Caleb Millard

CMPT 360 Spring 2023

Assignment 3

Quick Sort Algorithm Implementation

Assignment	Due Date	group(s)	Language	Language	Platform
1	Monday, Jan 23	1 & 2	Java	Delphi	Windows
2	Monday, Feb 6	1 & 2	C#	Visual basic	Windows
3	Monday, Feb 27th	3	Javascript		Windows
4					
5					
6					
7					

This assignment fulfills the following goals:
Group 3 Language: JavaScript(Node.Js)
Title:
Create and implement a ADT data structure and apply it to a program

Problem:
Learn a new language and create an ADT data structure, then put it into an application

Documentation:

Run the program
The program will output the average of the five grades

Imports used:

C#:
Using System;
Using System.IO;
Using System.Text;

Visual Basic:
Imports System
Imports System.IO
Imports System.Text

Error Handling:
With input, it will sort and attempt to convert all inputs to integers, and if it cannot do this, it will not work. It will work on any integers, positive and negative.

Pseudo Code for both C# and Visual Basic:

Get the input from the text files by iterating through the CSV file line by line and converting strings into integers.

Then find a point from which to start the sort by finding a close middle of the sort(this could be done in many different ways, but I chose this).

Then from this, you will iterate through one side of the split and continue to recursively split and swap pieces so it will sort the array.

Then from this, join all the integers with “, “ and output the sorted data into a text file on the desktop.

Variables used in C# program

Integers:	strlength, bottom, top, bench, splitpoint, position, length, datalength
Arrays:	lines, numbers, data,
Strings:	systemPath, path, input

C# Program Start:

```
using System;
using System.IO;
using System.Text;

//Caleb Millard
//SN: 613362
//CMPT 360 Spring 2023
//Assignment 2
//Title: Find And Implement an Efficient Sorting Algorithm

namespace Caleb{
    public class QuickSort{
        //the method that gathers the data from an input file
        public static int[] obtainData()
        {
            // inputing a data file via a path
            string systemPath = @"C:\Users\Caleb
Millard\Documents\input1000.txt";
            //this allows the file to read each line of the csv or txt
file as a new number
            string[] lines = System.IO.File.ReadAllLines(systemPath);
            int strlength = lines.Length;
            int[] numbers = new int[strlength];
            for(int i = 0;i < strlength;i++)
            {
                numbers[i] = Int32.Parse(lines[i]);
            }
            // we would be able to implement a dictionary to allow us to
parse through a number and assign it a value
            // with that we would also be able to sort alphabetically
because normally the > and < operands dont work on char
            return numbers;
        }
        // pick a number within the array to be your bench point or where
you are splitting the array
        public static int split(int[] data ,int bottom ,int top)
        {
            int bench = data[top];
            int splitpoint = bottom-1;
            //this is iterating throught eh available numbers and swapping
them so they are in the appropriate places in the array
            for (int i = bottom; i<top; i++)
            {
                if (data[i] <= bench)
                {
                    splitpoint += 1;
                    (data[splitpoint], data[i]) = (data[i],
data[splitpoint]);
                }
            }
            (data[splitpoint+1], data[top]) = (data[top],
data[splitpoint+1]);
            return (splitpoint+1);
        }
        // the sort algorithm brings al the pieces together and uses the
split method to combine them and actually sort the data
        public static void sort(int[] data,int bottom,int top)
        {
            if (bottom<top)
            {
                int position = split(data,bottom,top);
                sort(data,bottom,position-1);
                sort(data,position+1,top);
            }
        }
        //this method driver combines all the aspects of the program and
drives it, calling the data and the sorting algorthm methods
        public static void driver()
        {
            int[] data = obtainData();
            int length = data.Length;
            sort(data, 0, length-1);
            string path = @"C:\Users\Caleb
Millard\Desktop\sorted1000.txt";
            int datalength = data.Length;
            string input = "";
                for (int i = 0; i<datalength; i++)
                {
                    input = string.Join(", ", data);
                }
                //creates a new file or overwrites the current output file
                File.WriteAllText(path,input);
                Console.WriteLine("Sort completed and printed");
        }
        static void Main()
        {
            driver();
        }
    }
}
```
End of required code

Variables in Visual Basic Code:

Integer Array:	numberdata, data
String Array:	strdata
Integer:	length, bottom, top, splitpoint, temp1, temp2, position
String:	outputPath, input
Filestream:	streamer
Byte:	bytedata

Visual Basic Program Start:

```
Imports System
Imports System.IO
Imports System.Text
'Caleb Millard
'SN: 613362
'CMPT 360 Spring 2023
'Assignment 2
'Title: Find And Implement an Efficient Sorting Algorithm
Module VBModule
    Public length = 0
    'the method that gathers the data from an input file
    Public Function obtainData() As Integer()
        ' inputing a data file via a path
        Dim systemPath = "C:\Users\Caleb Millard\Documents\data10000.txt"
        'this allows the file to read each line of the csv or txt file as a new number
        Dim strdata() = System.IO.File.ReadAllLines(systemPath)
        length = strdata.Length
        Dim numberdata() = New Integer(length - 1) {}

        For i As Integer = 1 To length
            numberdata(i - 1) = Convert.ToDecimal(strdata(i - 1))
        Next
        'we would be able to implement a dictionary to allow us to parse through a number And
assign it a value
        'with that we would also be able to sort alphabetically because normally the > And <
operands dont work on char
        Return numberdata
    End Function

    Dim bench

    'pick a number within the array to be your bench point or where you are splitting the array
    Function split(data() As Integer, bottom As Integer, top As Integer) As Integer
        Dim splitpoint As Integer

        bench = data(top)
        splitpoint = bottom - 1
        ' this is iterating throught eh available numbers and swapping them so they are in the
appropriate places in the array
        For i As Integer = bottom To top - 1
            If (data(i) <= bench) Then
                splitpoint = (splitpoint + 1)
                Dim temp1 = data(splitpoint)
            data(splitpoint) = data(i)
            data(i) = temp1
         End If
      Next
      Dim temp2 = data(splitpoint + 1)
      data(splitpoint + 1) = data(top)
      data(top) = temp2
      Return splitpoint + 1
   End Function
   'the sort algorithm brings al the pieces together and uses the split method to combine them
and actually sort the data
   Sub sort(data() As Integer, bottom As Integer, top As Integer)
      If (bottom < top) Then
         Dim Position As Integer = split(data, bottom, top)
         sort(data, bottom, Position - 1)
         sort(data, Position + 1, top)
      End If

   End Sub

   'this method driver combines all the aspects of the program And drives it, calling the data And
the sorting algorthm methods
   Sub driver()
      Dim data() = obtainData()
      Dim length = data.Length()
      sort(data, 0, length - 1)
      Dim outputPath As String = "C:\Users\Caleb Millard\Desktop\sorted10000.txt"
      Dim datalength = data.Length
      Dim input = ""
      For i As Integer = 1 To datalength
         input = String.Join(", ", data)
      Next
      Dim streamer As FileStream = File.Create(outputPath)
      ' Add text to the file.
      Dim bytedata As Byte() = New UTF8Encoding(True).GetBytes(input)
      'creates a new file or overwrites the current output file
      streamer.Write(bytedata, 0, bytedata.Length)
      streamer.Close()
   End Sub
   Sub Main()
      driver()
   End Sub
End Module
```
End of required code

For the input sets i put them all on the same line for sake of readability and saving paper
100 set:
32 86 57 99 20 84 28 80 52 78 9 39 54 100 47 44 27 68 54 10 8 52 0 64 21 48 80 22 77 45 9 52 28 65 73
17 25 62 46 0 49 77 49 39 16 57 93 77 32 16 35 85 41 13 44 43 96 92 21 71 84 72 7 67 11 69 39 87 64
70 4 25 9 82 84 64 73 98 29 41 69 85 70 11 83 17 87 32 25 71 62 62 15 44 65 86 24 35 3 96

1000 set:
39 57 35 191 271 377 299 176 281 93 307 312 21 343 7 342 231 200 472 371 378 246 70 292 455 35 57
337 460 423 27 370 390 457 474 18 37 310 407 336 375 421 256 47 470 17 402 470 405 149 380 168
437 294 113 369 181 79 210 75 102 436 8 280 19 367 472 102 310 236 290 485 95 485 470 41 401 94
227 207 425 341 363 53 316 162 336 36 464 240 367 386 30 62 6 273 245 215 377 95 283 147 185 214
286 295 36 73 227 92 78 41 83 136 59 449 306 454 305 381 66 6 198 377 180 274 132 111 188 67 196
437 42 293 118 424 303 159 486 286 250 304 132 287 128 147 347 254 269 160 8 53 354 250 211 174
295 226 328 218 265 183 313 461 265 191 189 149 225 10 93 460 97 408 128 154 254 378 10 224 220
239 245 47 353 164 295 95 385 398 20 430 436 245 211 206 208 214 356 213 254 102 429 30 267 324
89 129 100 461 460 128 269 263 404 204 372 302 122 166 457 298 244 59 263 185 488 75 80 287 388
346 247 307 235 397 54 427 444 91 88 442 202 114 290 467 316 31 56 51 319 101 406 240 167 173
346 150 406 325 72 471 135 44 448 92 200 34 266 123 416 449 374 380 249 327 398 125 6 353 255 31
85 293 156 129 300 296 444 231 493 50 141 254 207 233 499 30 428 374 385 455 491 206 161 249 138
378 225 34 246 157 359 228 438 186 285 35 157 284 457 433 74 158 216 151 395 31 312 137 265 84
386 295 470 469 455 385 282 482 32 296 404 355 276 53 164 374 142 140 136 214 1 315 48 410 32
318 412 166 159 422 246 72 372 215 177 466 388 3 463 380 374 367 198 120 100 143 380 121 169 278
32 110 247 214 475 317 313 416 414 266 158 8 237 365 481 65 95 69 306 265 341 183 445 26 274 40
334 436 155 366 54 272 271 177 491 253 433 235 89 409 314 101 477 79 244 237 144 209 214 470 72
44 473 441 322 8 64 20 468 246 228 41 429 437 220 186 108 45 216 394 237 183 234 302 156 499 308
344 341 218 205 350 109 192 471 120 44 296 198 493 479 371 94 368 349 263 381 470 495 487 109
308 362 186 332 317 300 477 359 396 351 159 139 129 307 245 73 149 611 796 835 565 786 929 837
682 606 799 571 928 820 687 953 799 734 684 752 872 682 603 888 885 959 924 670 726 578 949 558
510 502 524 698 572 744 992 798 544 574 642 934 923 656 941 852 811 972 941 690 890 631 861 889
634 909 978 762 682 930 656 737 608 969 691 519 820 876 883 971 895 741 868 708 774 559 939 871
881 967 647 957 866 872 977 959 946 922 581 595 711 863 961 544 903 752 929 738 721 665 714 687
904 868 711 713 999 835 591 994 934 722 614 548 955 792 587 770 962 774 538 746 972 726 626 542
793 654 689 564 618 885 755 993 816 962 966 961 750 518 585 716 873 632 828 964 991 565 584 915
982 803 840 517 652 596 851 751 857 504 735 624 728 875 567 751 987 866 507 815 693 698 924 739
662 718 646 602 979 911 551 819 909 899 571 584 760 954 971 582 558 843 510 857 568 660 705 885
682 851 575 741 556 549 920 528 793 858 530 531 909 908 760 940 795 900 655 529 920 516 541 991
934 906 919 538 720 842 700 646 566 765 733 770 997 926 521 728 594 515 602 608 631 968 929 567
991 920 645 501 788 947 941 925 852 899 843 822 528 755 886 861 944 880 904 906 987 843 927 973
593 782 907 765 861 746 725 904 969 670 696 843 952 774 995 688 668 524 910 846 661 804 857 660
999 626 942 997 777 841 674 704 980 588 711 850 843 896 731 555 958 992 937 676 546 507 727 578
974 893 837 740 570 607 679 727 943 791 784 703 681 541 509 810 733 588 820 562 608 697 868 875
692 814 523 518 689 666 691 707 792 694 842 696 950 947 987 915 647 930 779 752 830 988 508 602
961 742 937 835 753 713 824 977 840 813 677 647 590 831 942 597 532 555 615 784 770 762 737 756
553 530 941 849 544 557 724 813 502 565 777 738 933 729 510 722 760 813 805 874 523 857 572 654
676 722 731 679 921 982 600 527 702 797 911 815 975 791 656 867 764 698 934 740 569 570 866 630
521 966 881 708 720 553 772 834 717 571 585 692 764 753 797 621 510 875 571 653 853 549 662 603

```
721 620 877 967 701 841 752 714 937 590 863 984 641 969 849 558 776 913 754 582 994 767 822 983
858 946 907 828 522 568 779 508 888 706 615 710 589
```

100 set output: C#

```
0, 0, 3, 4, 7, 8, 9, 9, 9, 10, 11, 11, 13, 15, 16, 16, 17, 17, 20, 21, 21, 22, 24, 25, 25, 25, 27, 28, 28, 29, 32,
32, 32, 35, 35, 39, 39, 39, 41, 41, 43, 44, 44, 44, 45, 46, 47, 48, 49, 49, 52, 52, 52, 54, 54, 57, 57, 62, 62,
62, 64, 64, 64, 65, 65, 67, 68, 69, 69, 70, 70, 71, 71, 72, 73, 73, 77, 77, 77, 78, 80, 80, 82, 83, 84, 84, 84,
85, 85, 86, 86, 87, 87, 92, 93, 96, 96, 98, 99, 100
```

1000 set output: C#

```
1, 3, 6, 6, 6, 7, 8, 8, 8, 8, 10, 10, 17, 18, 19, 20, 20, 21, 26, 27, 30, 30, 30, 31, 31, 31, 32, 32, 32, 34, 34,
35, 35, 35, 36, 36, 37, 39, 40, 41, 41, 41, 42, 44, 44, 44, 45, 47, 47, 48, 50, 51, 53, 53, 53, 54, 54, 56, 57,
57, 59, 59, 62, 64, 65, 66, 67, 69, 70, 72, 72, 72, 73, 73, 74, 75, 75, 78, 79, 79, 80, 83, 84, 85, 88, 89, 89,
91, 92, 92, 93, 93, 94, 94, 95, 95, 95, 95, 97, 100, 100, 101, 101, 102, 102, 102, 108, 109, 109, 110, 111,
113, 114, 118, 120, 120, 121, 122, 123, 125, 128, 128, 128, 129, 129, 129, 132, 132, 135, 136, 136, 137,
138, 139, 140, 141, 142, 143, 144, 147, 147, 149, 149, 149, 150, 151, 154, 155, 156, 156, 157, 157, 158,
158, 159, 159, 159, 160, 161, 162, 164, 164, 166, 166, 167, 168, 169, 173, 174, 176, 177, 177, 180, 181,
183, 183, 183, 185, 185, 186, 186, 186, 188, 189, 191, 191, 192, 196, 198, 198, 198, 200, 200, 202, 204,
205, 206, 206, 207, 207, 208, 209, 210, 211, 211, 213, 214, 214, 214, 214, 214, 215, 215, 216, 216, 218,
218, 220, 220, 224, 225, 225, 226, 227, 227, 228, 228, 231, 231, 233, 234, 235, 235, 236, 237, 237, 237,
239, 240, 240, 244, 244, 245, 245, 245, 245, 246, 246, 246, 246, 247, 247, 249, 249, 250, 250, 253, 254,
254, 254, 254, 255, 256, 263, 263, 263, 265, 265, 265, 265, 266, 266, 267, 269, 269, 271, 271, 272, 273,
274, 274, 276, 278, 280, 281, 282, 283, 284, 285, 286, 286, 287, 287, 290, 290, 292, 293, 293, 294, 295,
295, 295, 295, 296, 296, 296, 298, 299, 300, 300, 302, 302, 303, 304, 305, 306, 306, 307, 307, 307, 308,
308, 310, 310, 312, 312, 313, 313, 314, 315, 316, 316, 317, 317, 318, 319, 322, 324, 325, 327, 328, 332,
334, 336, 336, 337, 341, 341, 341, 342, 343, 344, 346, 346, 347, 349, 350, 351, 353, 353, 354, 355, 356,
359, 359, 362, 363, 365, 366, 367, 367, 367, 368, 369, 370, 371, 371, 372, 372, 374, 374, 374, 374, 375,
377, 377, 377, 378, 378, 378, 380, 380, 380, 380, 381, 381, 385, 385, 385, 386, 386, 388, 388, 390, 394,
395, 396, 397, 398, 398, 401, 402, 404, 404, 405, 406, 406, 407, 408, 409, 410, 412, 414, 416, 416, 421,
422, 423, 424, 425, 427, 428, 429, 429, 430, 433, 433, 436, 436, 436, 437, 437, 437, 438, 441, 442, 444,
444, 445, 448, 449, 449, 454, 455, 455, 455, 457, 457, 457, 460, 460, 460, 461, 461, 463, 464, 466, 467,
468, 469, 470, 470, 470, 470, 470, 470, 471, 471, 472, 472, 473, 474, 475, 477, 477, 479, 481, 482, 485,
485, 486, 487, 488, 491, 491, 493, 493, 495, 499, 499, 501, 502, 502, 504, 507, 507, 508, 508, 509, 510,
510, 510, 510, 515, 516, 517, 518, 518, 519, 521, 521, 522, 523, 523, 524, 524, 527, 528, 528, 529, 530,
530, 531, 532, 538, 538, 541, 541, 542, 544, 544, 544, 546, 548, 549, 549, 551, 553, 553, 555, 555, 556,
557, 558, 558, 558, 559, 562, 564, 565, 565, 565, 566, 567, 567, 568, 568, 569, 570, 570, 571, 571, 571,
571, 572, 572, 574, 575, 578, 578, 581, 582, 582, 584, 584, 585, 585, 587, 588, 588, 589, 590, 590, 591,
593, 594, 595, 596, 597, 600, 602, 602, 602, 603, 603, 606, 607, 608, 608, 608, 611, 614, 615, 615, 618,
620, 621, 624, 626, 626, 630, 631, 631, 632, 634, 641, 642, 645, 646, 646, 647, 647, 647, 652, 653, 654,
654, 655, 656, 656, 656, 660, 660, 661, 662, 662, 665, 666, 668, 670, 670, 674, 676, 676, 677, 679, 679,
681, 682, 682, 682, 682, 684, 687, 687, 688, 689, 689, 690, 691, 691, 692, 692, 693, 694, 696, 696, 697,
698, 698, 698, 700, 701, 702, 703, 704, 705, 706, 707, 708, 708, 710, 711, 711, 711, 713, 713, 714, 714,
716, 717, 718, 720, 720, 721, 721, 722, 722, 722, 724, 725, 726, 726, 727, 727, 728, 728, 729, 731, 731,
733, 733, 734, 735, 737, 737, 738, 738, 739, 740, 740, 741, 741, 742, 744, 746, 746, 750, 751, 751, 752,
752, 752, 752, 753, 753, 754, 755, 755, 756, 760, 760, 760, 762, 762, 764, 764, 765, 765, 767, 770, 770,
770, 772, 774, 774, 774, 776, 777, 777, 779, 779, 782, 784, 784, 786, 788, 791, 791, 792, 792, 793, 793,
795, 796, 797, 797, 798, 799, 799, 803, 804, 805, 810, 811, 813, 813, 813, 814, 815, 815, 816, 819, 820,
```

820, 820, 822, 822, 824, 828, 828, 830, 831, 834, 835, 835, 835, 837, 837, 840, 840, 841, 841, 842, 842,
843, 843, 843, 843, 843, 846, 849, 849, 850, 851, 851, 852, 852, 853, 857, 857, 857, 857, 858, 858, 861,
861, 861, 863, 863, 866, 866, 866, 867, 868, 868, 868, 871, 872, 872, 873, 874, 875, 875, 875, 876, 877,
880, 881, 881, 883, 885, 885, 885, 886, 888, 888, 889, 890, 893, 895, 896, 899, 899, 900, 903, 904, 904,
904, 906, 906, 907, 907, 908, 909, 909, 909, 910, 911, 911, 913, 915, 915, 919, 920, 920, 920, 921, 922,
923, 924, 924, 925, 926, 927, 928, 929, 929, 929, 930, 930, 933, 934, 934, 934, 934, 937, 937, 937, 939,
940, 941, 941, 941, 941, 942, 942, 943, 944, 946, 946, 947, 947, 949, 950, 952, 953, 954, 955, 957, 958,
959, 959, 961, 961, 961, 962, 962, 964, 966, 966, 967, 967, 968, 969, 969, 969, 971, 971, 972, 972, 973,
974, 975, 977, 977, 978, 979, 980, 982, 982, 983, 984, 987, 987, 987, 988, 991, 991, 991, 992, 992, 993,
994, 994, 995, 997, 997, 999, 999

100 set output: VB

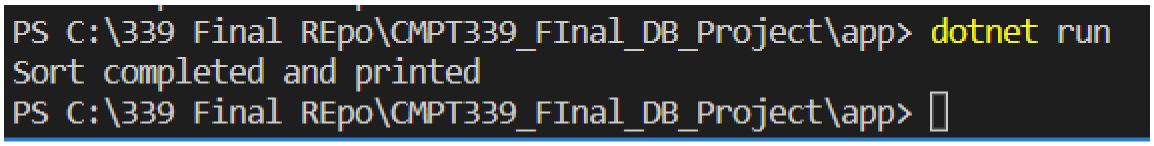
0, 0, 3, 4, 7, 8, 9, 9, 9, 10, 11, 11, 13, 15, 16, 16, 17, 17, 20, 21, 21, 22, 24, 25, 25, 25, 27, 28, 28, 29, 32,
32, 32, 35, 35, 39, 39, 39, 41, 41, 43, 44, 44, 44, 45, 46, 47, 48, 49, 49, 52, 52, 52, 54, 54, 57, 57, 62, 62,
62, 64, 64, 64, 65, 65, 67, 68, 69, 69, 70, 70, 71, 71, 72, 73, 73, 77, 77, 77, 78, 80, 80, 82, 83, 84, 84, 84,
85, 85, 86, 86, 87, 87, 92, 93, 96, 96, 98, 99, 100

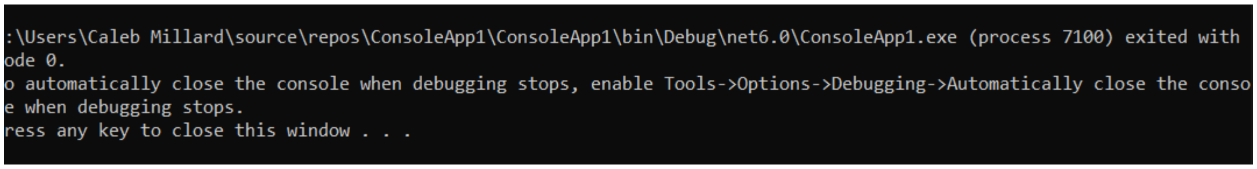
1000 set output: VB

1, 3, 6, 6, 6, 7, 8, 8, 8, 8, 10, 10, 17, 18, 19, 20, 20, 21, 26, 27, 30, 30, 30, 31, 31, 31, 32, 32, 32, 34, 34,
35, 35, 35, 36, 36, 37, 39, 40, 41, 41, 41, 42, 44, 44, 44, 45, 47, 47, 48, 50, 51, 53, 53, 53, 54, 54, 56, 57,
57, 59, 59, 62, 64, 65, 66, 67, 69, 70, 72, 72, 72, 73, 73, 74, 75, 75, 78, 79, 79, 80, 83, 84, 85, 88, 89, 89,
91, 92, 92, 93, 93, 94, 94, 95, 95, 95, 95, 97, 100, 100, 101, 101, 102, 102, 102, 108, 109, 109, 110, 111,
113, 114, 118, 120, 120, 121, 122, 123, 125, 128, 128, 128, 129, 129, 129, 132, 132, 135, 136, 136, 137,
138, 139, 140, 141, 142, 143, 144, 147, 147, 149, 149, 149, 150, 151, 154, 155, 156, 156, 157, 157, 158,
158, 159, 159, 159, 160, 161, 162, 164, 164, 166, 166, 167, 168, 169, 173, 174, 176, 177, 177, 180, 181,
183, 183, 183, 185, 185, 186, 186, 186, 188, 189, 191, 191, 192, 196, 198, 198, 198, 200, 200, 202, 204,
205, 206, 206, 207, 207, 208, 209, 210, 211, 211, 213, 214, 214, 214, 214, 214, 215, 215, 216, 216, 218,
218, 220, 220, 224, 225, 225, 226, 227, 227, 228, 228, 231, 231, 233, 234, 235, 235, 236, 237, 237, 237,
239, 240, 240, 244, 244, 245, 245, 245, 245, 246, 246, 246, 246, 247, 247, 249, 249, 250, 250, 253, 254,
254, 254, 254, 255, 256, 263, 263, 263, 265, 265, 265, 265, 266, 266, 267, 269, 269, 271, 271, 272, 273,
274, 274, 276, 278, 280, 281, 282, 283, 284, 285, 286, 286, 287, 287, 290, 290, 292, 293, 293, 294, 295,
295, 295, 295, 296, 296, 296, 298, 299, 300, 300, 302, 302, 303, 304, 305, 306, 306, 307, 307, 307, 308,
308, 310, 310, 312, 312, 313, 313, 314, 315, 316, 316, 317, 317, 318, 319, 322, 324, 325, 327, 328, 332,
334, 336, 336, 337, 341, 341, 341, 342, 343, 344, 346, 346, 347, 349, 350, 351, 353, 353, 354, 355, 356,
359, 359, 362, 363, 365, 366, 367, 367, 367, 368, 369, 370, 371, 371, 372, 372, 374, 374, 374, 374, 375,
377, 377, 377, 378, 378, 378, 380, 380, 380, 380, 381, 381, 385, 385, 385, 386, 386, 388, 388, 390, 394,
395, 396, 397, 398, 398, 401, 402, 404, 404, 405, 406, 406, 407, 408, 409, 410, 412, 414, 416, 416, 421,
422, 423, 424, 425, 427, 428, 429, 429, 430, 433, 433, 436, 436, 436, 437, 437, 437, 438, 441, 442, 444,
444, 445, 448, 449, 449, 454, 455, 455, 455, 457, 457, 457, 460, 460, 460, 461, 461, 463, 464, 466, 467,
468, 469, 470, 470, 470, 470, 470, 470, 471, 471, 472, 472, 473, 474, 475, 477, 477, 479, 481, 482, 485,
485, 486, 487, 488, 491, 491, 493, 493, 495, 499, 499, 501, 502, 502, 504, 507, 507, 508, 508, 509, 510,
510, 510, 510, 515, 516, 517, 518, 518, 519, 521, 521, 522, 523, 523, 524, 524, 527, 528, 528, 529, 530,
530, 531, 532, 538, 538, 541, 541, 542, 544, 544, 544, 546, 548, 549, 549, 551, 553, 553, 555, 555, 556,
557, 558, 558, 558, 559, 562, 564, 565, 565, 565, 566, 567, 567, 568, 568, 569, 570, 570, 571, 571, 571,

571, 572, 572, 574, 575, 578, 578, 581, 582, 582, 584, 584, 585, 585, 587, 588, 588, 589, 590, 590, 591, 593, 594, 595, 596, 597, 600, 602, 602, 602, 603, 603, 606, 607, 608, 608, 608, 611, 614, 615, 615, 618, 620, 621, 624, 626, 626, 630, 631, 631, 632, 634, 641, 642, 645, 646, 646, 647, 647, 647, 652, 653, 654, 654, 655, 656, 656, 656, 660, 660, 661, 662, 662, 665, 666, 668, 670, 670, 674, 676, 676, 677, 679, 679, 681, 682, 682, 682, 682, 684, 687, 687, 688, 689, 689, 690, 691, 691, 692, 692, 693, 694, 696, 696, 697, 698, 698, 698, 700, 701, 702, 703, 704, 705, 706, 707, 708, 708, 710, 711, 711, 711, 713, 713, 714, 714, 716, 717, 718, 720, 720, 721, 721, 722, 722, 722, 724, 725, 726, 726, 727, 727, 728, 728, 729, 731, 731, 733, 733, 734, 735, 737, 737, 738, 738, 739, 740, 740, 741, 741, 742, 744, 746, 746, 750, 751, 751, 752, 752, 752, 752, 753, 753, 754, 755, 755, 756, 760, 760, 760, 762, 762, 764, 764, 765, 765, 767, 770, 770, 770, 772, 774, 774, 774, 776, 777, 777, 779, 779, 782, 784, 784, 786, 788, 791, 791, 792, 792, 793, 793, 795, 796, 797, 797, 798, 799, 799, 803, 804, 805, 810, 811, 813, 813, 813, 814, 815, 815, 816, 819, 820, 820, 820, 822, 822, 824, 828, 828, 830, 831, 834, 835, 835, 835, 837, 837, 840, 840, 841, 841, 842, 842, 843, 843, 843, 843, 843, 846, 849, 849, 850, 851, 851, 852, 852, 853, 857, 857, 857, 857, 858, 858, 861, 861, 861, 863, 863, 866, 866, 866, 867, 868, 868, 868, 871, 872, 872, 873, 874, 875, 875, 875, 876, 877, 880, 881, 881, 883, 885, 885, 885, 886, 888, 888, 889, 890, 893, 895, 896, 899, 899, 900, 903, 904, 904, 904, 906, 906, 907, 907, 908, 909, 909, 909, 910, 911, 911, 913, 915, 915, 919, 920, 920, 920, 921, 922, 923, 924, 924, 925, 926, 927, 928, 929, 929, 929, 930, 930, 933, 934, 934, 934, 934, 937, 937, 937, 939, 940, 941, 941, 941, 941, 942, 942, 943, 944, 946, 946, 947, 947, 949, 950, 952, 953, 954, 955, 957, 958, 959, 959, 961, 961, 961, 962, 962, 964, 966, 966, 967, 967, 968, 969, 969, 969, 971, 971, 972, 972, 973, 974, 975, 977, 977, 978, 979, 980, 982, 982, 983, 984, 987, 987, 987, 988, 991, 991, 991, 992, 992, 993, 994, 994, 995, 997, 997, 999, 999

Sample line of 10000:
109 192 471 120 44 296 198 493 479 371 94 368 349 263 381 470 495 487 109 308 362 186 332 317 300 477 359 396 351 159 139 129 307 245 73 149 611 796 835 565 786 929 837 682 606 799 571 928 820 687 953 799 734 684 752 872 682 603 888 885 959 924 670 726 578 949 558 510 502 524 698 572 744 992 798 544
Sample output:
1, 3, 6, 6, 6, 7, 8, 8, 8, 8, 10, 10, 17, 18, 19, 20, 20, 21, 26, 27, 30, 30, 30, 31, 31, 31, 32, 32, 32, 34, 34, 35, 35, 35, 36, 36, 37, 39, 40, 41, 41, 41, 42, 44, 44, 44, 45, 47, 47, 48, 50, 51, 53, 53, 53, 54, 54, 56, 57, 57, 59, 59, 62, 64, 65, 66, 67, 69, 70, 72, 72, 72, 73, 73

Screenshots:
C#

```
PS C:\339 Final REpo\CMPT339_FInal_DB_Project\app> dotnet run
Sort completed and printed
PS C:\339 Final REpo\CMPT339_FInal_DB_Project\app>
```
VB:



```
:\Users\Caleb Millard\source\repos\ConsoleApp1\ConsoleApp1\bin\Debug\net6.0\ConsoleApp1.exe (process 7100) exited with
ode 0.
o automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
e when debugging stops.
ress any key to close this window . . .
```

Conclusions:

I chose to use the quicksort algorithm because it has an efficiency of O(n log n) which is in the average case one of the best sorting algorithms to use. Making the quick sort algorithm was easy for C# since I already do that semi-regularly with game data, but when it came to inputting and outputting a text file, I really struggled. When it came to visual basic, it was really similar to C# since they are both made by Microsoft; the main issue I had was how they instantiate arrays was an issue since they start arrays instantiation from one, so if you create an array 500, then it creates an array 501 long. Which created a bunch of issues. After I fixed this, it was pretty smooth sailing. The main way I would implement the allowance of strings being sorted alphabetically would be creating a dictionary and sorting by the dictionary value of characters from the dictionary rather than trying to sort alphabetically since most languages don't allow < or > operands on characters.