Andrew Collins
Section: 002

Final Exam/Project

Guide Line:

1. Description of problem
2. Description of the Algorithms
3. Testing and Experimental Data
4. C and C++ files used.

Description of the Problem:

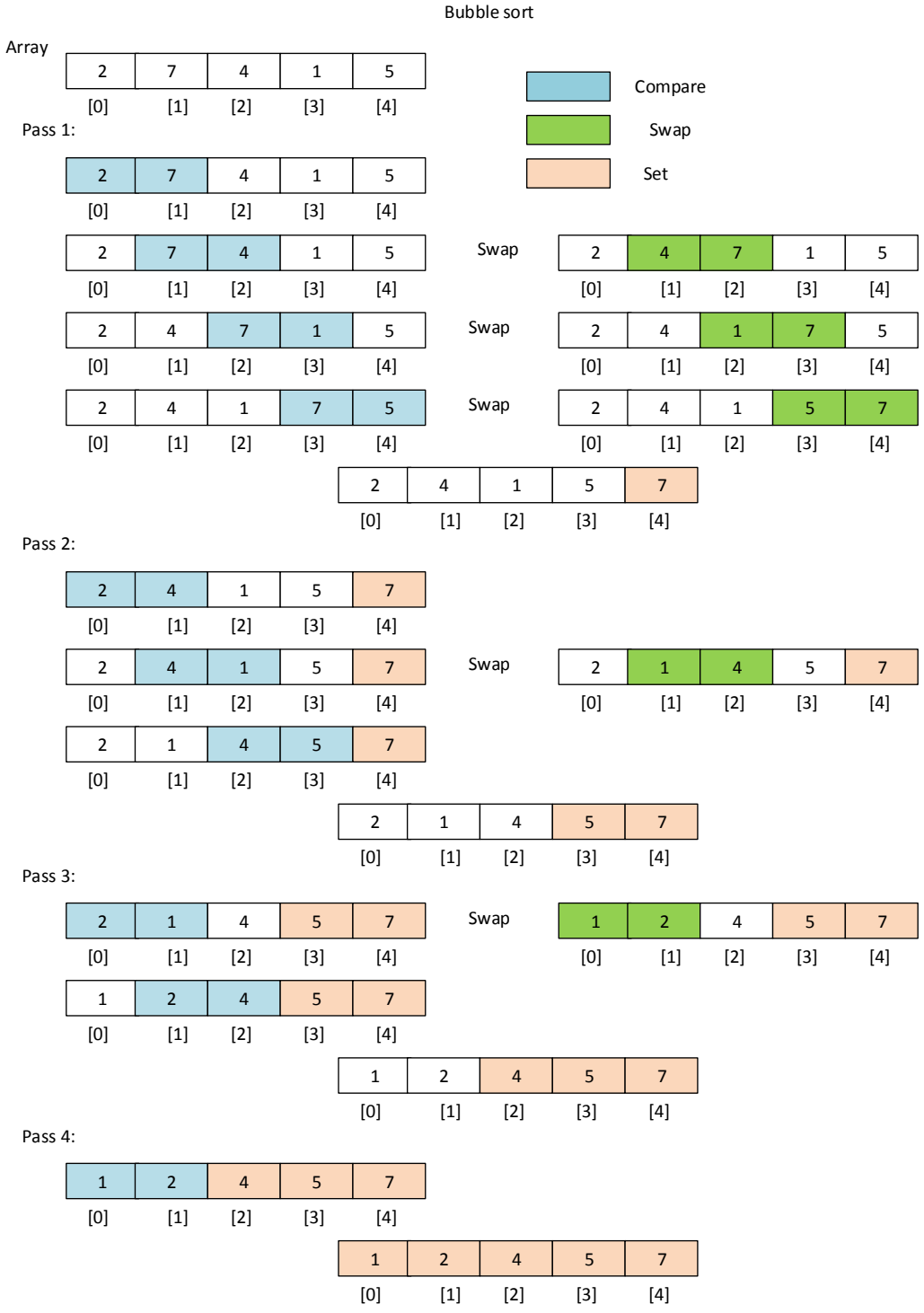
The project deals with the understanding of different sorting algorithm: bubble sort, insertion sort, selection sort and merge sort. Find the worst case Big O for each algorithm and explain how each algorithm works using text and pictures. Given the sorting.c file, we are to look at the code for each sorting algorithm and add comments and headers that explain what the code does. Then we are to create a minimum of 15 test cases and implement each sorting algorithm and record the times.

Andrew Collins
Section: 002

Description of the Algorithms:

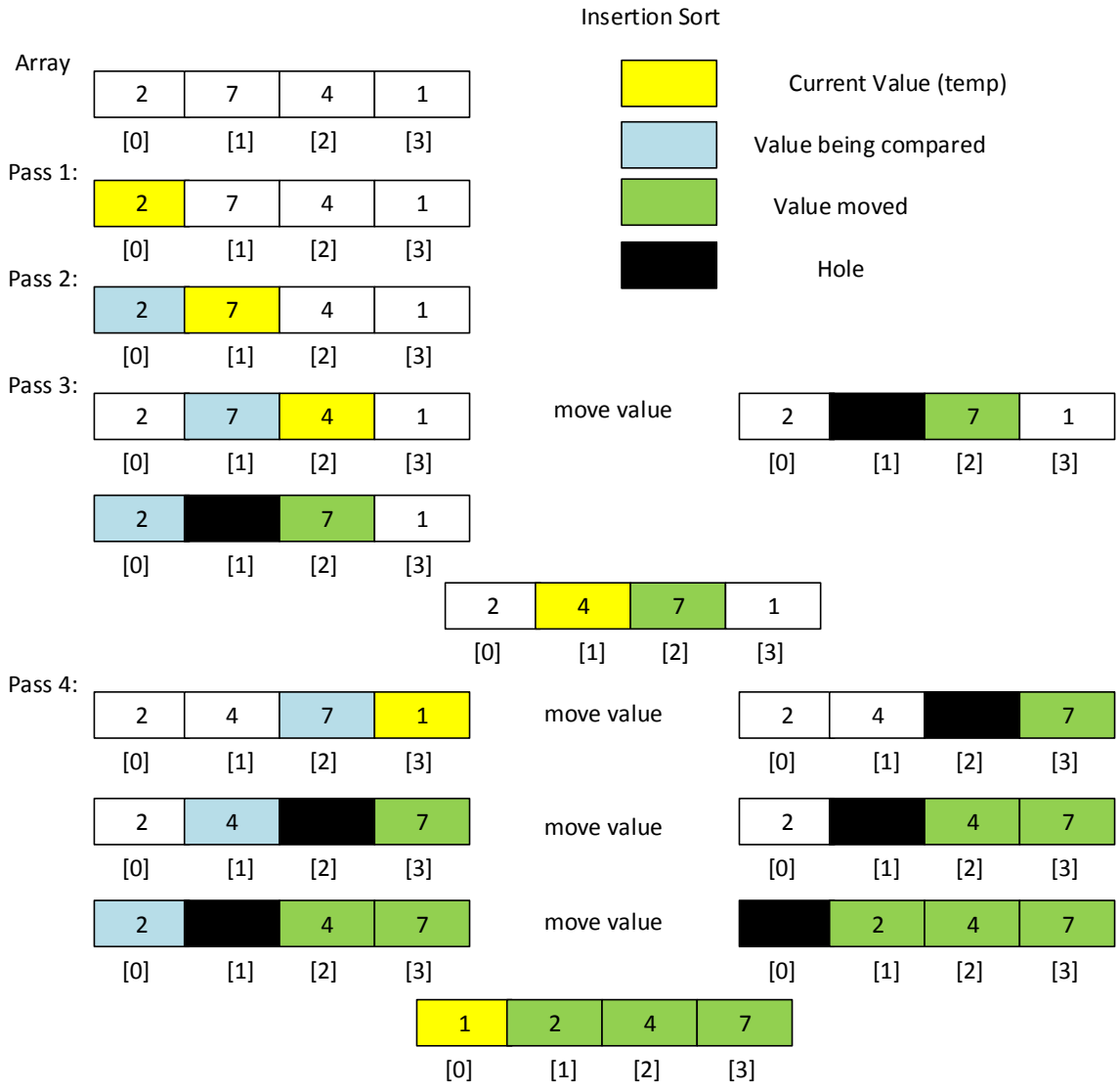
Bubble sort:

Bubble sort is a comparison bases algorithm in which pairs of adjacent elements are compared and if they are not in order they are swapped. The worst case time complexity (or worst case Big O) for this algorithms is O(n^2) when n is the number of elements. More details in "sorting.c".



Andrew Collins
Section: 002

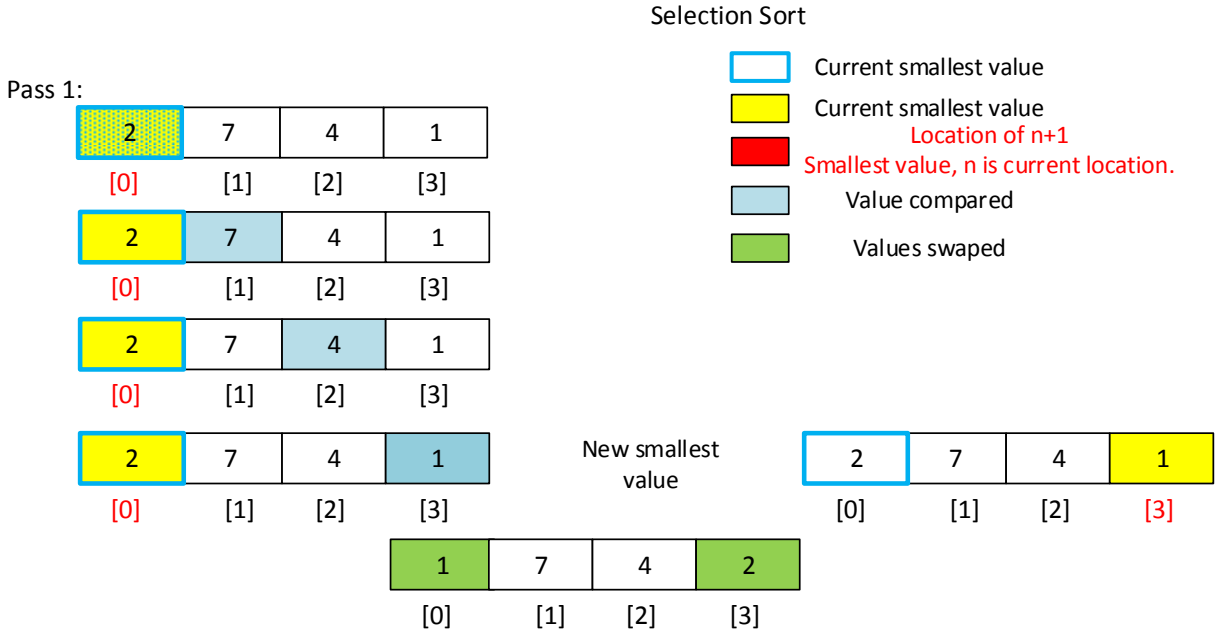
Insertion Sort:
Insertion sort is a comparison base algorithm. An element in an array is compared to is compared to previous elements in that array, (for ascending order) if the previous element in the array is larger than the current element the previous element is moved to the one location to the right writing over the location of our current location, we next look at the n-2 (n is the location of our current value), and do the same but if the value is larger we move it to the right writing over n-1. This loops till the previous value (n-x= the location of the value being compare n-x >=0) is no less than our current value and our current is written to n-x+1 or if n-x is the first location in the array then and that value is grater then the value is mover to the right and n-x location gets our current value. The worst case time complexity (or worst case Big O) for this algorithms is O(n^2) when n is the number of elements. More details in “sorting.c”.



Andrew Collins
Section: 002

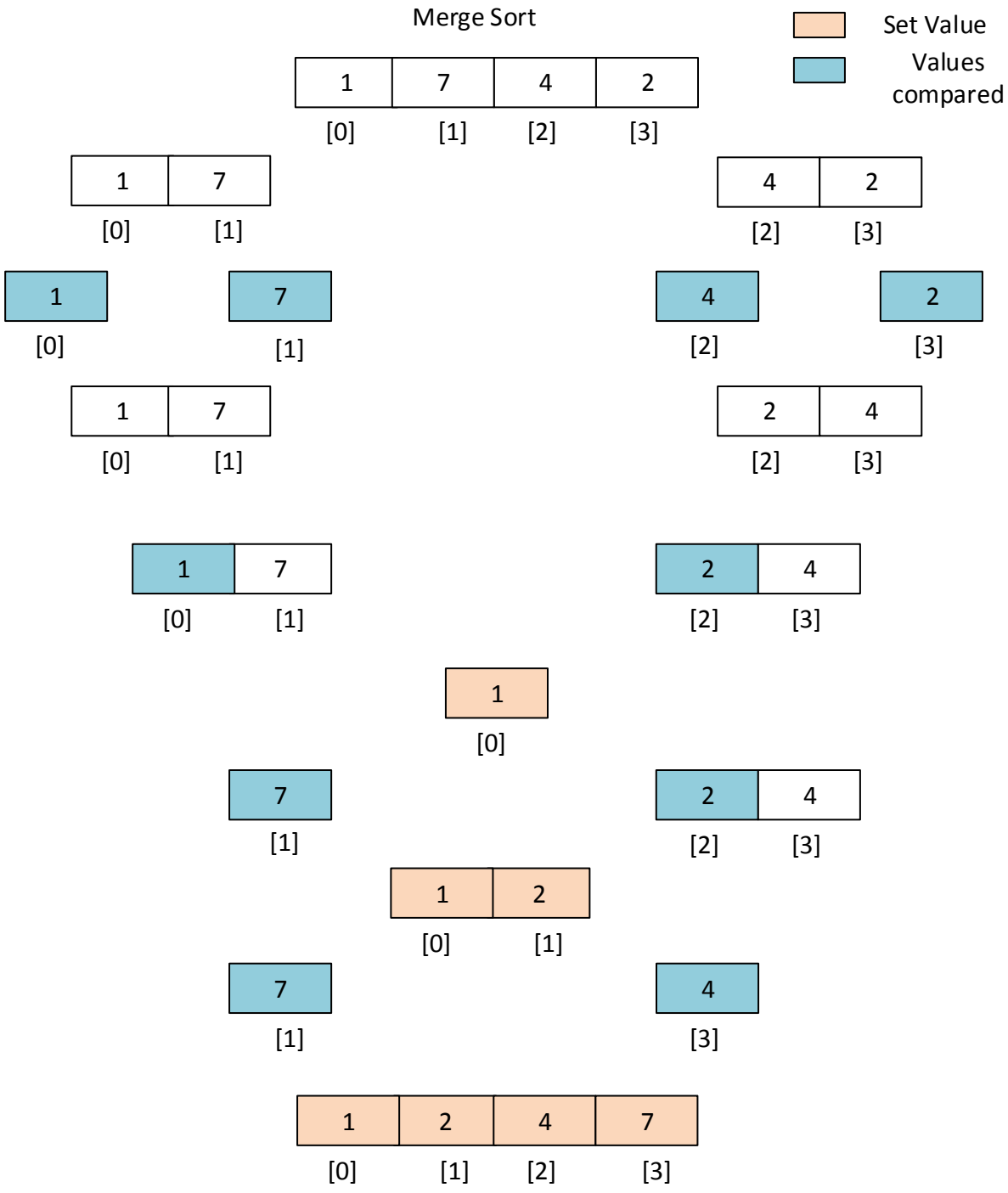
Selection Sort:

Selection sort is a comparison based sort. We start with our first or current location array location, whose value will be our current smallest value, we will compare our current smallest value to the next value in our array. If the next value is smaller than our current smallest value it becomes our current smallest value, we now compare this value to the next value. If ever our current smallest is greater than the next vale the next value becomes our current smallest value. After we go through our array and find the smallest value we swap the value of our current array location and the current smallest value, then move to the next array location and repeat. The worst case time complexity (or worst case Big O) for this algorithms is O(n^2) when n is the number of elements.



Andrew Collins
Section: 002

Merger Sort:
Merger sort is a comparison base sorting algorithm. The main principle behind the algorithm is to break up the sorting into smaller sections. The array is divide into half's, till we are comparing two element of our array. Once a section is sorted its values are compared to other sections sorted values, this continues till we have our original sorted array. The worst case time complexity (or worst case Big O) for this algorithms is O(nlog(n)) when n is the number of elements.



Andrew Collins
Section: 002

Testing and Experimental Data:

To begin my testing, I started by writing a .ccp file that would generate 15 .txt values, the each name of each file corresponds to the number of values in it (2.txt 2 value to 16384.txt). The values in each test file were randomly generated and were positive integer between 0 – 10000. After I had the files setup I made changes to the sorting.c file so the program would read from each file automatically and run a sorting algorithm for each .txt file.

At an N (number of value) of 128 is when we see the differences in the algorithms timings.

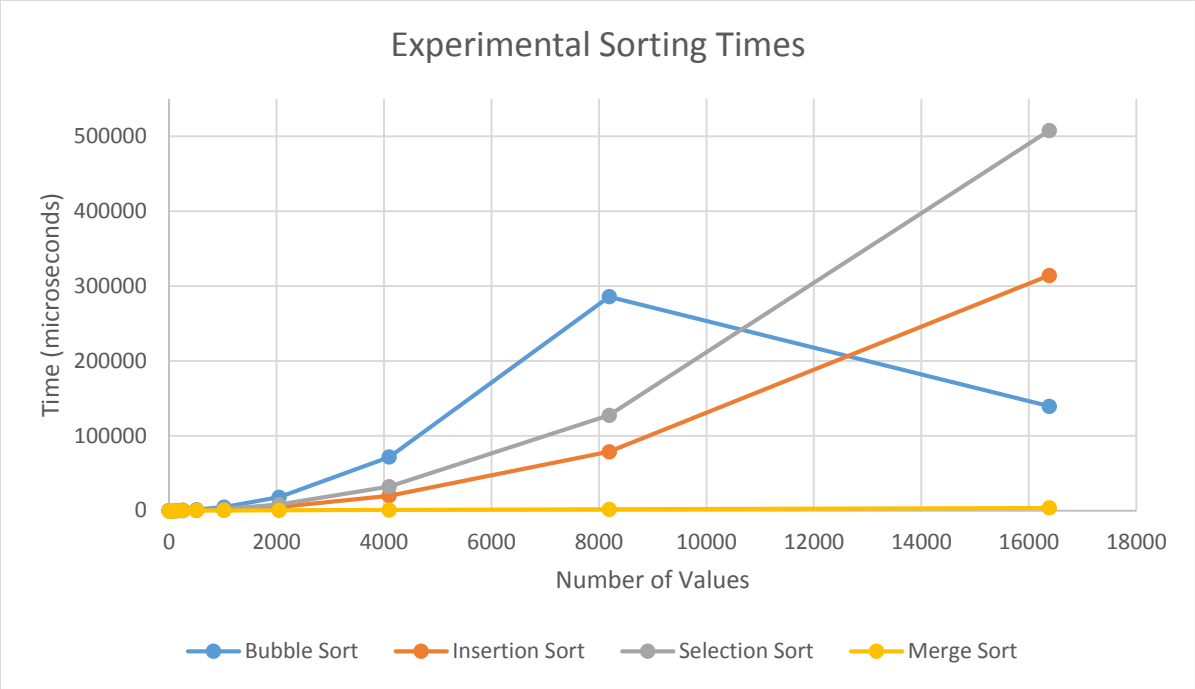
From the graph we can see that the timings for the Insertion, Selection and Bubble sorts seemed to increase at a steep pace as our values for N increased, as for the Merge sort the timings barely increased.

After looking at the big O of each algorithm, the timings for each were mostly as expected. However the timing for the bubble did drop for one of our values for N. I would account this random unsorted order of the N file tested.

The way the file was sorted dose make a difference in the timing of the algorithms. Since each algorithm uses different swapping and comparing methods the timing bases on a presorted or reverse sorted file will differ. For example if we have a reverse sorted file the selection sort will take longer since it will have to keep changing the smallest value as it goes through the entire array.

		Experimental Sorting Times (micro seconds)			
	Input Values (N)	Bubble Sort	Insertion Sort	Selection Sort	Merge Sort
1	1	0	0	0	0
2	2	0	0	0	0
3	4	1	0	0	1
4	8	0	0	1	1
5	16	2	1	1	2
6	32	5	2	4	4
7	64	17	6	11	8
8	128	67	20	39	16
9	256	270	95	144	36
10	512	1119	316	533	79
11	1024	4552	1325	2058	176
12	2048	17967	4822	8087	377
13	4096	71610	19786	31991	802
14	8192	285627	78517	127266	1715
15	16384	139249	314092	507877	3649

Andrew Collins
Section: 002



		Time Complexity Worst Case Big O			
		O(n^2)	O(n^2)	O(n^2)	O(nlog(n))
	Input Values (N)	Bubble Sort	Insertion Sort	Selection Sort	Merge Sort
1	1	1	1	1	0
2	2	4	4	4	0.602059991
3	4	16	16	16	2.408239965
4	8	64	64	64	7.224719896
5	16	256	256	256	19.26591972
6	32	1024	1024	1024	48.16479931
7	64	4096	4096	4096	115.5955183
8	128	16384	16384	16384	269.7228761
9	256	65536	65536	65536	616.5094311
10	512	262144	262144	262144	1387.14622
11	1024	1048576	1048576	1048576	3082.547156
12	2048	4194304	4194304	4194304	6781.603742
13	4096	16777216	16777216	16777216	14796.22635
14	8192	67108864	67108864	67108864	32058.49042
15	16384	268435456	268435456	268435456	69049.05629

Andrew Collins
Section: 002

Write.cpp

```cpp
#include <iostream>
#include <fstream>
#include <string>
#include <cstdlib>
#include <sstream>
using namespace std;

int main(){


    ofstream outfile;
    string filename;
    int num=1;
    //cout<<"Enter number of values:";
    //cin>> num;
    //cout<<"Enter file name:";
    //cin>>filename;

    cout<<endl;
    srand(time(NULL));

    for(int f=0;f<15;f++){
    stringstream ss;
       ss<<num;
       filename =ss.str()+".txt";
       cout<<filename<<endl;
       outfile.open(filename.c_str());
       for(int i=0;i<num;i++){


          outfile<<rand()%10000;
          outfile<<"\n";


       }

       outfile.close();
       num=num*2;
    }


    return 0;
}
```

Andrew Collins
Section: 002

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
typedef struct timeval time;
void print_array(int *, int);
void merge(int *, int, int, int);
void merge_sort(int *, int, int);
void selection_sort(int *, int);
void insertion_sort(int *, int);
void bubble_sort(int *, int);
void fill_array(int **, int *i,char *);
void add_num(int **, int *, int);

int main(){
   int fn=1;//fn is the number of values in the file, it is also the filename ("fn"+".txt")
   int *nums=NULL;//our array to hold all the values, intially unsorted
   char name[20];//will hold the name of the file to read from
   int size=0;//is the number of values in the file
   time stop, start;//start and stop time from clock
   int t;//for loop itterator

   //loops 15 times so we can read each file with different number of values to be sorted,
  for( t=0;t<15;t++){
      // int *nums=NULL;
      //char name[20];//me
      //int size=0;
      //time stop, start;

      sprintf(name,"%d",fn);//converts an integer value to a c string,first part of filename**
     strcat(name,".txt");//adds .txt to the C string so we have text file to open **
     // strcpy(name,"test.txt");//adds .txt to the C string so we have text file to open **
     // printf("file name is: %s",name);//print out the file name to be opened**
      fill_array(&nums, &size,name);//passes the file to read to fill array
      //fill_array(&nums, &size);
      //print_array(nums, size);

      //Time the function here
      //printf("Bubble Sort\n");
     // printf("Inserttion Sort\n");
      printf("Merge Sort. Size:%d\n",size);
     // printf("Selection Sort\n");
      gettimeofday(&start, NULL);

      //bubble_sort(nums, size);
      // insertion_sort(nums, size);
      merge_sort(nums, 0, size-1);
      //selection_sort(nums, size);

      gettimeofday(&stop, NULL);

      printf("MicroSeconds: %d\n", stop.tv_usec-start.tv_usec);
      printf("Seconds: %d\n\n", stop.tv_sec-start.tv_sec);

      // print_array(nums, size);
      free(nums);
      nums=NULL;
      fn=fn*2;
      size=0;
      memset(name,0,20);
   }


   return 0;
}
```

Andrew Collins
Section: 002

```
/*************************************************
 * Description: This fills an array with contents
 * from a file
 * Params: address of array pointer to ints and
 *         address of size of the array
 * Pre-conditions:
 *      nums- points to valid array or NULL
 *      size- is accurate number of elements in array >=0
 * Post-conditions:
 *      array- pointer is pointing to valid array of integers or NULL
 *      size- has a positive integer
 * Returns: none
 * ************************************************/
//void fill_array(int **nums, int *size) {
void fill_array(int **nums, int *size,char* name) {
   FILE *fptr;
   char filename[20], num[10];

   //printf("Enter the filename: ");
   //scanf("%s",filename);
   //fptr=fopen(filename,"r");
   fptr=fopen(name,"r");

   while(fscanf(fptr,"%s", num)!=EOF){
      add_num(nums, size, atoi(num));
   }

   fclose(fptr);
}
```

Andrew Collins
Section: 002

```c
/*********************************************
 * Description: this function sorts portions of our array,
 * the sets these portions in our array
 * Params: array of integers and size of the array
 * Returns: none
 * Post-conditions: nums- has unsorted sections
 * Pre-conditions: nums- has sorted sections
 * *********************************************/
void merge(int *nums, int left, int mid, int right){
   int i, j, lower_half, upper_half;
   int temp[(right-left)+1];

   lower_half=left;  //What is this for?sets the left most array locatoin
   upper_half=mid+1;  //What is this for?sets the right most arrat location

   //What does this loop do?
   //starts from the very left of the lower array and very left of the upper array and compares vales,
   // woking its way to the right of each array; setting the next empty location of our temp array
   for(i=0; (lower_half<=mid)&&(upper_half<=right); i++){
      //What does this condition do?
      //if the left value of the lower array is less that the left value of the
      //upper array, set the lower left value to temp next avaliable location,
      // then move to the next value of the lower array.
      if(nums[lower_half]<=nums[upper_half]){
         temp[i]=nums[lower_half];
         lower_half++;
      }
      //What does this condition do?
      //if the left value of the upper array is less that the left value of the
      //lower array, set the upper left value to temp next location avaliable,
      // then move to the next value of the upper array.
      else{
         temp[i]=nums[upper_half];
         upper_half++;
      }
   }

   //What does this condition and loop do?
   //copies the upper half of the sorted array to temp
   if(lower_half>mid)
      for(j=upper_half;j<=right;j++, i++)
         temp[i]=nums[j];
   //What does this else and loop do?
   //copies the lower half our our sorted array to temp
   else
      for(j=lower_half;j<=mid;j++, i++)
         temp[i]=nums[j];

   //What does this loop do?
   //array gets the sorted portions
   for(j=0,i=left;i<=right;i++,j++)
      nums[i]=temp[j];
}
```

```c
/*********************************************
 * Description:this function is recursively called to half our array,
 * Params: array of integers annd the left and right most loction
 * of our array sections of halfs.
 * Pre-conditions:nums- is unsorted array
 * Post-conditions: nums- is sorted array
 * Returns: none
 * *********************************************/
void merge_sort(int *nums, int left, int right) {
   int mid;
   if(left<right) {
      mid=(right+left)/2;
      //  printf("befor 1 Left:%d Right:%d Mid:%d\n",left,right,mid);
      merge_sort(nums, left, mid);  //what does this call do?calls this function recursivelly sending it the
      //left half of each section of  the array
      //  printf("after 1 Left:%d Right:%d Mid:%d\n",left,right,mid);
      merge_sort(nums, mid+1, right); //what does this call do?calls the function recursively sending it the
      //right half of each section of the array
      merge(nums, left, mid, right);  //what does this call do?this function will sort and set sections of our
      //array.
   }
}
```

Andrew Collins
Section: 002

```
/*********************************************
 * Description:This function takes our array and and sorts the array in ascending order.
 * It does this by making sure that that in each location in the array (x), the value in
 * that loaction is the (xth+1) smallest term, so in the 0 location we have the fist smallest term, in the 1
 * location we have the second smallest term and so on.
 * Params: array of integers and size of the array
 * Pre-conditions:
 *      size- is accurate number of elements in the array >=0
 * Post-conditions:
 * Returns:
 * *********************************************/
void selection_sort(int *nums, int size) {
   int i, j;
   int temp, min;

   //What does this loop do?
   //this loop set the location of our value, staring at the first location
   //and loop stops at the second last location in the array.
   //(min) is the location of where the (i+1) smallest value in our array is.
   for(i=0; i<size-1; i++) {
      min=i;
      //What does this loop do?
      //this loop comapers the value at our current location
      //to all the values in our array. If the value at the other location is smaller that
      //the value at our current loaction then (min) will now be the location of the other value.
      //our current value is now the other value and is now compaerd to the rest of the values
      //after our current value.
      for(j=i; j<size; j++)
         if(nums[j]<nums[min])
            min=j;
      //What elements are being swaped?
      //temp will hold the valu of our current location,
      //i- is the location of the (i+1) smallest value should go.
      //so num[i] will get the (i+1) smallest term. (temp) will be moved
      //to where our new (i) term uesd to live.
      temp=nums[i];
      nums[i]=nums[min];
      nums[min]=temp;
   }
}
```

Andrew Collins
Section: 002

```
/*********************************************
 * Description: Bubble Sort- This function takes our array and sorts the array in ascending order.
 *              it uses two loops and a comparitive if statement
 *              outer loop- passes throught the array N times,
 *              N is the number of elements in our array, for each pass one element is set,
 *              first pass largest element is set, second pass second largest elsment is set
 *              and so on.
 *              inner loop- starts at the first element(a), compares to the next(b)
 *              if (a) is > (b) swap, if (a) is swaped (a) is now compared to (c)
 *              if (a) < (c) no swap and (c) is compared to (d), if (c) is the largest element in our array
 *              (c) is bubbled ( compared and swaped) till (c) is set as the last element in the array
 *              and loop ends, then the process is repeated for setting the second largest element, till
 *              size-i-1=0, i is the number of passes which is also the number of elements set.
 * Params: array of integers and size of the array
 * Pre-conditions:
 *      nums-pionts to a valid array of integers, is unsorted
 *      size- is accurate number of elements in the array >=0
 *
 * Post-conditions:
 *      nums-pionts to a valid array of integers, is sorted in ascending order
 * Returns: none
 * *******************************************/
void bubble_sort(int *nums, int size) {
   int i, j;
   int temp;

   //What does this loop do?
   //makes passes through our array, the number os passes depends of the size of our array
   //at the first pass we will set the largest element in the array
   //second pass we will set the second largest element,
   //and this comtinues till all the element are is ascending order.
   for(i=0; i<size; i++) {
      //What does this loop do?
      //the number of times looped depends on the numbers of set elements,each pass one element is set
      //for each pass we will take one element(a) and compare it to the next adjacent
      //element(b), if the first element(a) is grater than the next(b) we will swap them,
      //then we will caompare (a) with the next element (c), if (c) in not greater there is no swap,
      //and (c) is our new value,  now we look at (c) and compare it to the next element (d).
      //So this inner loop continues
      //till the largest element in the array is in the last pasotion in the array.
      //once a value is set in the array the loop ittereates to one less that the last value set.
      for(j=0; j<size-i-1; j++) {
         if(nums[j]>nums[j+1]) {
            temp=nums[j];
            nums[j]=nums[j+1];
            nums[j+1]=temp;
         }

         print_array(nums, size);
      }
   }
}
```

Andrew Collins
Section: 002

```c
/*********************************************
 * Description:Insetring sort- This function takes our array and sorts the array in ascending
 * order.it uses two for loop, the outer (for) loop will loop N times where N is the sizes of the array,
 * the inner (for) loop - has two conditions that control the number of loops, condition 1 is that we have to
 * bet at the second element of our array; condition 2 is that the current element we are looking at
 * must be smaller that the previous element in the array, is these conditons are me we enter the loop,
 * in the loop we move the previous (larger) value to where our current value lives (shift previous value
 * to the right),we now look at the n-2 (n is where our current value lives) if that value is also bigger
 * than our current value we shift it to the right, we continue this till the previous value is not larger
 * than our current value or we are at the first poistion of the array,and our nested loop ends, now
 * or curernt value is inserted in the location where we the previous value is lowere than our current value,
 * or if we are at the first location we insert it there
 * Params: array of integers and size of the array
 * Params:
 * Pre-conditions:
 *      nums-array of unsorted values
 *      size- is accurate number of elements in the array >=0
 * Post-conditions:
 *      nums- is a sorted arrary(in ascending order)
 * Returns: none
 * ********************************************/
void insertion_sort(int *nums, int size) {
   int i, j;
   int temp;

   //What does this loop do?
   //this the outer for loop that well itterate throug each element in our array,
   //assign it to temp, after the nesterd for loop it will assign temp to the apporiate
   //location in our array, based on condition in our nested loop.
   for(i=0; i<size; i++) {
      temp=nums[i];
      printf("temp:%d\n",temp);
      //What does this loop do?
      //the fist condition is that we are at a location in the array that is not the first position
      //second condition is that our temp value or current value is less than the the previous value in our
      //array, if this is the case we enter the loopand move the previous value to the position where our temp
      //lives in our array. we check our conditions again, but now we are looking at the value two previous to
      //our current location(if this location exists), if that value is grater that our temp value it is moved
      //one to the right, this continues till we are at the first location or the previous value is less than
      //our current value.
      for(j=i; j>0 && nums[j-1]>temp; j--)
         nums[j]=nums[j-1];
      //What does this statment do?
      //one we are out of the nested loop, we assign temp to the location in the array
      //where the previous value is less that our temp,or if we are at the first location
      //temp will be put in the first location.
      nums[j]=temp;
   }
}
```

Andrew Collins
Section: 002

```
flip3 ~/cs162/assignments/Final 281% ./s
file name is: 1.txtBubble Sort
MicroSeconds: 0
Seconds: 0

file name is: 2.txtBubble Sort
MicroSeconds: 0
Seconds: 0

file name is: 4.txtBubble Sort
MicroSeconds: 1
Seconds: 0

file name is: 8.txtBubble Sort
MicroSeconds: 0
Seconds: 0

file name is: 16.txtBubble Sort
MicroSeconds: 2
Seconds: 0

file name is: 32.txtBubble Sort
MicroSeconds: 5
Seconds: 0

file name is: 64.txtBubble Sort
MicroSeconds: 17
Seconds: 0

file name is: 128.txtBubble Sort
MicroSeconds: 67
Seconds: 0

file name is: 256.txtBubble Sort
MicroSeconds: 270
Seconds: 0

file name is: 512.txtBubble Sort
MicroSeconds: 1119
Seconds: 0

file name is: 1024.txtBubble Sort
MicroSeconds: 4552
Seconds: 0

file name is: 2048.txtBubble Sort
MicroSeconds: 17967
Seconds: 0

file name is: 4096.txtBubble Sort
MicroSeconds: 71610
Seconds: 0

file name is: 8192.txtBubble Sort
MicroSeconds: 285627
Seconds: 0

file name is: 16384.txtBubble Sort
MicroSeconds: 139249
Seconds: 1
```

```
flip3 ~/cs162/assignments/Final 278% ./s
file name is: 1.txtInserttion Sort
MicroSeconds: 0
Seconds: 0

file name is: 2.txtInserttion Sort
MicroSeconds: 0
Seconds: 0

file name is: 4.txtInserttion Sort
MicroSeconds: 0
Seconds: 0

file name is: 8.txtInserttion Sort
MicroSeconds: 0
Seconds: 0

file name is: 16.txtInserttion Sort
MicroSeconds: 1
Seconds: 0

file name is: 32.txtInserttion Sort
MicroSeconds: 2
Seconds: 0

file name is: 64.txtInserttion Sort
MicroSeconds: 6
Seconds: 0

file name is: 128.txtInserttion Sort
MicroSeconds: 20
Seconds: 0

file name is: 256.txtInserttion Sort
MicroSeconds: 95
Seconds: 0

file name is: 512.txtInserttion Sort
MicroSeconds: 316
Seconds: 0

file name is: 1024.txtInserttion Sort
MicroSeconds: 1325
Seconds: 0

file name is: 2048.txtInserttion Sort
MicroSeconds: 4822
Seconds: 0

file name is: 4096.txtInserttion Sort
MicroSeconds: 19786
Seconds: 0

file name is: 8192.txtInserttion Sort
MicroSeconds: 78517
Seconds: 0

file name is: 16384.txtInserttion Sort
MicroSeconds: 314092
Seconds: 0
```

Andrew Collins
Section: 002

```
flip3 ~/cs162/assignments/Final 292% ./s
file name is: 1.txtSelection Sort
MicroSeconds: 0
Seconds: 0

file name is: 2.txtSelection Sort
MicroSeconds: 0
Seconds: 0

file name is: 4.txtSelection Sort
MicroSeconds: 0
Seconds: 0

file name is: 8.txtSelection Sort
MicroSeconds: 1
Seconds: 0

file name is: 16.txtSelection Sort
MicroSeconds: 1
Seconds: 0

file name is: 32.txtSelection Sort
MicroSeconds: 4
Seconds: 0

file name is: 64.txtSelection Sort
MicroSeconds: 11
Seconds: 0

file name is: 128.txtSelection Sort
MicroSeconds: 39
Seconds: 0

file name is: 256.txtSelection Sort
MicroSeconds: 144
Seconds: 0

file name is: 512.txtSelection Sort
MicroSeconds: 533
Seconds: 0

file name is: 1024.txtSelection Sort
MicroSeconds: 2058
Seconds: 0

file name is: 2048.txtSelection Sort
MicroSeconds: 8087
Seconds: 0

file name is: 4096.txtSelection Sort
MicroSeconds: 31991
Seconds: 0

file name is: 8192.txtSelection Sort
MicroSeconds: 127266
Seconds: 0

file name is: 16384.txtSelection Sort
MicroSeconds: 507877
Seconds: 0
```

```
flip3 ~/cs162/assignments/Final 289% ./s
file name is: 1.txtMerge Sort
MicroSeconds: 0
Seconds: 0

file name is: 2.txtMerge Sort
MicroSeconds: 0
Seconds: 0

file name is: 4.txtMerge Sort
MicroSeconds: 1
Seconds: 0

file name is: 8.txtMerge Sort
MicroSeconds: 1
Seconds: 0

file name is: 16.txtMerge Sort
MicroSeconds: 2
Seconds: 0

file name is: 32.txtMerge Sort
MicroSeconds: 4
Seconds: 0

file name is: 64.txtMerge Sort
MicroSeconds: 8
Seconds: 0

file name is: 128.txtMerge Sort
MicroSeconds: 16
Seconds: 0

file name is: 256.txtMerge Sort
MicroSeconds: 36
Seconds: 0

file name is: 512.txtMerge Sort
MicroSeconds: 79
Seconds: 0

file name is: 1024.txtMerge Sort
MicroSeconds: 176
Seconds: 0

file name is: 2048.txtMerge Sort
MicroSeconds: 377
Seconds: 0

file name is: 4096.txtMerge Sort
MicroSeconds: 802
Seconds: 0

file name is: 8192.txtMerge Sort
MicroSeconds: 1715
Seconds: 0

file name is: 16384.txtMerge Sort
MicroSeconds: 3649
Seconds: 0
```