REPORT OF ASSIGMENT 1

1-)QUESTION DESCRIPTION

In the given question, the C program should be coded based on user selection for comparing the three different sort algorithms(Bubble Sort, Selection Sort and Merge Sort)

Program can generate array that sized 10 between 1.000.000.000 has random assigned number contents by using dynamic memory allocation techniques.

Each sort algorithm must run M times (M>9) for same number of inputs and calculation of average run time for each algorithm required. In addition, this step should be done for different number of inputs. For a good comparison, 3 different sort algorithms should run for the array created as random inputs one time. Then randomize same sized array contents again for next iteration of algorithms' time calculation.

The size of biggest array that can be allocated dynamically must be found and the aforementioned steps must be done for this array.

Finally, the program give runtime of sort algorithms as output.

2-)SOLUTION DESCRIPTION

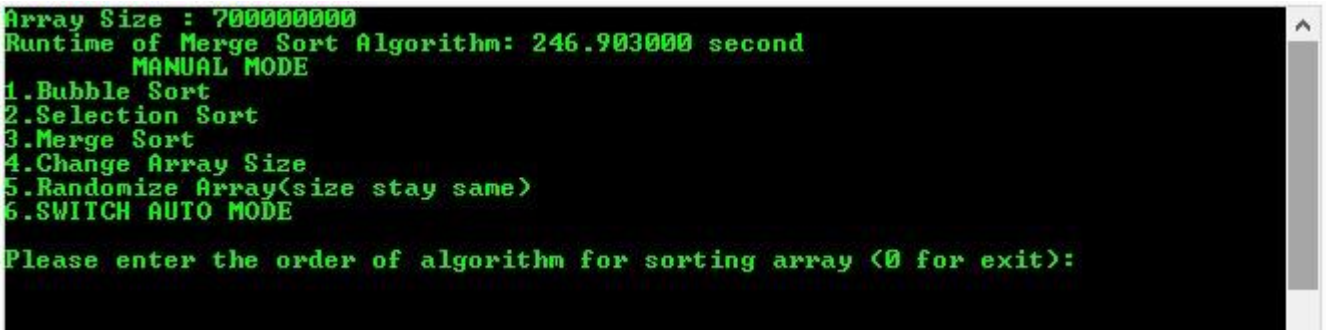
The program is based on user selection. User give the size of inputs. Then, the memory allocated dynamically for given size of inputs if possible. In addition, after main array allocated and randomized, the temporary array allocated as destination array and copy main contents not to lose array for 3 three algorithms' healty comparison. User can choose each sort algorithm to see the runtime. The program seperated as two mode Manual and Auto.

In manual mode, sort algorithms run one by one and see the runtimes of each algorithms that be chosen. Besides, user can change size of array (realloc runs for that and helps to apply algorithms for different number of inputs) or can randomize allocated array again for manual average time calculation of sort algorithms without exit program.

In Auto Mode, user give iteration number to see results which contains all sorting runtimes per iteration and average runtimes of all sort algorithms. It is coded to get results more easy way.

The size of biggest array that can be allocated is 2.147.483.647 for integer size variable if you choose unsigned int size variable it is 4.294.967.295 But it is based on your computer RAM capacity and changed instant memory usage on time.

In my pc there is 5,2 gb useable memory for some time. Array size is almost ((5*1024*1024*1024)/4)= 1.342.177.280 But in program I allocate max 3 array(main,temp and for merge sort) Useable array size is 447.392.426 for mentioned steps part of question. I just tried it on merge sort for 700.000.000 instance.



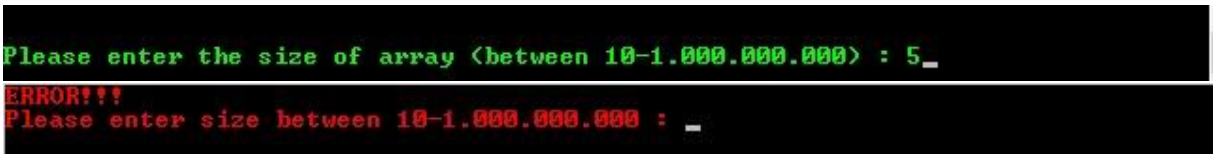
HARDWARE SPECIFICATION OF COMPUTER THAT USED FOR THAT PROGRAM

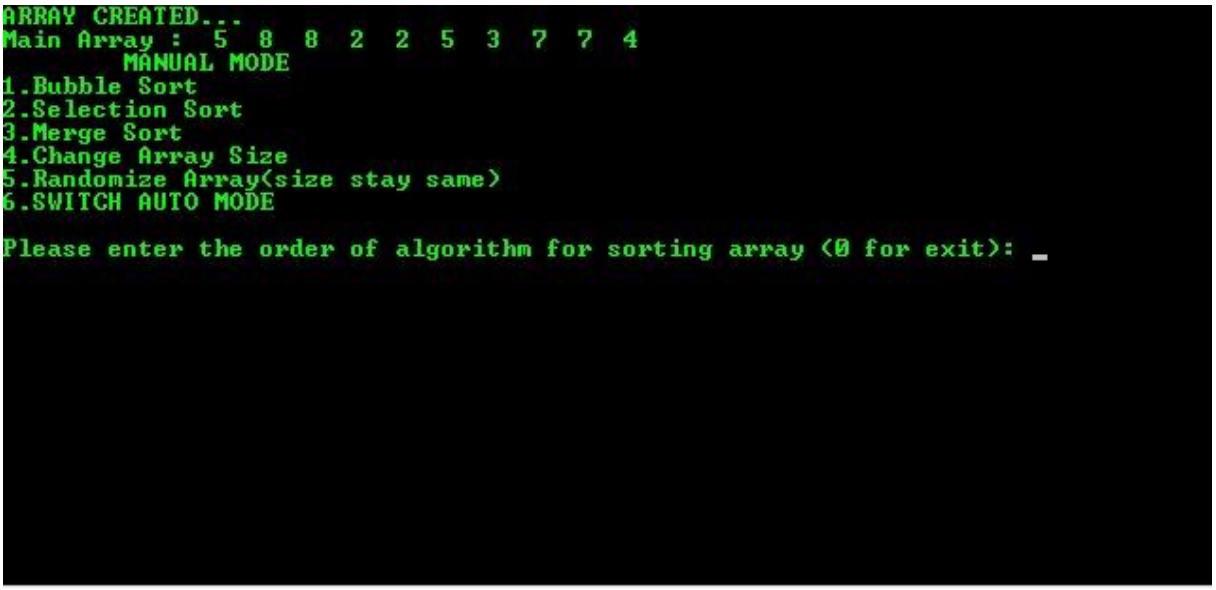
CPU	**:**	INTEL(R) CORE(TM) i5-2410M CPU @2.30 GHz
RAM	**:**	8 GB
Compiler Used	**:**	GCC
IDE	**:**	DEV-C++(Version 5.11)
Operating System	**:**	Windows Embedded 8.1 Industry Pro

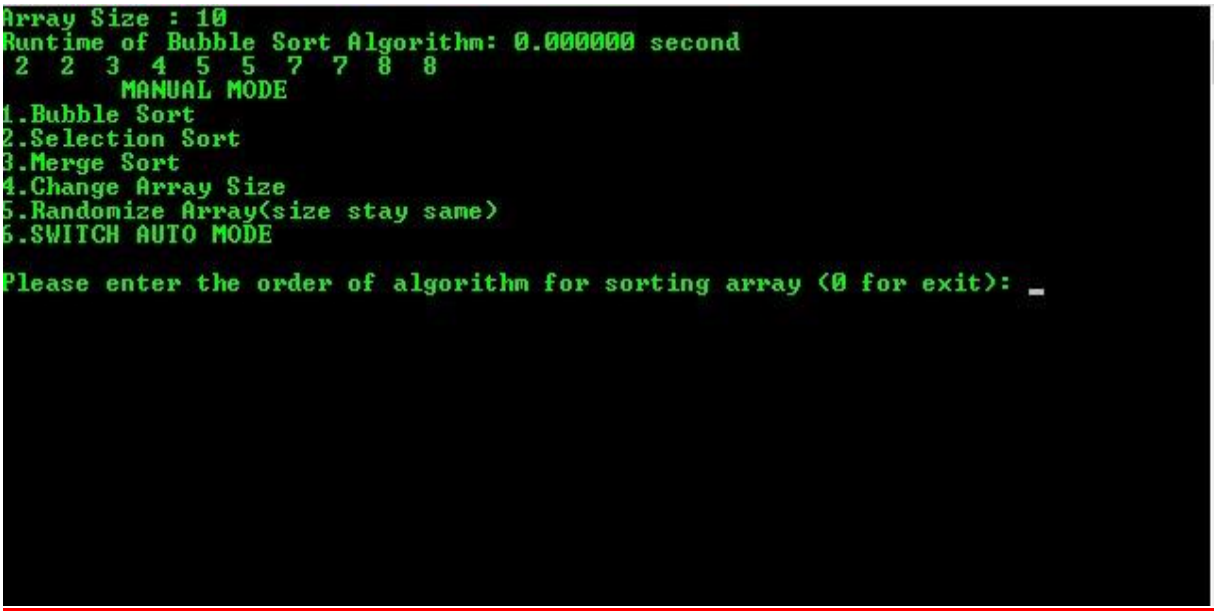
3-)ANALYSİS

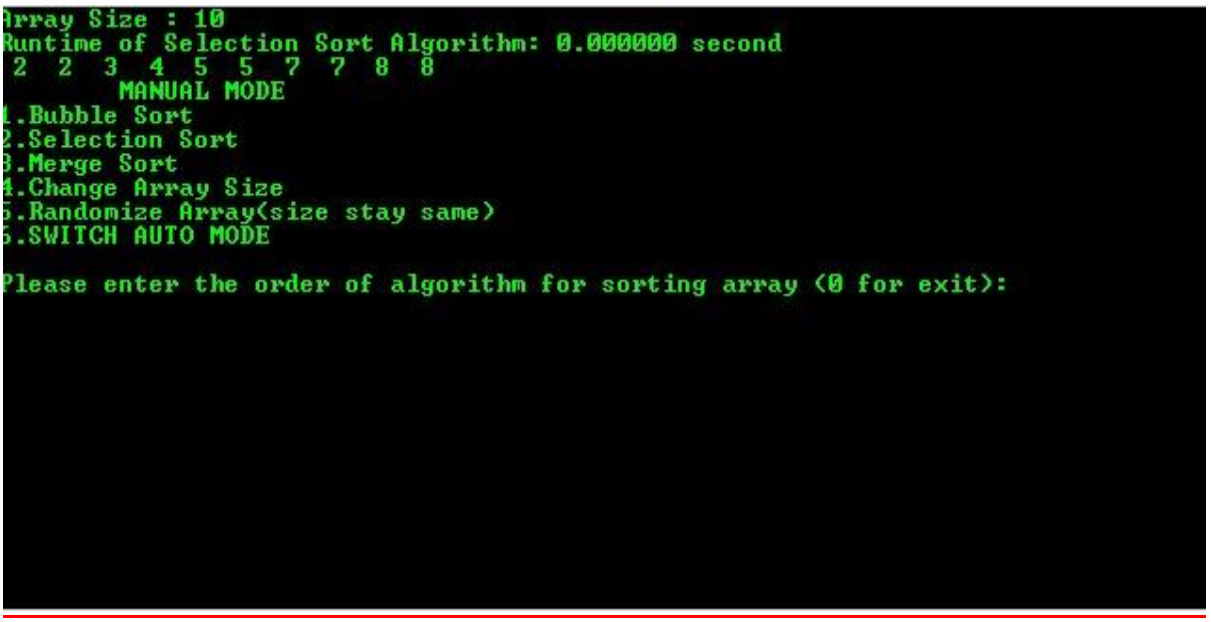
SCREENSHOTS

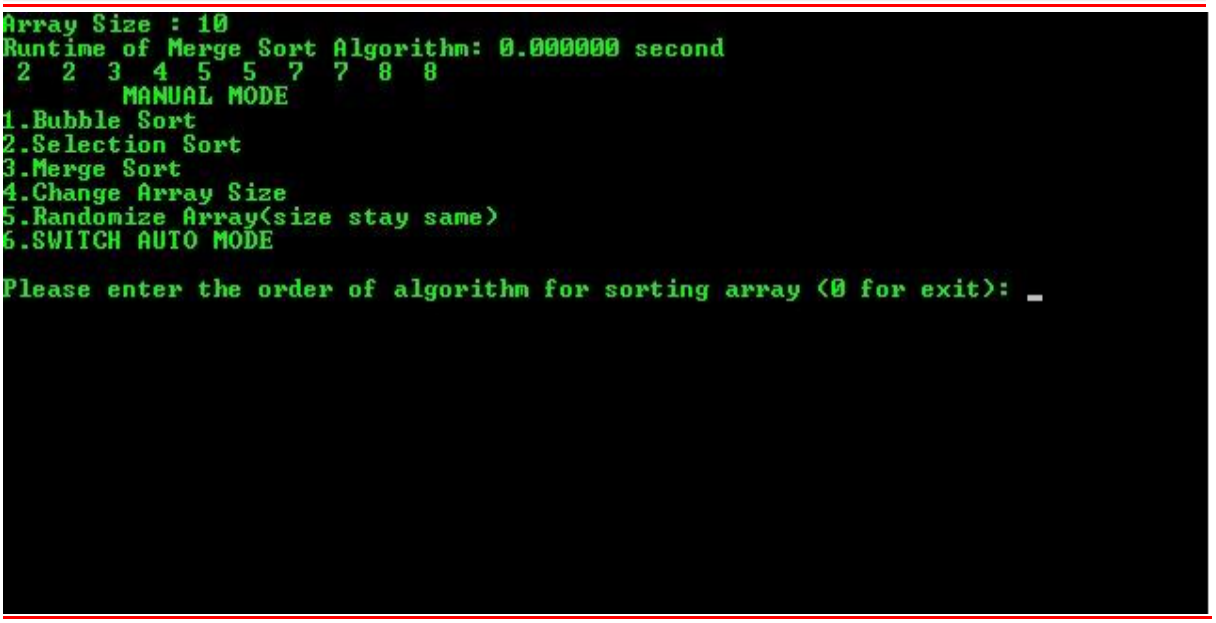
FOR CORRECTNESS AND MANUAL MODE DETAILS

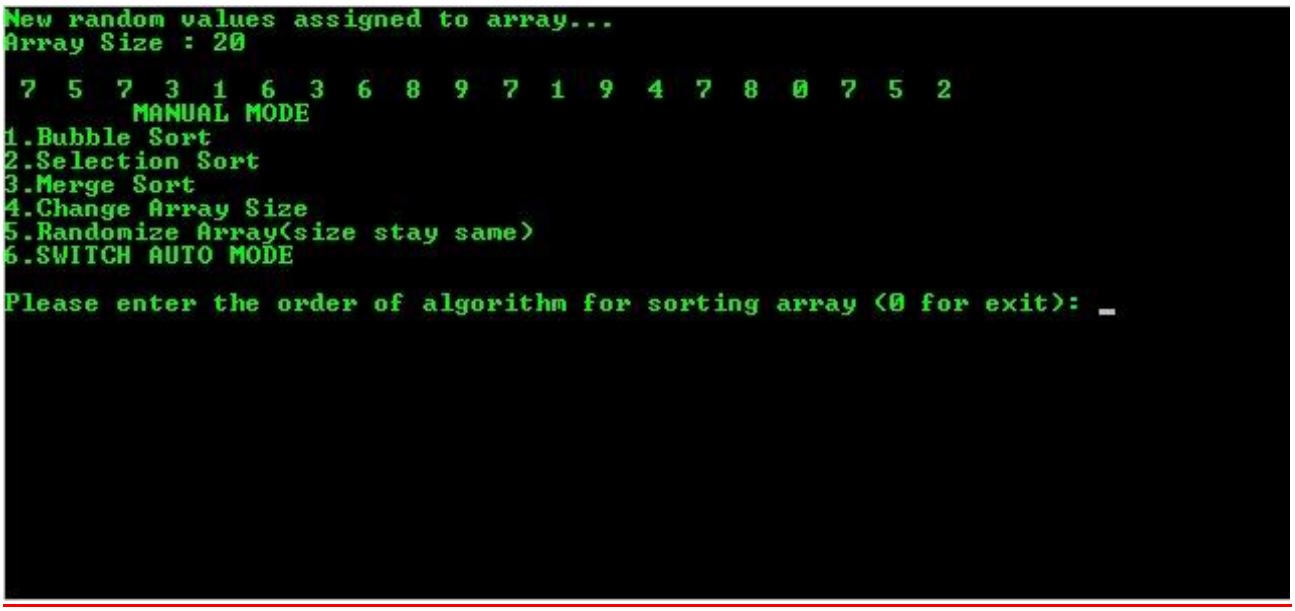




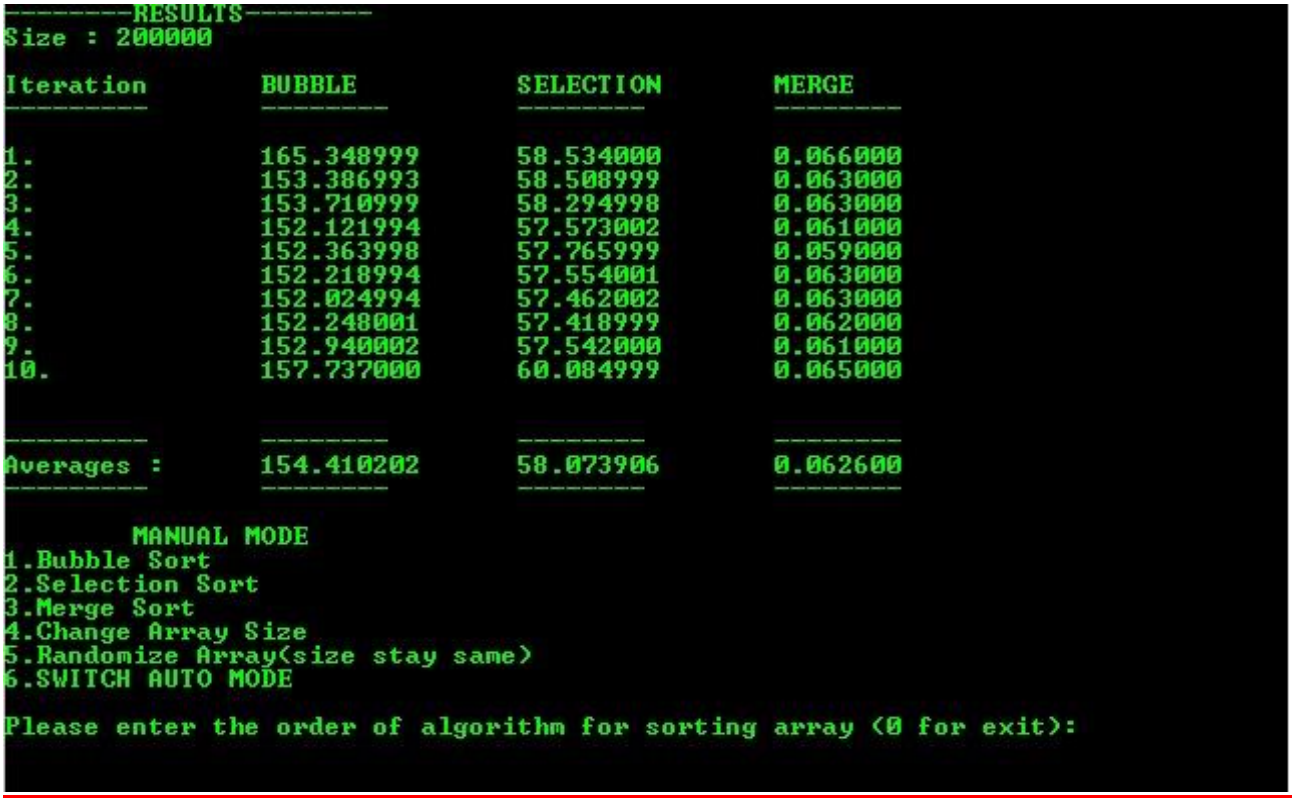

Array Size : 10
Runtime of Bubble Sort Algorithm: 0.000000 second
2 2 3 4 5 5 7 7 8 8
MANUAL MODE
1.Bubble Sort
2.Selection Sort
3.Merge Sort
4.Change Array Size
5.Randomize Array(size stay same)
6.SWITCH AUTO MODE
Please enter the order of algorithm for sorting array (0 for exit):

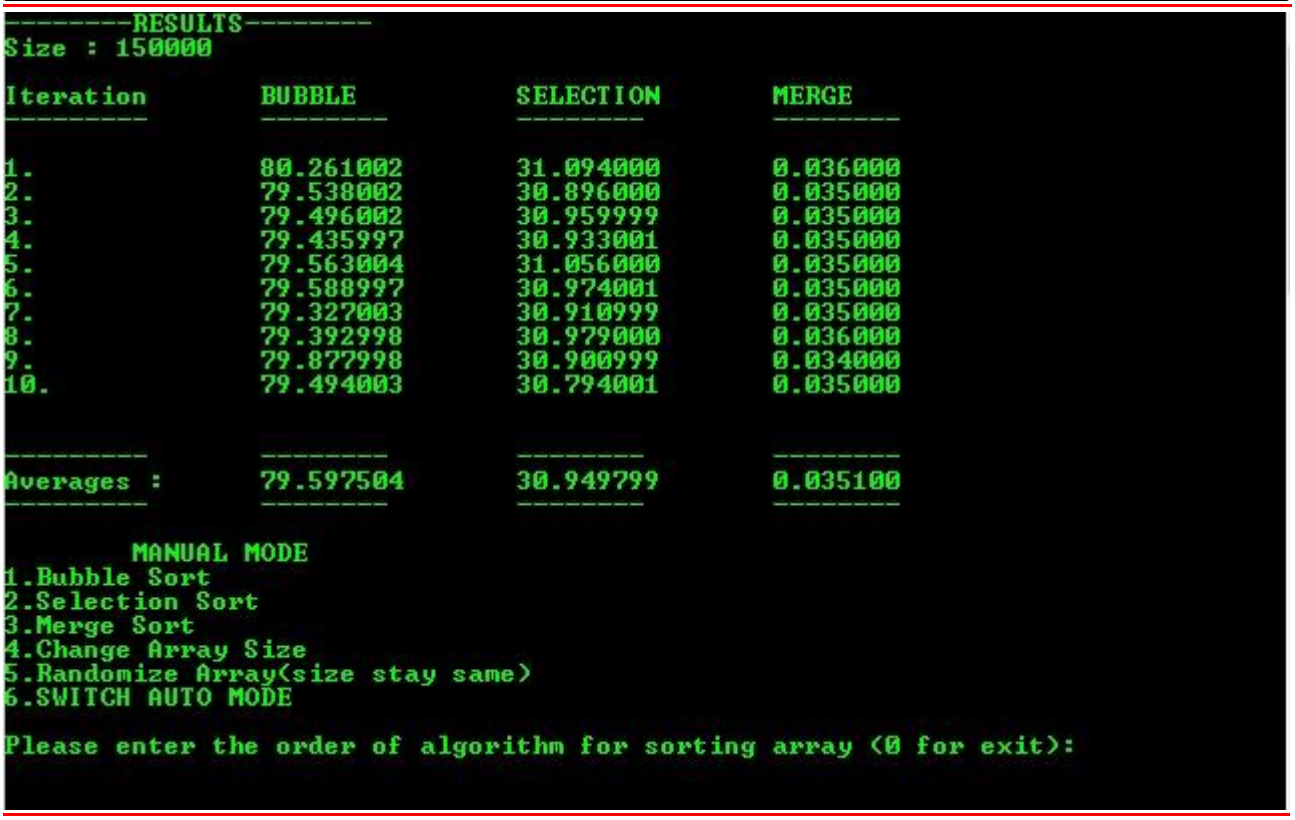

Array Size : 10
Runtime of Selection Sort Algorithm: 0.000000 second
2 2 3 4 5 5 7 7 8 8
MANUAL MODE
1.Bubble Sort
2.Selection Sort
3.Merge Sort
4.Change Array Size
5.Randomize Array(size stay same)
6.SWITCH AUTO MODE
Please enter the order of algorithm for sorting array (0 for exit):

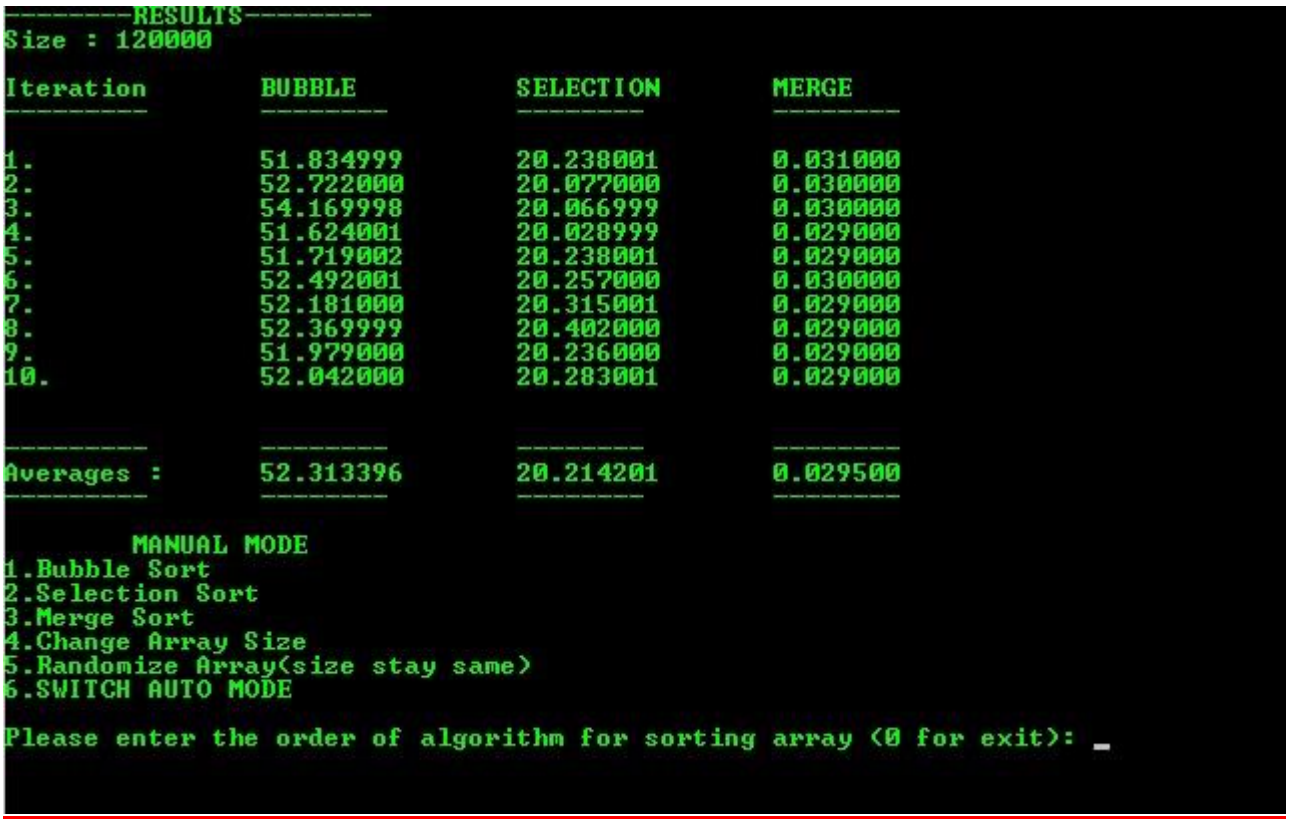

Array Size : 10
Runtime of Merge Sort Algorithm: 0.000000 second
2 2 3 4 5 5 7 7 8 8
MANUAL MODE
1.Bubble Sort
2.Selection Sort
3.Merge Sort
4.Change Array Size
5.Randomize Array(size stay same)
6.SWITCH AUTO MODE
Please enter the order of algorithm for sorting array (0 for exit):



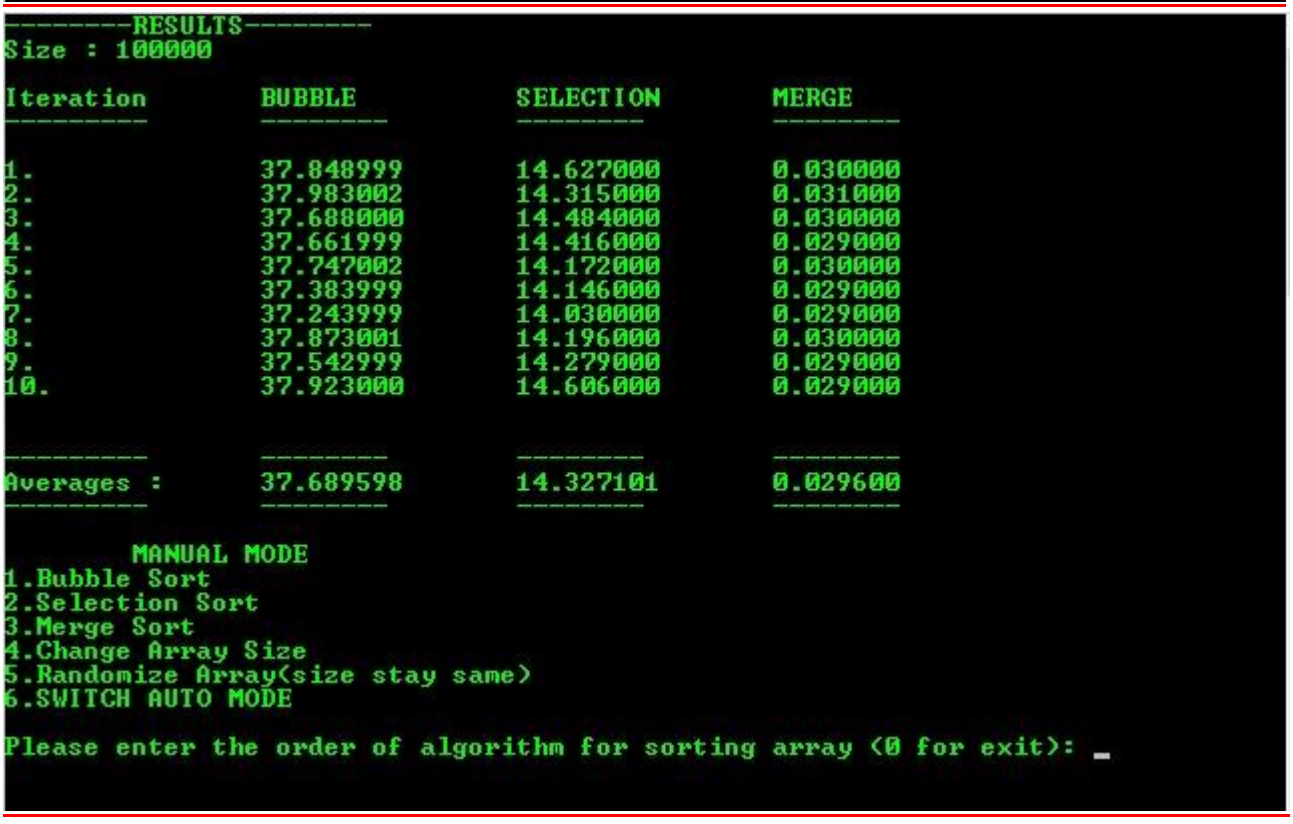
FOR AVERAGE RUNTIMES AND AUTO-MODE RUN


--------RESULTS--------
Size : 200000
Iteration BUBBLE SELECTION MERGE
1. 165.348999 58.534000 0.066000
2. 153.386993 58.508999 0.063000
3. 153.710999 58.294998 0.063000
4. 152.121994 57.573002 0.061000
5. 152.363998 57.765999 0.059000
6. 152.218994 57.554001 0.063000
7. 152.024994 57.462002 0.063000
8. 152.248001 57.418999 0.062000
9. 152.940002 57.542000 0.061000
10. 157.737000 60.084999 0.065000
Averages : 154.410202 58.073906 0.062600
MANUAL MODE
1.Bubble Sort
2.Selection Sort
3.Merge Sort
4.Change Array Size
5.Randomize Array(size stay same)
6.SWITCH AUTO MODE
Please enter the order of algorithm for sorting array (0 for exit):


--------RESULTS--------
Size : 150000
Iteration BUBBLE SELECTION MERGE
1. 80.261002 31.094000 0.036000
2. 79.538002 30.896000 0.035000
3. 79.496002 30.959999 0.035000
4. 79.435997 30.933001 0.035000
5. 79.563004 31.056000 0.035000
6. 79.588997 30.974001 0.035000
7. 79.327003 30.910999 0.035000
8. 79.392998 30.979000 0.036000
9. 79.877998 30.900999 0.034000
10. 79.494003 30.794001 0.035000
Averages : 79.597504 30.949799 0.035100
MANUAL MODE
1.Bubble Sort
2.Selection Sort
3.Merge Sort
4.Change Array Size
5.Randomize Array(size stay same)
6.SWITCH AUTO MODE
Please enter the order of algorithm for sorting array (0 for exit):



```
--------RESULTS--------
Size : 120000

Iteration       BUBBLE          SELECTION       MERGE
---------       --------        --------        --------

1.              51.834999       20.238001       0.031000
2.              52.722000       20.077000       0.030000
3.              54.169998       20.066999       0.030000
4.              51.624001       20.028999       0.029000
5.              51.719002       20.238001       0.029000
6.              52.492001       20.257000       0.030000
7.              52.181000       20.315001       0.029000
8.              52.369999       20.402000       0.029000
9.              51.979000       20.236000       0.029000
10.             52.042000       20.283001       0.029000


---------       --------        --------        --------
Averages :      52.313396       20.214201       0.029500
---------       --------        --------        --------

        MANUAL MODE
1.Bubble Sort
2.Selection Sort
3.Merge Sort
4.Change Array Size
5.Randomize Array(size stay same)
6.SWITCH AUTO MODE

Please enter the order of algorithm for sorting array (0 for exit): _
```



```
--------RESULTS--------
Size : 100000

Iteration       BUBBLE          SELECTION       MERGE
---------       --------        --------        --------

1.              37.848999       14.627000       0.030000
2.              37.983002       14.315000       0.031000
3.              37.688000       14.484000       0.030000
4.              37.661999       14.416000       0.029000
5.              37.747002       14.172000       0.030000
6.              37.383999       14.146000       0.029000
7.              37.243999       14.030000       0.029000
8.              37.873001       14.196000       0.030000
9.              37.542999       14.279000       0.029000
10.             37.923000       14.606000       0.029000


---------       --------        --------        --------
Averages :      37.689598       14.327101       0.029600
---------       --------        --------        --------

        MANUAL MODE
1.Bubble Sort
2.Selection Sort
3.Merge Sort
4.Change Array Size
5.Randomize Array(size stay same)
6.SWITCH AUTO MODE

Please enter the order of algorithm for sorting array (0 for exit): _
```

```
--------RESULTS---------
Size : 50000

Iteration       BUBBLE          SELECTION       MERGE
---------       --------        --------        --------

1.              9.103000        3.532000        0.014000
2.              9.166000        3.531000        0.014000
3.              9.212000        3.582000        0.014000
4.              9.771000        3.747000        0.015000
5.              9.438000        3.561000        0.017000
6.              9.395000        3.538000        0.014000
7.              9.214000        3.515000        0.014000
8.              9.142000        3.539000        0.014000
9.              9.254000        3.477000        0.013000
10.             9.187000        3.583000        0.014000


---------       --------        --------        --------
Averages :      9.288198        3.560500        0.014300
---------       --------        --------        --------

        MANUAL MODE
1.Bubble Sort
2.Selection Sort
3.Merge Sort
4.Change Array Size
5.Randomize Array(size stay same)
6.SWITCH AUTO MODE

Please enter the order of algorithm for sorting array (0 for exit):
```

```
--------RESULTS---------
Size : 20000

Iteration       BUBBLE          SELECTION       MERGE
---------       --------        --------        --------

1.              1.325000        0.544000        0.007000
2.              1.356000        0.550000        0.005000
3.              1.322000        0.556000        0.004000
4.              1.325000        0.561000        0.005000
5.              1.338000        0.561000        0.004000
6.              1.328000        0.556000        0.004000
7.              1.340000        0.548000        0.006000
8.              1.320000        0.539000        0.004000
9.              1.331000        0.547000        0.004000
10.             1.332000        0.549000        0.005000


---------       --------        --------        --------
Averages :      1.331700        0.551100        0.004800
---------       --------        --------        --------

        MANUAL MODE
1.Bubble Sort
2.Selection Sort
3.Merge Sort
4.Change Array Size
5.Randomize Array(size stay same)
6.SWITCH AUTO MODE

Please enter the order of algorithm for sorting array (0 for exit):
```

4-)SOURCE CODE

```
/**
@file
BLM2541 spring2016 assignment 1.

A program that calculate runtimes of three sorting algorithms which are Bubble Sort, Selection Sort
and Merge Sort. It has two modes: Manual and Auto.
In manuel mode you can choose any algorithm that you want to calculate runtime or randomize array
again or create new sized random array.It based on user selection.
In auto mode you can enter iteration number and try all algorithms based on iteration. After trying
all algorithms, array size stay same and new random array values will assign.
Then you can see on the screen average runtime for each algorithm.

@author

Name                 :           Muhammed Yasin SAĐLAM
Student No           :           15011804
Date                 :           30/10/2016
E-Mail               :           myasinsaglam1907@gmail.com
Compiler Used  :           GCC
IDE                   :           DEV-C++(Version 5.11)
Operating System :         Windows Embedded 8.1 Industry Pro
*/
#include<stdio.h>
#include<stdlib.h>
#include<time.h>

//     Function procedures that used in this program
void printArray(int *, int);
void Sel_sort(int *, int);
void Bubble_sort(int *, int);
void arr_Copy(int *, int *, int);
void merge(int *, int, int, int);
void Merge_sort(int *, int, int);
int* Allocator(int);
void Randomizer(int *, int);
/**
Main function, Reads array size from the user and controls it(Size have to be 10 between
1.000.000.000 ).
Allocates (2 times-first array and temp array) memory for arrays that will be sorted. Because we
need two array to not lose same randomized array.
Then it seperates program two mode : MANUAL and AUTO.
-In MANUAL MODE-
It reads order of any algorithm from the user to calculate runtime or randomize array again or
create new sized random array.
Then it prints runtime of chosen algorithm on the screen
-In AUTO MODE-
It reads iteration number from the user and apply all algorithms based on iteration. After applying
all algorithms, array size stay same and new random array values will assign.
Then you can see on the screen average runtimes for all algorithms.
*/
int main() {
	system("COLOR a");
	srand(time(NULL));
	int size, choice, i, iter;                                          //variables for get
size(size), user selection(choice), iteration number in auto mode(iter)
	float bubbletime, selectime, mergetime;                        //variables to calculate average
runtimes of each algorithm
	clock_t start, end;                                                          //variables for
calculating runtimes of one algorithm
	int *array1, *tmp_arr;                                                      //array
declarations (array1 is main array, tmp_arr is the array that will be sorted array in sorting
algorithms)

	printf("\n\nPlease enter the size of array (between 10-1.000.000.000) : ");
	scanf("%d", &size);                                                  //Reading Size
	while (size<10 || size>1000000000) {                        //Size control for determined
scale
```

```
            system("CLS");
            system("COLOR c");
            printf("ERROR!!!\nPlease enter size between 10-1.000.000.000 : ");
            scanf("%d", &size);                                     //Reading Size until entering
values of given scale
      }
      system("COLOR a");
      system("CLS");
      array1 = Allocator(size);                                     //It allocates memory for array1
with calling by function
      Randomizer(array1, size);                                     //It assigns random values into
array1 with calling by function
      printf("ARRAY CREATED...\n");
      /*printf("Main Array : ");
      printArray(array1,size);                                      //print to control with calling
by function   */

      printf("\tMANUAL MODE\n1.Bubble Sort\n2.Selection Sort\n3.Merge Sort\n4.Change Array
Size\n5.Randomize Array(size stay same)\n6.SWITCH AUTO MODE \n\nPlease enter the order of algorithm
for sorting array (0 for exit): ");
      scanf("%d", &choice);                                                 //Reading choice from the
user
      system("CLS");

      while (choice != 0) {
            tmp_arr = Allocator(size);                              //It allocates temp array with
calling by function.It written in while loop because the size can be change based user selection in
while.
            arr_Copy(tmp_arr, array1, size);                        //It copies main array to temp
array with calling by function
                                                                                                  //
      printArray(tmp_arr,size);                                 //print to control
            printf("WORKING... PLEASE WAIT...\n");
            if (choice == 1) {                                                    //This choice
calculates bubble sort runtime
                  start = clock();
                  Bubble_sort(tmp_arr, size);                           //Bubble sorting with
calling by function
                  end = clock();
                  system("CLS");
                  printf("Array Size : %d\nRuntime of Bubble Sort Algorithm: %f second\n", size,
((float)(end - start)) / 1000);
                  //    printArray(tmp_arr,size);                       //print to control
            }
            if (choice == 2) {                                                    //This choice
calculates selection sort runtime
                  start = clock();
                  Sel_sort(tmp_arr, size);                              //Selection sorting with
calling by function
                  end = clock();
                  system("CLS");
                  printf("Array Size : %d\nRuntime of Selection Sort Algorithm: %f second\n",
size, ((float)(end - start)) / 1000);
                  //    printArray(tmp_arr,size);                       //print to control
            }
            if (choice == 3) {                                                    //This choice
calculates selection sort runtime
                  start = clock();
                  Merge_sort(tmp_arr, 0, size - 1);             //Merge sorting with calling by
function
                  end = clock();
                  system("CLS");
                  printf("Array Size : %d\nRuntime of Merge Sort Algorithm: %f second\n", size,
((float)(end - start)) / 1000);
                  //printArray(tmp_arr,size);                     //print to control
            }
            if (choice == 4) {                                                    //This choice
changes array size
                  system("CLS");
```

```
                    printf("Enter New size : ");
                    scanf("%d", &size);                                     //Reading new size
                    while (size<10 || size>1000000000) {    //Size control for determined scale
                           system("CLS");
                           system("COLOR c");
                           printf("ERROR!!!\nPlease enter size between 10-1.000.000.000 : ");
                           scanf("%d", &size);                      //Reading Size until entering
values of given scale
                           system("CLS");
                           system("COLOR a");
                    }
                    system("CLS");
                    array1 = (int*)realloc(array1, (sizeof(int)*size));    //Main array Re-
Allocation for New Size
                    if (!array1) {                                          //Allocation Control
                           system("COLOR c");
                           printf("RE-ALLOCATION FAILED!!!... Quitting...");
                           return 0;
                    }
                    Randomizer(array1, size);                       //Randomizing New sized main
array with calling by function
                    printf("New sized random array created... \nNew Array Size : %d\n\n", size);
                    //     printArray(array1,size);                         //print to control

             }
             if (choice == 5) {                                                    //This choice
assign new randomized values to main array(size stay same)
                    Randomizer(array1, size);                       //Randomizing main array with
calling by function
                    system("CLS");
                    printf("New random values assigned to array...\nArray Size : %d\n\n", size);
                    //     printArray(array1,size);                         //print to control
                    //     system("PAUSE");
             }
             if (choice == 6) {                                                    //This choice for
passing AUTO mode
                    system("CLS");
                    printf("Size : %d\n", size);
                    printf("\tAUTO MODE\nPlease enter iteration number M ( M>9 is suggested ) :");
                    scanf("%d", &iter);
      //Reading iteration number
                    system("CLS");
                    printf("--------RESULTS--------\nSize : %d \n\n", size);
                    printf("Iteration\tBUBBLE\t\tSELECTION\tMERGE\n");
                    printf("---------\t--------\t--------\t--------\n\n");
                    bubbletime = 0; selectime = 0; mergetime = 0;
                    for (i = 0; i<iter; i++) {
                           printf("%d.\t\t", i + 1);

                           arr_Copy(tmp_arr, array1, size);                              //temp array
values are turn into main array values with calling by function

                           start = clock();
                           Bubble_sort(tmp_arr, size);                                   //Bubble
sorting with calling by function
                           end = clock();
                           printf("%f\t", ((float)(end - start)) / 1000);        //Output the
runtime
                           bubbletime += ((float)(end - start)) / 1000;
      //Calculating all of bubble sort runtimes

                           arr_Copy(tmp_arr, array1, size);                              //temp array
values are turn into main array values with calling by function

                           start = clock();
                           Sel_sort(tmp_arr, size);
      //Selection sorting with calling by function
                           end = clock();
```

```
                        printf("%f\t", ((float)(end - start)) / 1000);        //Output the
runtime
                        selectime += ((float)(end - start)) / 1000;
        //Calculating all of selection sort runtimes

                        arr_Copy(tmp_arr, array1, size);                              //temp array
values are turn into main array values with calling by function

                        start = clock();
                        Merge_sort(tmp_arr, 0, size - 1);                             //Merge
sorting with calling by function
                        end = clock();
                        printf("%f\n", ((float)(end - start)) / 1000);        //Output the
runtime
                        mergetime += ((float)(end - start)) / 1000;
        //Calculating all of merge sort runtimes

                        Randomizer(array1, size);
        //Randomizing main array with calling by function
                }
                printf("\n\n---------\t--------\t--------\t--------\n");
                printf("Averages :\t%f\t%f\t%f\n", (bubbletime / iter), (selectime / iter),
(mergetime / iter));       //Output the avereage runtimes of algorithms
                printf("---------\t--------\t--------\t--------\n\n");
            }
            //printArray(tmp_arr,size);                                  //print to control
            free(tmp_arr);                                                        // Free temp
array until new choice

        //printArray(array1,size);                                  //print to control
            printf("\tMANUAL MODE\n1.Bubble Sort\n2.Selection Sort\n3.Merge Sort\n4.Change Array
Size\n5.Randomize Array(size stay same)\n6.SWITCH AUTO MODE \n\nPlease enter the order of algorithm
for sorting array (0 for exit): ");
            scanf("%d", &choice);                                                 //Reading choice
from the user
            system("CLS");
        }


        free(array1);                                                         //Free main array
        system("PAUSE");
        return 0;
}
/**
@param *A                 array that will be printed on screen
@param size               size of array
*/
void printArray(int *A, int size) {
        int i;
        for (i = 0; i < size; i++)
            printf(" %d ", A[i]);
        printf("\n");
}

/**
@param *arr2        destination array of copying
@param *arr1        source array of copying
@param size               size of array
*/
void arr_Copy(int *arr2, int *arr1, int size) {
        int i;
        for (i = 0; i<size; i++)
            arr2[i] = arr1[i];
}
/**
@param *arr               array that will be selection sorted
@param size               size of array
*/
void Sel_sort(int *arr, int size) {
```

```
	int i, j, temp, Mindex;
	for (i = 0; i<size; i++) {
		Mindex = i;
		for (j = i + 1; j<size; j++) {
			if (arr[Mindex]>arr[j])
				Mindex = j;
		}
		temp = arr[Mindex];
		arr[Mindex] = arr[i];
		arr[i] = temp;
	}
}
/**
@param *arr		array that will be bubble sorted
@param size		size of array
*/
void Bubble_sort(int *arr, int size) {
	int i, j, temp;
	for (i = 1; i<size; i++) {
		for (j = 0; j<size - i; j++) {
			if (arr[j]>arr[j + 1]) {
				temp = arr[j];
				arr[j] = arr[j + 1];
				arr[j + 1] = temp;
			}
		}
	}
}
/**
@param *arr		array that will be merged
@param l_index		left index of array
@param mid		middle index of array
@param r_index		right index of array
*/
void merge(int *arr, int l_index, int mid, int r_index) {
	int *temp;
	temp = (int*)malloc(sizeof(int)*(r_index - l_index + 1));
	if (!temp) {
		system("COLOR c");
		printf("Not enough space for merge sort!!! Quitting...");
		exit(0);
	}
	int i, j, k;
	i = l_index;
	j = mid + 1;
	k = 0;
	while (i <= mid && j <= r_index) {
		if (arr[i] <= arr[j])
			temp[k++] = arr[i++];
		else
			temp[k++] = arr[j++];
	}
	while (i <= mid)
		temp[k++] = arr[i++];
	while (j <= r_index)
		temp[k++] = arr[j++];
	k--;
	while (k >= 0) {
		arr[l_index + k] = temp[k];
		k--;
	}


	free(temp);
}
/**
@param *arr		array that will be merge sorted
@param left		start index of array
@param right		end index of array
```

```
*/
void Merge_sort(int *arr, int left, int right) {
	if (left<right) {
		int mid = (left + right) / 2;
		Merge_sort(arr, left, mid);
		Merge_sort(arr, mid + 1, right);
		merge(arr, left, mid, right);
	}
}
/**
@param size		size of array that will be allocated
@return array		allocated array using malloc
*/

int *Allocator(int size) {
	int *array;
	array = (int*)malloc(sizeof(int)*size);
	if (!array) {
		system("COLOR c");
		printf("Array Not Allocated !!! Quitting...");
		return 0;
	}
	return array;
}
/**
@param *arr		array that will be randomized
@param size		size of array
*/
void Randomizer(int *arr, int size) {
	int i;
	for (i = 0; i<size; i++)
		arr[i] = rand();
}
```