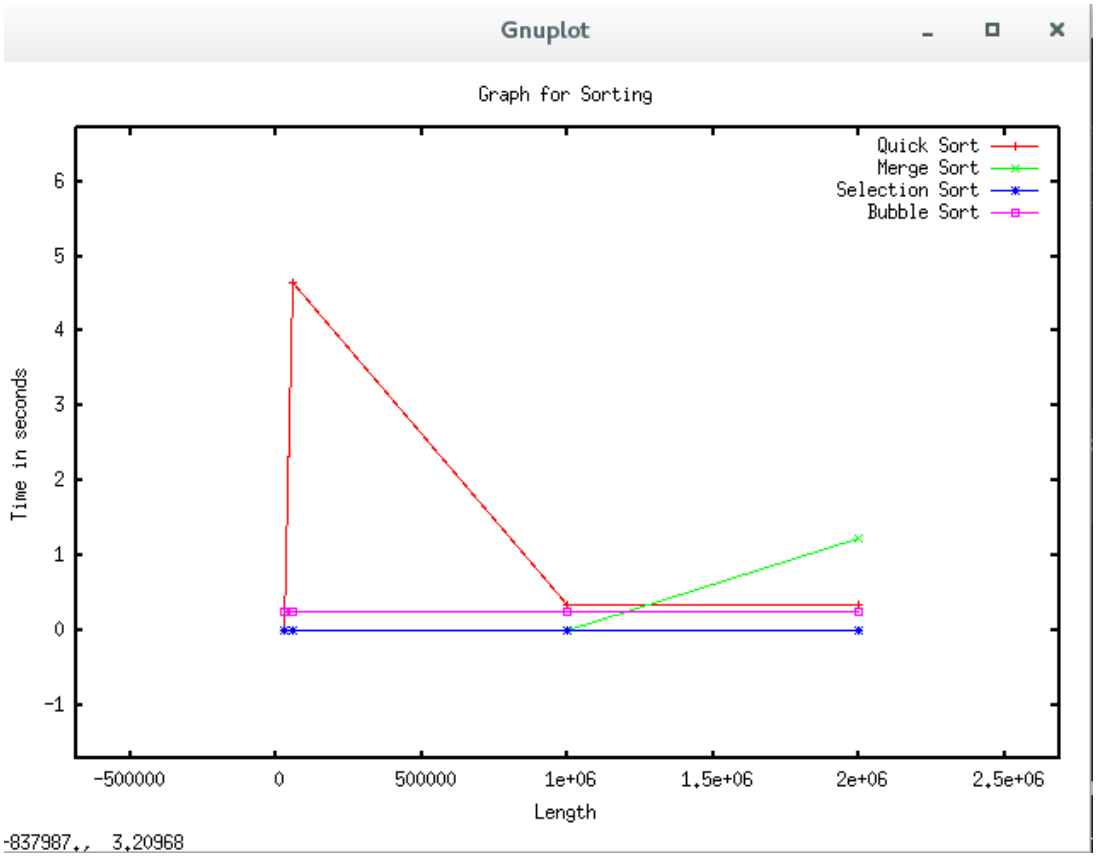


graph.out

```
10000 0.010000 0.000000 0.000000 0.000000
50000 0.020000 0.000000 0.000000 0.000000
100000 0.050000 0.000000 0.000000 0.000000
500000 0.200000 0.000000 0.000000 0.000000
1000000 0.390000 0.000000 0.000000 0.000000
10000000 4.520000 0.000000 0.000000 0.000000
```

Run the program for the 20000, 50000,1000000 and 2000000 random numbers
In graph, x axis is Length of the numbers and y axis is Time in seconds.

Time taken by bubble sorting to sort 30000 numbers – 0.24000 seconds
Time taken by selection sorting to sort 50000 numbers – 4.64000 seconds
Time taken by quick sorting to sort 1000000 numbers – 0.33000 seconds
Time taken by merge sorting to sort 2000000 numbers – 1.23000 seconds



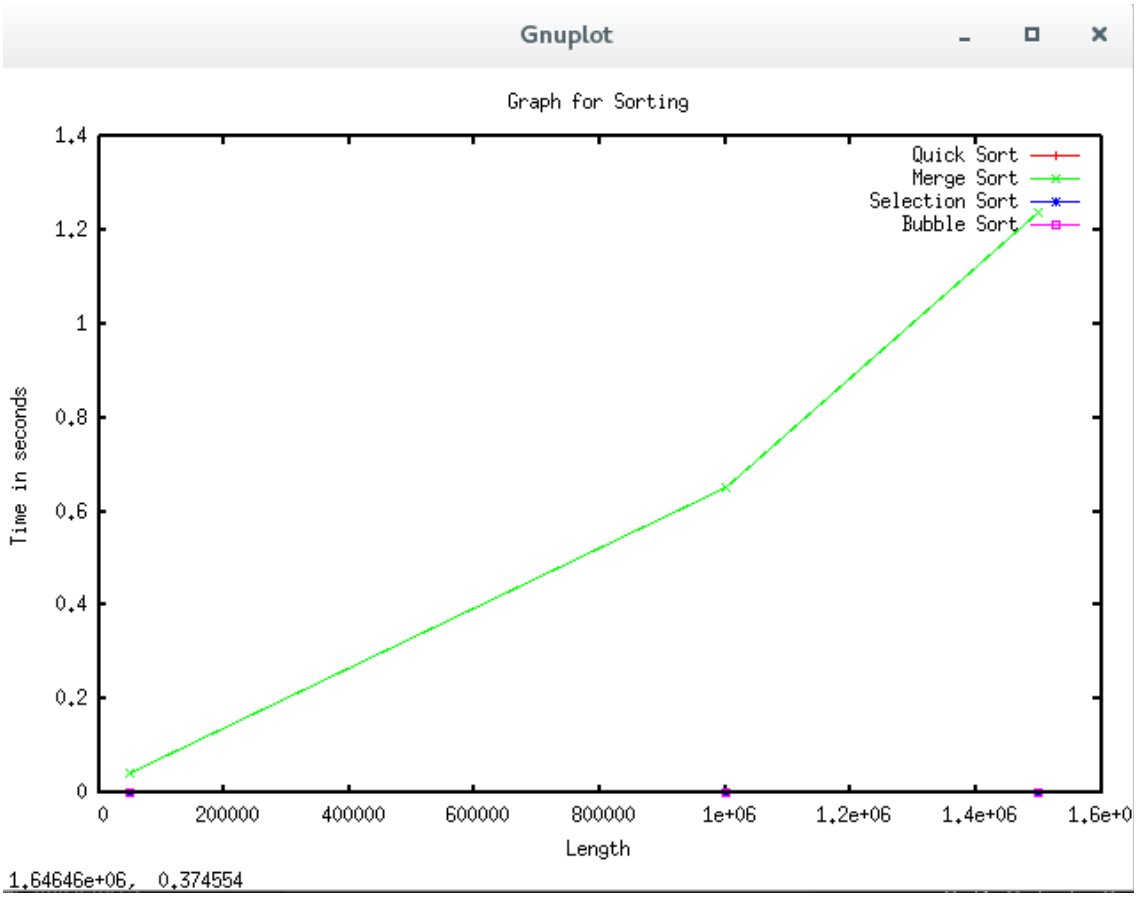
This is output file of gnuplot i.e **graph.out**. As displayed in the file below, the figures show that when when I ran the program to sort out the 30000 by bubble sorting method, it also included 0.0000 second for the quick, merge and selection for 30000 numbers. Same for all other numbers.

```
1  30000 0.000000 0.000000 0.000000 0.240000
2  60000 4.640000 0.000000 0.000000 0.240000
3  1000000 0.330000 0.000000 0.000000 0.240000
4  2000000 0.330000 1.230000 0.000000 0.240000
5
```

Run the program for the 50000, 1000000 and 1500000 random numbers

In graph, x axis is Length of the numbers and y axis is Time in seconds.

Time taken by merge sorting to sort 50000 numbers – 0.04000 seconds
Time taken by merge sorting to sort 1000000 numbers – 0.65000 seconds
Time taken by merge sorting to sort 1500000 numbers – 1.24000 seconds



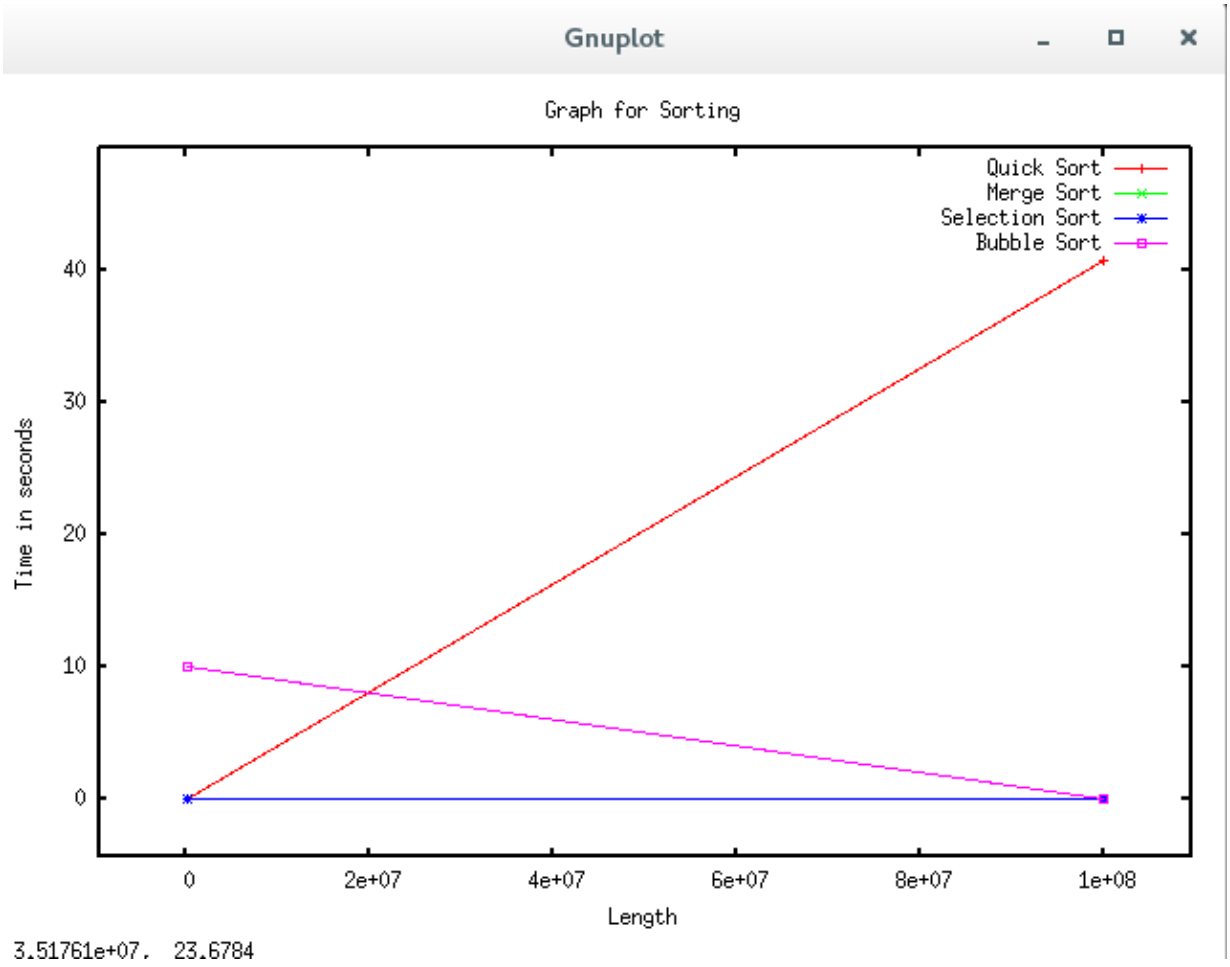
graph.out

```
50000 0.000000 0.040000 0.000000 0.000000
1000000 0.000000 0.650000 0.000000 0.000000
1500000 0.000000 1.240000 0.000000 0.000000
```

Run the program for the 100000000 random numbers

In graph, x axis is Length of the numbers and y axis is Time in seconds.

Time taken by quick sorting to sort 100000000 numbers – 40.77000 seconds
Time taken by bubble sorting to sort 1000000 numbers – 10.12000 seconds



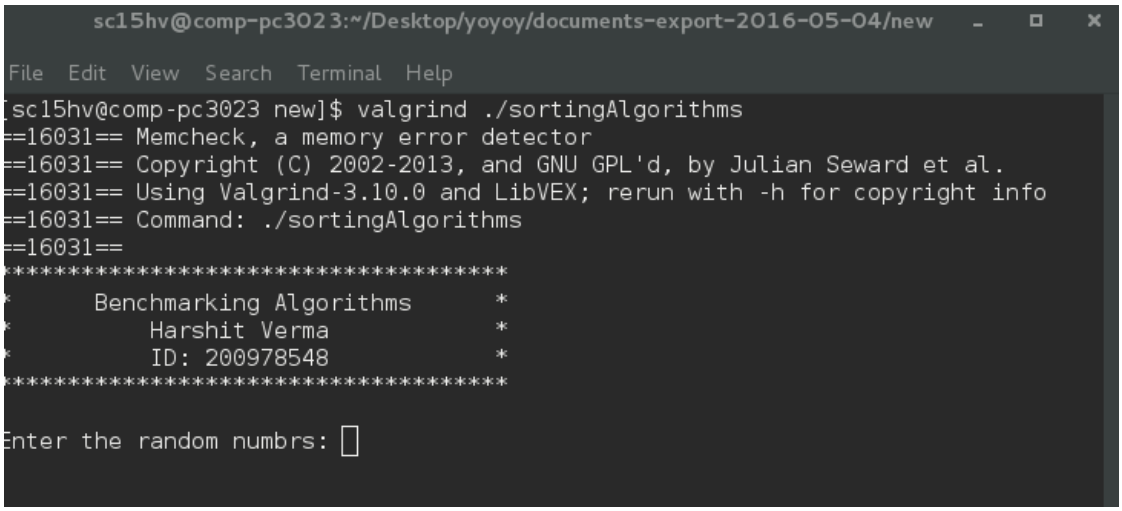
graph.out

```
100000000 40.770000 0.000000 0.000000 0.000000
200000 0.000000 0.000000 0.000000 10.120000
```

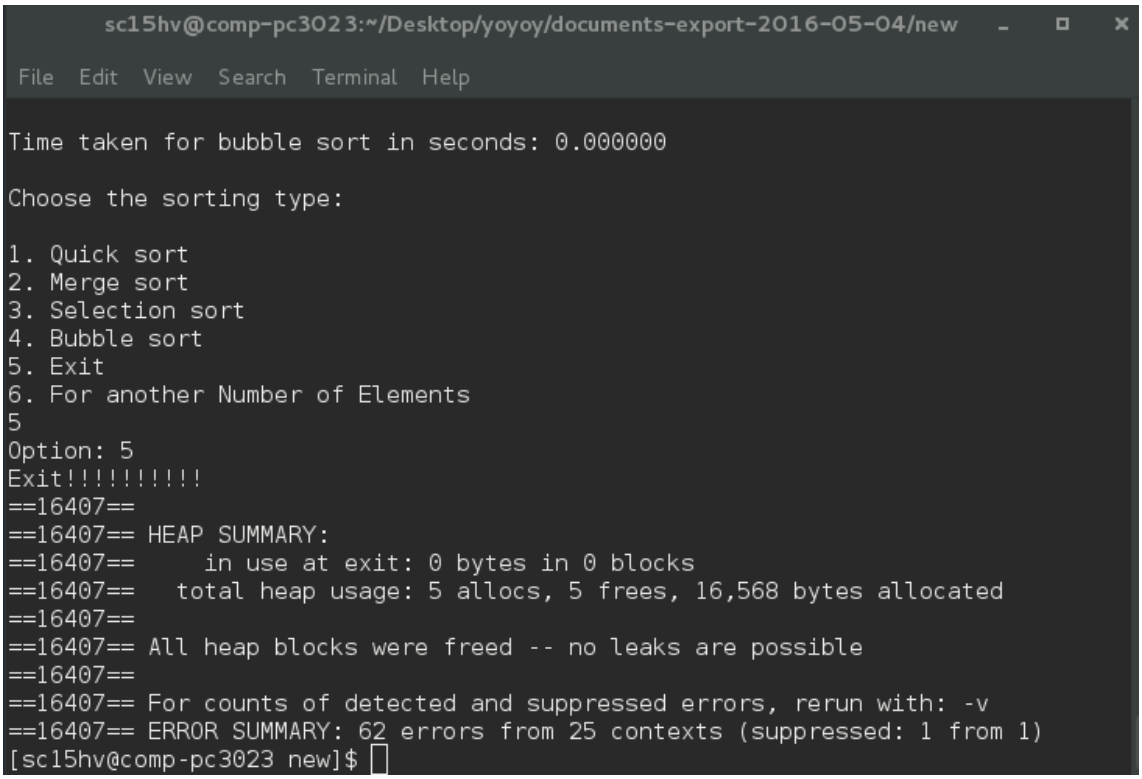
Testing valgrind debugging tool while running the code :

valgrind debugging tool: valgrind is a multipurpose code profiling and memory debugging tool for Linux. It checks and find memory leaks

– I added this tool to my program as I incurred errors in the process of testing my code, as there were memory leaks. therefore I added 'free()' command in my program to free up all used memory at the end so such error was avoided.



As shown above; I have run the all different types of sorting functions and there is no memory leaking errors in my program



Testing some other functions :

S.NO	TESTING THE FUNCTIONS	EXPECTED	ACTUAL
1.	rand() function	It should generates the random numbers.	This function generated random numbers succesfully
2.	swapForBubbleSort(), swapforQuickSort(), swapforMergeSort(), swaforSelecionSort()	It should swap the two different numbers in the elements	These functions swap the numbers according to numeric order.
3.	printBubbleSorting(), printquickSorting(), printselectionSorting(), printMergeSorting().	It should print out the sorted list.	These functions printing out the sorted list
4.	bubbleSorting()	It is the main function which should implement the bubble sorting	This function sort out the unsorted list by using bubble sorting method.
5.	merge()	It is the main function which should implement the merge sorting	This function sort out the sorted list by using merge sorting method
6.	quickSorting()	It is the main function which should implement the quick sorting	This function sort out the sorted list by using merge sorting method
7.	SelectionSort()	It is the main function which implement the selection sort	This function sort out the sorted list by using merge selection sort
8.	testBubbleSorting() testQuickSorting() testMergeSorting() testSelectionSorting()	These functions are calling the print function and main implement function to complete the sortings	These functions calls the print and main implement function properly which is called inside main file.
9.	time	It should display the time taken for algorithm to sort out the unsorted random numbers	It shows the time taken by the different sorting to sort out. **And I have added the tests screen shots of every sorting above for different random numbers**
10.	graph.gnu	This file should display the result on gnuplot as graph where x is length and y is time	It's plotting the point on graph. **And I have also added the tests and screen shots of gnuplot graphs.**

CONCLUSION

In this conclusion I will outline any disadvantages to my code and any potential improvements I would recommend to myself and what benefits these will have on the program.

The first recommendation is to add more OPCOUNTS as this would to count the operational functions of the code and it also make the program execute faster.

I would also add more comments at the start of each section rather than just a short line.

This shows that the quick sort method remains the fastest above all and this method has been improved due to the efficiency of my code using the methods I have.