Ramzy Izza Wardhana – 472698 – CSB – Lab Programming Assignment 9

Program 1

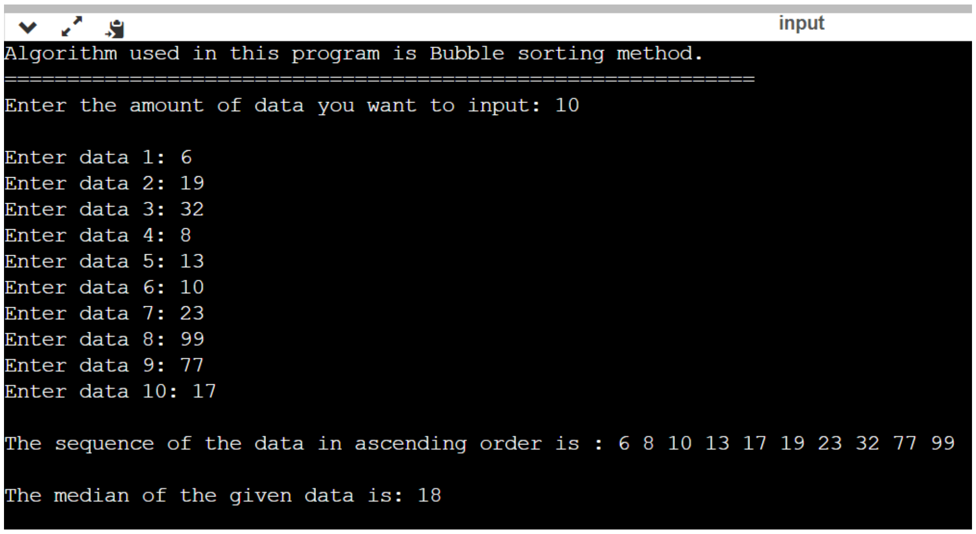
1. (20 points) Create a program to compute the median of an array consisting of N integers!

Source Code :

```
#include <iostream>
#include <iomanip>
using namespace std;

//declaration of variable
int sizeArray,x;

int main(){

   cout << "Program to receive N data from user then sort it and find the median" << endl;
   cout << "Algorithm used in this program is Bubble sorting method. " << endl;
   cout << "=============================================================" << endl;
   //user detemine the size of an array
   cout << "Enter the amount of data you want to input: ";
   cin >> sizeArray;
   float array[sizeArray]; //array declaration with N size
   cout << endl;
   //receive N amount of input from user and assign to array
   for (int i = 0; i < sizeArray; i++){
      cout << "Enter data " << i+1 << ": ";
      cin >> array[i];
   }
   //use the bubble sorting algorithm to sort the data
   for (int j = 0; j < sizeArray-1; j++){
      for(int k = 0; k <sizeArray-1;k++){
         if(array[k] > array[k+1]){
            //swab the value if array[n] > array[n+1]
           x = array[k];
           array[k] = array[k+1];
           array[k+1] = x;
           }
      }
   }
   //display the sorted sequence of data (bubble sort algorithm)
   cout << endl;
   cout << "The sequence of the data in ascending order is : ";
```

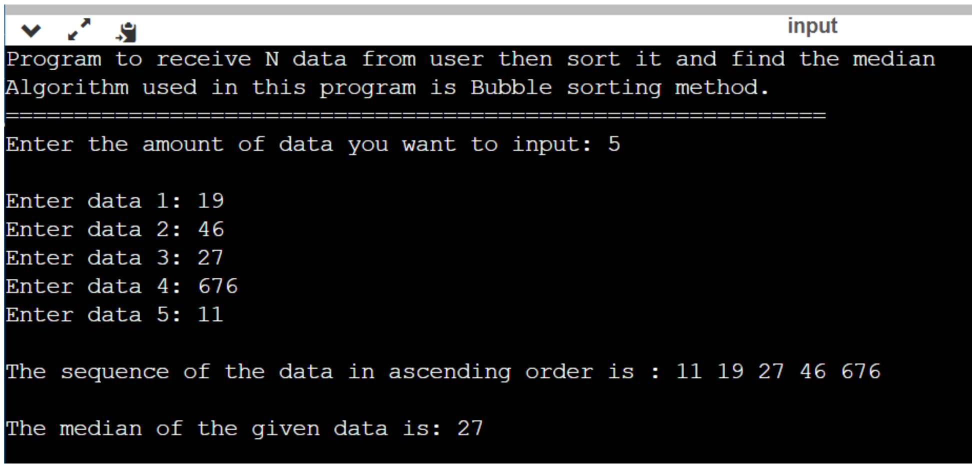
```
    for (int p = 0; p < sizeArray; p++){
        cout << array[p] << " ";
    }
    cout << endl << endl;
    //display the median of data by accessing the mid value of an array
    int mid1, mid2;
    if (sizeArray % 2 == 0){
        mid1 = (sizeArray/2)-1;
        mid2 = (sizeArray/2);
        float total = (array[mid1]+array[mid2])/2;
        cout << "The median of the given data is: " << setprecision(3) << float(total) << endl;
    }
    else if(sizeArray % 2 == 1){

        int mid = (sizeArray/2);
        cout << "The median of the given data is: " << array[mid] << endl;
    }
    return 0;
}
```

Input & Output :



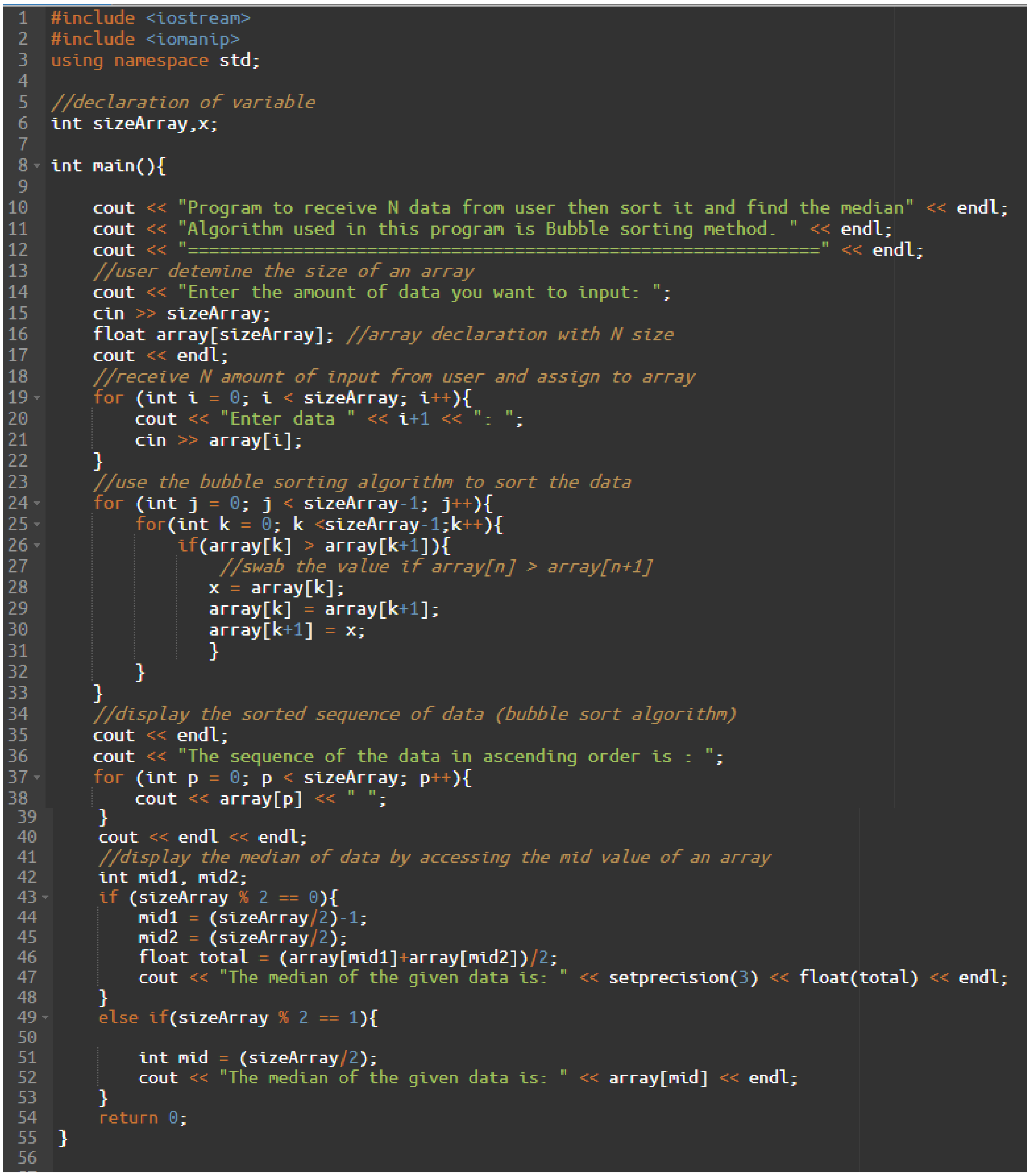
input

```
Algorithm used in this program is Bubble sorting method.
============================================================
Enter the amount of data you want to input: 10

Enter data 1: 6
Enter data 2: 19
Enter data 3: 32
Enter data 4: 8
Enter data 5: 13
Enter data 6: 10
Enter data 7: 23
Enter data 8: 99
Enter data 9: 77
Enter data 10: 17

The sequence of the data in ascending order is : 6 8 10 13 17 19 23 32 77 99

The median of the given data is: 18
```



input

```
Program to receive N data from user then sort it and find the median
Algorithm used in this program is Bubble sorting method.
============================================================
Enter the amount of data you want to input: 5

Enter data 1: 19
Enter data 2: 46
Enter data 3: 27
Enter data 4: 676
Enter data 5: 11

The sequence of the data in ascending order is : 11 19 27 46 676

The median of the given data is: 27
```



```cpp
#include <iostream>
#include <iomanip>
using namespace std;

//declaration of variable
int sizeArray,x;

int main(){

    cout << "Program to receive N data from user then sort it and find the median" << endl;
    cout << "Algorithm used in this program is Bubble sorting method. " << endl;
    cout << "===========================================================" << endl;
    //user detemine the size of an array
    cout << "Enter the amount of data you want to input: ";
    cin >> sizeArray;
    float array[sizeArray]; //array declaration with N size
    cout << endl;
    //receive N amount of input from user and assign to array
    for (int i = 0; i < sizeArray; i++){
        cout << "Enter data " << i+1 << ": ";
        cin >> array[i];
    }
    //use the bubble sorting algorithm to sort the data
    for (int j = 0; j < sizeArray-1; j++){
        for(int k = 0; k <sizeArray-1;k++){
            if(array[k] > array[k+1]){
                //swab the value if array[n] > array[n+1]
                x = array[k];
                array[k] = array[k+1];
                array[k+1] = x;
                }
        }
    }
    //display the sorted sequence of data (bubble sort algorithm)
    cout << endl;
    cout << "The sequence of the data in ascending order is : ";
    for (int p = 0; p < sizeArray; p++){
        cout << array[p] << " ";
    }
    cout << endl << endl;
    //display the median of data by accessing the mid value of an array
    int mid1, mid2;
    if (sizeArray % 2 == 0){
        mid1 = (sizeArray/2)-1;
        mid2 = (sizeArray/2);
        float total = (array[mid1]+array[mid2])/2;
        cout << "The median of the given data is: " << setprecision(3) << float(total) << endl;
    }
    else if(sizeArray % 2 == 1){

        int mid = (sizeArray/2);
        cout << "The median of the given data is: " << array[mid] << endl;
    }
    return 0;
}

```

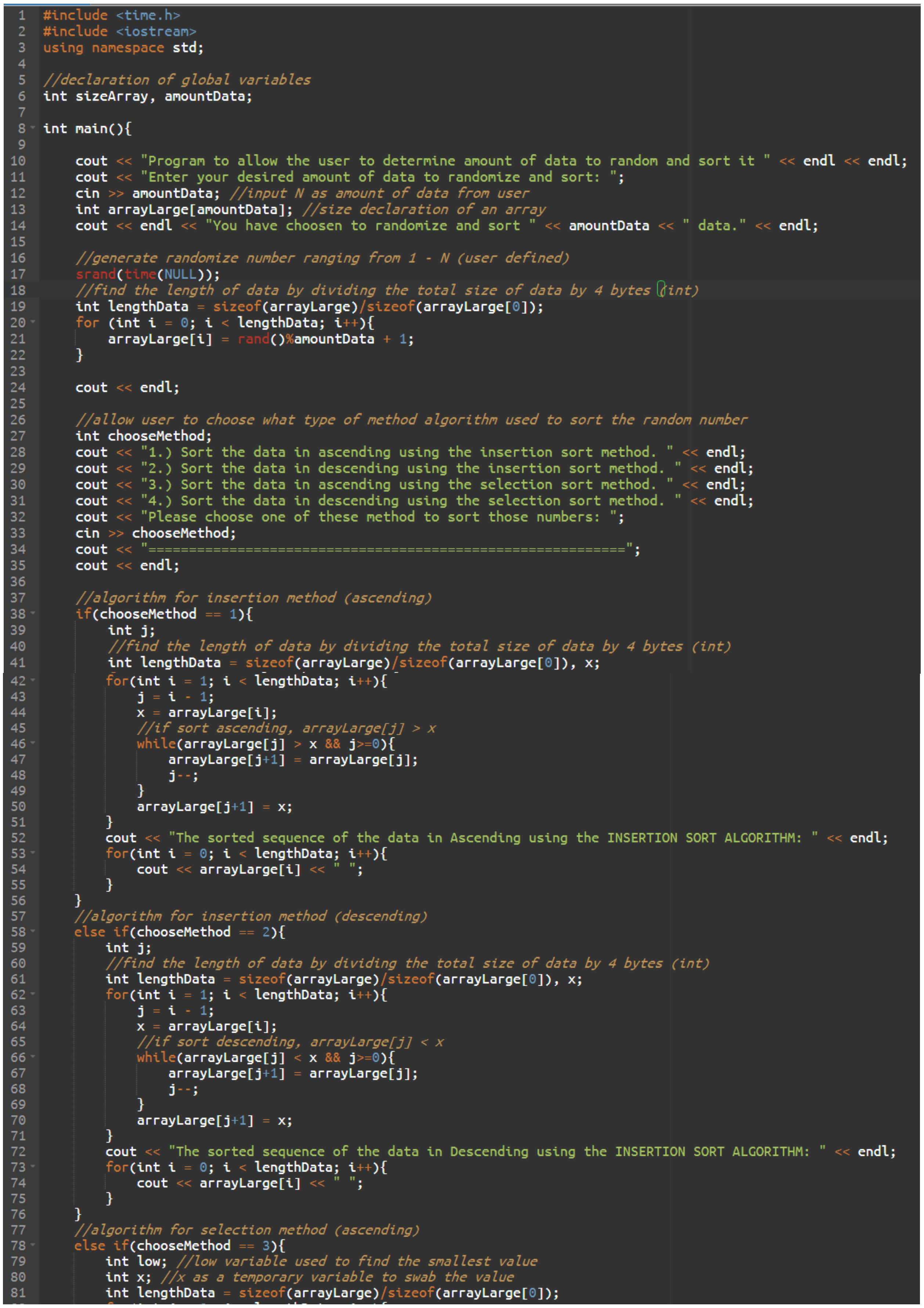
Program 2

2. (40 points) Create a program allowing the user to choose:
    a. Enter the number of data to be sorted
    b. Generate random data
    c. Sort the data in ascending using the insertion sort method
    d. Sort the data in descending using the insertion sort method
    e. Sort the data in ascending using the selection sort method
    f. Sort the data in descending using the selection sort method

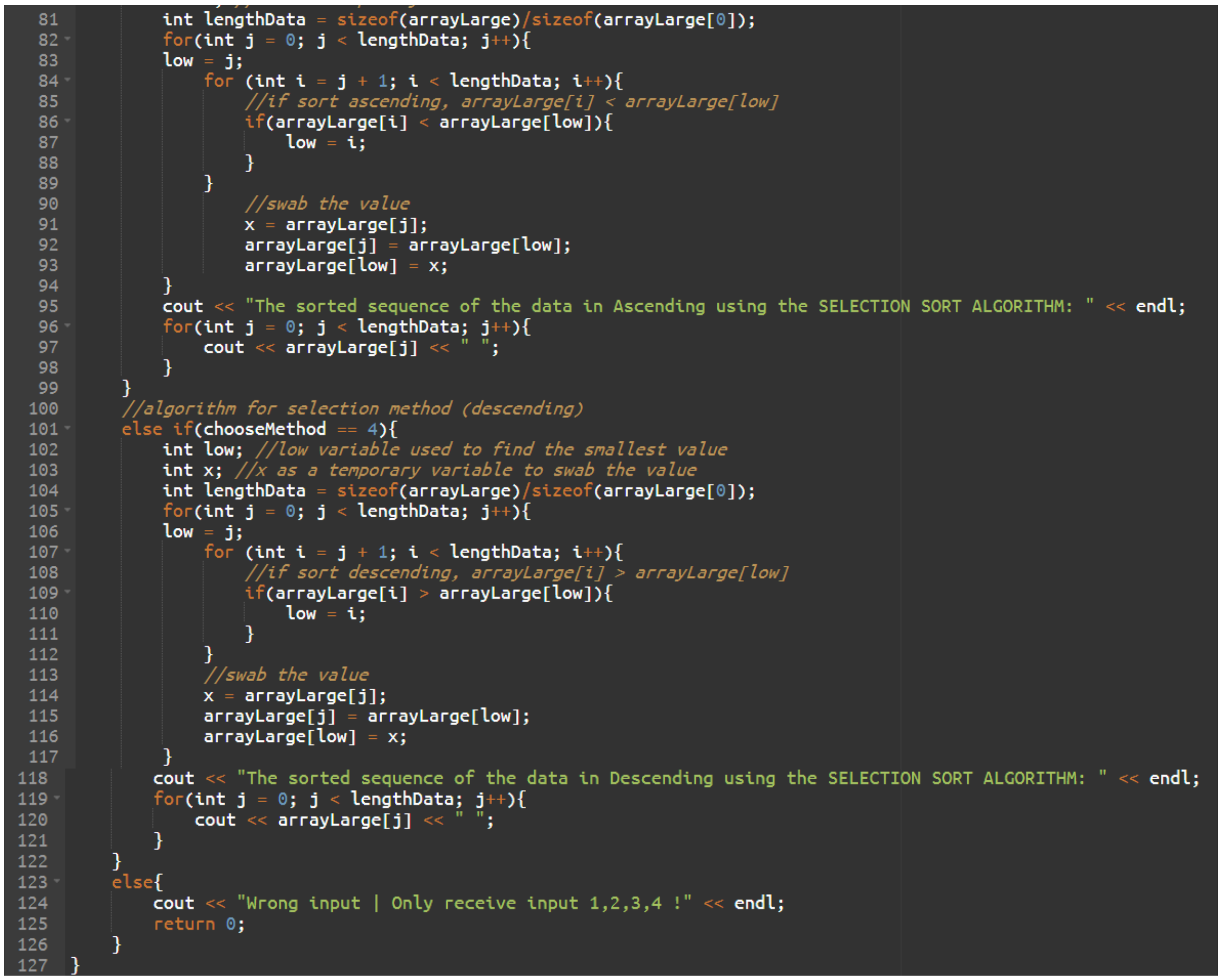
    Try with 100 data, 1000 data, 10000 data, and 100000 data. For each sort method, write the running time. Write the result in a table.

Source Code :

```
#include <time.h>
#include <iostream>
using namespace std;

//declaration of global variables
int sizeArray, amountData;

int main(){

  cout << "Program to allow the user to determine amount of data to random and sort it " <<
endl << endl;
  cout << "Enter your desired amount of data to randomize and sort: ";
  cin >> amountData; //input N as amount of data from user
  int arrayLarge[amountData]; //size declaration of an array
  cout << endl << "You have choosen to randomize and sort " << amountData << " data." <<
endl;

  //generate randomize number ranging from 1 - N (user defined)
  srand(time(NULL));
  //find the length of data by dividing the total size of data by 4 bytes (int)
  int lengthData = sizeof(arrayLarge)/sizeof(arrayLarge[0]);
  for (int i = 0; i < lengthData; i++){
    arrayLarge[i] = rand()%amountData + 1;
  }
  cout << endl;

  //allow user to choose what type of method algorithm used to sort the random number
  int chooseMethod;
  cout << "1.) Sort the data in ascending using the insertion sort method. " << endl;
  cout << "2.) Sort the data in descending using the insertion sort method. " << endl;
  cout << "3.) Sort the data in ascending using the selection sort method. " << endl;
  cout << "4.) Sort the data in descending using the selection sort method. " << endl;
  cout << "Please choose one of these method to sort those numbers: ";
  cin >> chooseMethod;
  cout << "==========================================================";
  cout << endl;

  //algorithm for insertion method (ascending)
  if(chooseMethod == 1){
    int j;
    //find the length of data by dividing the total size of data by 4 bytes (int)
    int lengthData = sizeof(arrayLarge)/sizeof(arrayLarge[0]), x;
    for(int i = 1; i < lengthData; i++){
      j = i - 1;
      x = arrayLarge[i];
      //if sort ascending, arrayLarge[j] > x
      while(arrayLarge[j] > x && j>=0){
        arrayLarge[j+1] = arrayLarge[j];
        j--;
      }
      arrayLarge[j+1] = x;
    }
    cout << "The sorted sequence of the data in Ascending using the INSERTION SORT
ALGORITHM: " << endl;
    for(int i = 0; i < lengthData; i++){
      cout << arrayLarge[i] << " ";
    }
  }
  //algorithm for insertion method (descending)
  else if(chooseMethod == 2){
    int j;
    //find the length of data by dividing the total size of data by 4 bytes (int)
    int lengthData = sizeof(arrayLarge)/sizeof(arrayLarge[0]), x;
    for(int i = 1; i < lengthData; i++){
            j = i - 1;
            x = arrayLarge[i];
            //if sort descending, arrayLarge[j] < x
            while(arrayLarge[j] < x && j>=0){
               arrayLarge[j+1] = arrayLarge[j];
               j--;
            }
            arrayLarge[j+1] = x;
         }
         cout << "The sorted sequence of the data in Descending using the INSERTION SORT
ALGORITHM: " << endl;
         for(int i = 0; i < lengthData; i++){
            cout << arrayLarge[i] << " ";
         }
      }
      //algorithm for selection method (ascending)
      else if(chooseMethod == 3){
         int low; //low variable used to find the smallest value
         int x; //x as a temporary variable to swab the value
         int lengthData = sizeof(arrayLarge)/sizeof(arrayLarge[0]);
         for(int j = 0; j < lengthData; j++){
         low = j;
            for (int i = j + 1; i < lengthData; i++){
               //if sort ascending, arrayLarge[i] < arrayLarge[low]
               if(arrayLarge[i] < arrayLarge[low]){
                  low = i;
               }
            }
               //swab the value
               x = arrayLarge[j];
               arrayLarge[j] = arrayLarge[low];
               arrayLarge[low] = x;
         }
         cout << "The sorted sequence of the data in Ascending using the SELECTION SORT
ALGORITHM: " << endl;
         for(int j = 0; j < lengthData; j++){
            cout << arrayLarge[j] << " ";
         }
      }
      //algorithm for selection method (descending)
      else if(chooseMethod == 4){
         int low; //low variable used to find the smallest value
         int x; //x as a temporary variable to swab the value
         int lengthData = sizeof(arrayLarge)/sizeof(arrayLarge[0]);
         for(int j = 0; j < lengthData; j++){
         low = j;
            for (int i = j + 1; i < lengthData; i++){
               //if sort descending, arrayLarge[i] > arrayLarge[low]
               if(arrayLarge[i] > arrayLarge[low]){
                  low = i;
               }
            }
            //swab the value
            x = arrayLarge[j];
            arrayLarge[j] = arrayLarge[low];
            arrayLarge[low] = x;
         }
         cout << "The sorted sequence of the data in Descending using the SELECTION SORT
ALGORITHM: " << endl;
         for(int j = 0; j < lengthData; j++){
            cout << arrayLarge[j] << " ";
         }
      }
      else{
         cout << "Wrong input | Only receive input 1,2,3,4 !" << endl;
         return 0;
      }
}
```



```cpp
#include <time.h>
#include <iostream>
using namespace std;

//declaration of global variables
int sizeArray, amountData;

int main(){

    cout << "Program to allow the user to determine amount of data to random and sort it " << endl << endl;
    cout << "Enter your desired amount of data to randomize and sort: ";
    cin >> amountData; //input N as amount of data from user
    int arrayLarge[amountData]; //size declaration of an array
    cout << endl << "You have choosen to randomize and sort " << amountData << " data." << endl;

    //generate randomize number ranging from 1 - N (user defined)
    srand(time(NULL));
    //find the length of data by dividing the total size of data by 4 bytes (int)
    int lengthData = sizeof(arrayLarge)/sizeof(arrayLarge[0]);
    for (int i = 0; i < lengthData; i++){
        arrayLarge[i] = rand()%amountData + 1;
    }

    cout << endl;

    //allow user to choose what type of method algorithm used to sort the random number
    int chooseMethod;
    cout << "1.) Sort the data in ascending using the insertion sort method. " << endl;
    cout << "2.) Sort the data in descending using the insertion sort method. " << endl;
    cout << "3.) Sort the data in ascending using the selection sort method. " << endl;
    cout << "4.) Sort the data in descending using the selection sort method. " << endl;
    cout << "Please choose one of these method to sort those numbers: ";
    cin >> chooseMethod;
    cout << "==========================================================";
    cout << endl;

    //algorithm for insertion method (ascending)
    if(chooseMethod == 1){
        int j;
        //find the length of data by dividing the total size of data by 4 bytes (int)
        int lengthData = sizeof(arrayLarge)/sizeof(arrayLarge[0]), x;
        for(int i = 1; i < lengthData; i++){
            j = i - 1;
            x = arrayLarge[i];
            //if sort ascending, arrayLarge[j] > x
            while(arrayLarge[j] > x && j>=0){
                arrayLarge[j+1] = arrayLarge[j];
                j--;
            }
            arrayLarge[j+1] = x;
        }
        cout << "The sorted sequence of the data in Ascending using the INSERTION SORT ALGORITHM: " << endl;
        for(int i = 0; i < lengthData; i++){
            cout << arrayLarge[i] << " ";
        }
    }
    //algorithm for insertion method (descending)
    else if(chooseMethod == 2){
        int j;
        //find the length of data by dividing the total size of data by 4 bytes (int)
        int lengthData = sizeof(arrayLarge)/sizeof(arrayLarge[0]), x;
        for(int i = 1; i < lengthData; i++){
            j = i - 1;
            x = arrayLarge[i];
            //if sort descending, arrayLarge[j] < x
            while(arrayLarge[j] < x && j>=0){
                arrayLarge[j+1] = arrayLarge[j];
                j--;
            }
            arrayLarge[j+1] = x;
        }
        cout << "The sorted sequence of the data in Descending using the INSERTION SORT ALGORITHM: " << endl;
        for(int i = 0; i < lengthData; i++){
            cout << arrayLarge[i] << " ";
        }
    }
    //algorithm for selection method (ascending)
    else if(chooseMethod == 3){
        int low; //low variable used to find the smallest value
        int x; //x as a temporary variable to swab the value
        int lengthData = sizeof(arrayLarge)/sizeof(arrayLarge[0]);
```



```
        int lengthData = sizeof(arrayLarge)/sizeof(arrayLarge[0]);
        for(int j = 0; j < lengthData; j++){
        low = j;
            for (int i = j + 1; i < lengthData; i++){
                //if sort ascending, arrayLarge[i] < arrayLarge[low]
                if(arrayLarge[i] < arrayLarge[low]){
                    low = i;
                }
            }
                //swab the value
                x = arrayLarge[j];
                arrayLarge[j] = arrayLarge[low];
                arrayLarge[low] = x;
        }
        cout << "The sorted sequence of the data in Ascending using the SELECTION SORT ALGORITHM: " << endl;
        for(int j = 0; j < lengthData; j++){
            cout << arrayLarge[j] << " ";
        }
    }
    //algorithm for selection method (descending)
    else if(chooseMethod == 4){
        int low; //low variable used to find the smallest value
        int x; //x as a temporary variable to swab the value
        int lengthData = sizeof(arrayLarge)/sizeof(arrayLarge[0]);
        for(int j = 0; j < lengthData; j++){
        low = j;
            for (int i = j + 1; i < lengthData; i++){
                //if sort descending, arrayLarge[i] > arrayLarge[low]
                if(arrayLarge[i] > arrayLarge[low]){
                    low = i;
                }
            }
            //swab the value
            x = arrayLarge[j];
            arrayLarge[j] = arrayLarge[low];
            arrayLarge[low] = x;
        }
        cout << "The sorted sequence of the data in Descending using the SELECTION SORT ALGORITHM: " << endl;
        for(int j = 0; j < lengthData; j++){
            cout << arrayLarge[j] << " ";
        }
    }
    else{
        cout << "Wrong input | Only receive input 1,2,3,4 !" << endl;
        return 0;
    }
}
```

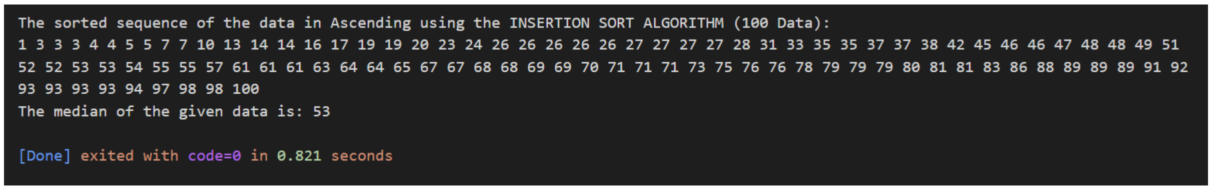
Program Runtime Result

Amount of Data	Insertion Sort Algorithm (Ascending)	Selection Sort Algorithm (Ascending)	Insertion Sort Algorithm (Descending)	Selection Sort Algorithm (Descending)
100 Data	0,821 Seconds	0,909 Seconds	0,946 Seconds	1,076 Seconds
1.000 Data	0,986 Seconds	1,074 Seconds	1,06 Seconds	1,096 Seconds
10.000 Data	1,235 Seconds	1,305 Seconds	1,335 Seconds	1,536 Seconds
100.000 Data	10,942 Seconds	15,369 Seconds	11,423 Seconds	18,329 Seconds

Input & Output

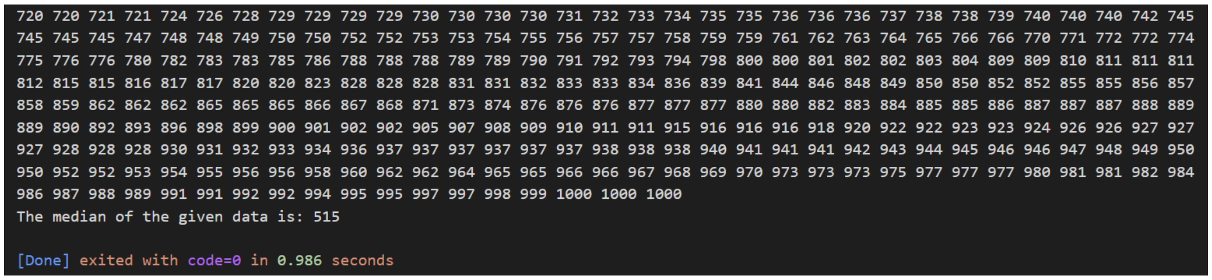
Sorted in Ascending Order [Insertion]

100 Data *Ascending* **(Insertion Sort)**



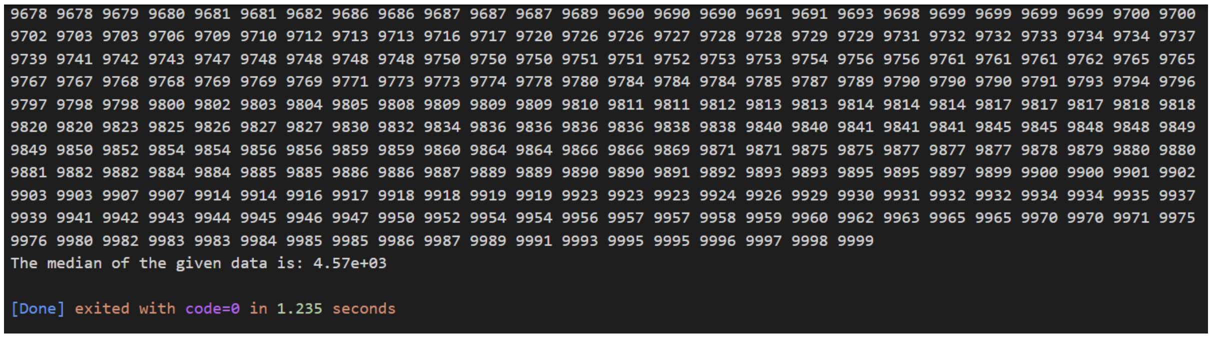
```
The sorted sequence of the data in Ascending using the INSERTION SORT ALGORITHM (100 Data):
1 3 3 3 4 4 5 5 7 7 10 13 14 14 16 17 19 19 20 23 24 26 26 26 26 26 27 27 27 27 28 31 33 35 35 37 37 38 42 45 46 46 47 48 48 49 51
52 52 53 53 54 55 55 57 61 61 61 63 64 64 65 67 67 68 68 69 69 70 71 71 71 73 75 76 76 78 79 79 79 80 81 81 83 86 88 89 89 89 91 92
93 93 93 93 94 97 98 98 100
The median of the given data is: 53

[Done] exited with code=0 in 0.821 seconds
```

1.000 Data *Ascending* **(Insertion Sort)**



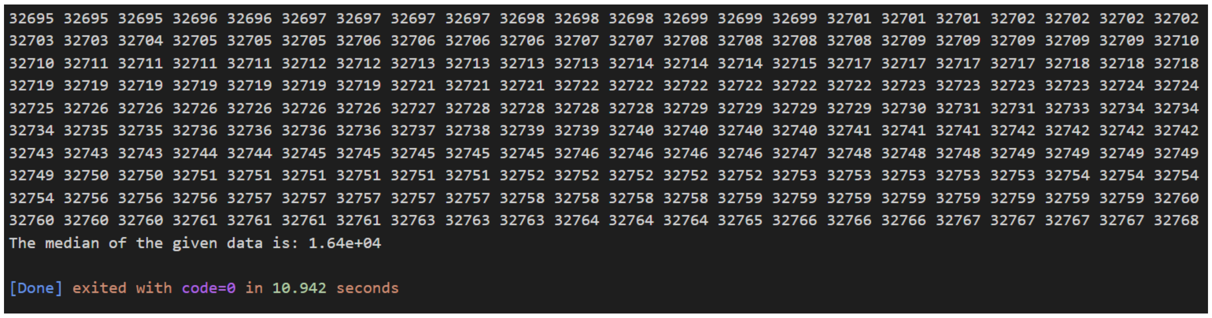
```
720 720 721 721 724 726 728 729 729 729 729 730 730 730 730 731 732 733 734 735 735 736 736 736 737 738 738 739 740 740 740 742 745
745 745 745 747 748 748 749 750 750 752 752 753 753 754 755 756 757 757 758 759 759 761 762 763 764 765 766 766 770 771 772 772 774
775 776 776 780 782 783 783 785 786 788 788 788 789 789 790 791 792 793 794 798 800 800 801 802 802 803 804 809 809 810 811 811 811
812 815 815 816 817 817 820 820 823 828 828 828 831 831 832 833 833 834 836 839 841 844 846 848 849 850 850 852 852 855 855 856 857
858 859 862 862 862 865 865 865 866 867 868 871 873 874 876 876 876 877 877 877 880 880 882 883 884 885 885 886 887 887 887 888 889
889 890 892 893 896 898 899 900 901 902 902 905 907 908 909 910 911 911 915 916 916 916 918 920 922 922 923 923 924 926 926 927 927
927 928 928 928 930 931 932 933 934 936 937 937 937 937 937 937 938 938 938 940 941 941 941 942 943 944 945 946 946 947 948 949 950
950 952 952 953 954 955 956 956 958 960 962 962 964 965 965 966 966 967 968 969 970 973 973 973 975 977 977 977 980 981 981 982 984
986 987 988 989 991 991 992 992 994 995 995 997 997 998 999 1000 1000 1000
The median of the given data is: 515

[Done] exited with code=0 in 0.986 seconds
```

10000 Data *Ascending* **(Insertion Sort)**



```
9678 9678 9679 9680 9681 9681 9682 9686 9686 9687 9687 9687 9689 9690 9690 9690 9691 9691 9693 9698 9699 9699 9699 9699 9700 9700
9702 9703 9703 9706 9709 9710 9712 9713 9713 9716 9717 9720 9726 9726 9727 9728 9728 9729 9729 9731 9732 9732 9733 9734 9734 9737
9739 9741 9742 9743 9747 9748 9748 9748 9748 9750 9750 9750 9751 9751 9752 9753 9753 9754 9756 9756 9761 9761 9761 9762 9765 9765
9767 9767 9768 9768 9769 9769 9769 9771 9773 9773 9774 9778 9780 9784 9784 9784 9785 9787 9789 9790 9790 9790 9791 9793 9794 9796
9797 9798 9798 9800 9802 9803 9804 9805 9808 9809 9809 9809 9810 9811 9811 9812 9813 9813 9814 9814 9814 9817 9817 9817 9818 9818
9820 9820 9823 9825 9826 9827 9827 9830 9832 9834 9836 9836 9836 9836 9838 9838 9840 9840 9841 9841 9841 9845 9845 9848 9848 9849
9849 9850 9852 9854 9854 9856 9856 9859 9859 9860 9864 9864 9866 9866 9869 9871 9871 9875 9875 9877 9877 9877 9878 9879 9880 9880
9881 9882 9882 9884 9884 9885 9885 9886 9886 9887 9889 9889 9890 9890 9891 9892 9893 9893 9895 9895 9897 9899 9900 9900 9901 9902
9903 9903 9907 9907 9914 9914 9916 9917 9918 9918 9919 9919 9923 9923 9923 9924 9926 9929 9930 9931 9932 9932 9934 9934 9935 9937
9939 9941 9942 9943 9944 9945 9946 9947 9950 9952 9954 9954 9956 9957 9957 9958 9959 9960 9962 9963 9965 9965 9970 9970 9971 9975
9976 9980 9982 9983 9983 9984 9985 9985 9986 9987 9989 9991 9993 9995 9995 9996 9997 9998 9999
The median of the given data is: 4.57e+03

[Done] exited with code=0 in 1.235 seconds
```

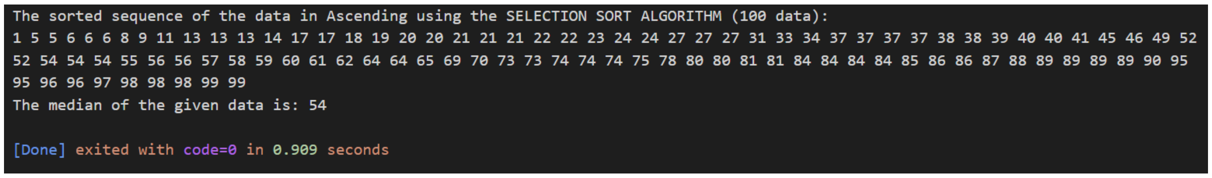
100.000 Data *Ascending* **(Insertion Sort)**



```
32695 32695 32695 32696 32696 32697 32697 32697 32697 32698 32698 32698 32699 32699 32699 32701 32701 32701 32702 32702 32702 32702
32703 32703 32704 32705 32705 32705 32706 32706 32706 32706 32707 32707 32708 32708 32708 32708 32709 32709 32709 32709 32709 32710
32710 32711 32711 32711 32711 32712 32712 32713 32713 32713 32713 32714 32714 32714 32715 32717 32717 32717 32717 32718 32718 32718
32719 32719 32719 32719 32719 32719 32719 32721 32721 32721 32722 32722 32722 32722 32722 32722 32723 32723 32723 32723 32724 32724
32725 32726 32726 32726 32726 32726 32726 32727 32728 32728 32728 32728 32729 32729 32729 32729 32730 32731 32731 32733 32734 32734
32734 32735 32735 32736 32736 32736 32736 32737 32738 32739 32739 32740 32740 32740 32740 32741 32741 32741 32742 32742 32742 32742
32743 32743 32743 32744 32744 32745 32745 32745 32745 32745 32746 32746 32746 32746 32747 32748 32748 32748 32749 32749 32749 32749
32749 32750 32750 32751 32751 32751 32751 32751 32751 32752 32752 32752 32752 32752 32753 32753 32753 32753 32753 32754 32754 32754
32754 32756 32756 32756 32757 32757 32757 32757 32757 32758 32758 32758 32758 32759 32759 32759 32759 32759 32759 32759 32759 32760
32760 32760 32760 32761 32761 32761 32761 32763 32763 32763 32764 32764 32764 32765 32766 32766 32766 32767 32767 32767 32767 32768
The median of the given data is: 1.64e+04

[Done] exited with code=0 in 10.942 seconds
```

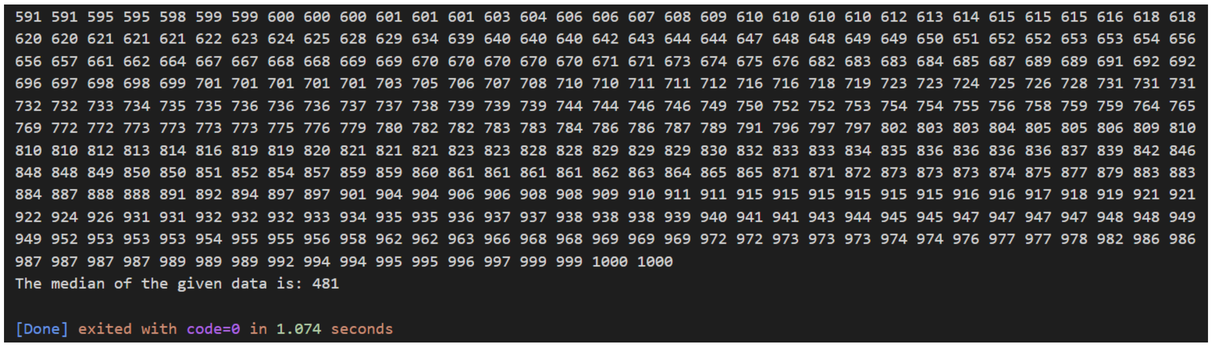
Sorted in Ascending Order [Selection]

100 Data *Ascending* **(Selection Sort)**



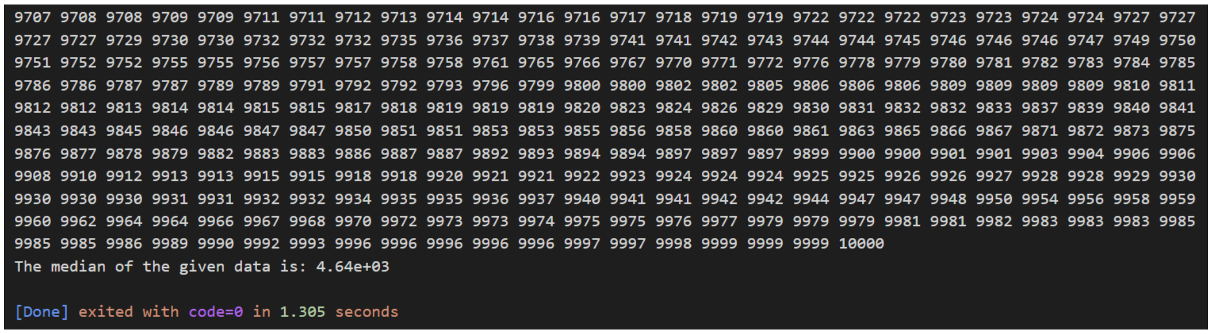
```
The sorted sequence of the data in Ascending using the SELECTION SORT ALGORITHM (100 data):
1 5 5 6 6 6 8 9 11 13 13 13 14 17 17 18 19 20 20 21 21 21 22 22 23 24 24 27 27 27 31 33 34 37 37 37 37 38 38 39 40 40 41 45 46 49 52
52 54 54 54 55 56 56 57 58 59 60 61 62 64 64 65 69 70 73 73 74 74 74 75 78 80 80 81 81 84 84 84 84 85 86 86 87 88 89 89 89 89 90 95
95 96 96 97 98 98 98 99 99
The median of the given data is: 54

[Done] exited with code=0 in 0.909 seconds
```

1000 Data *Ascending* **(Selection Sort)**



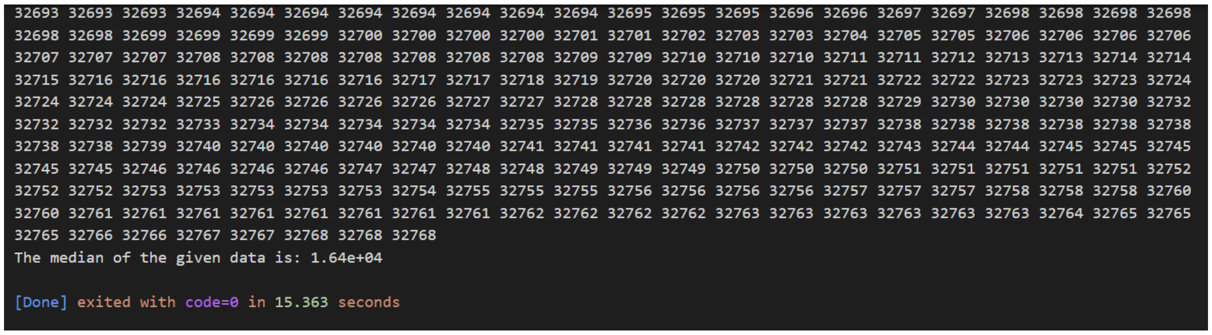
```
591 591 595 595 598 599 599 600 600 600 601 601 601 603 604 606 606 607 608 609 610 610 610 610 612 613 614 615 615 615 616 618 618
620 620 621 621 621 622 623 624 625 628 629 634 639 640 640 640 642 643 644 644 647 648 648 649 649 650 651 652 652 653 653 654 656
656 657 661 662 664 667 667 668 668 669 669 670 670 670 670 670 671 671 673 674 675 676 682 683 683 684 685 687 689 689 691 692 692
696 697 698 698 699 701 701 701 701 701 703 705 706 707 708 710 710 711 711 712 716 716 718 719 723 723 724 725 726 728 731 731 731
732 732 733 734 735 735 736 736 736 737 737 738 739 739 739 744 744 746 746 749 750 752 752 753 754 754 755 756 758 759 759 764 765
769 772 772 773 773 773 773 775 776 779 780 782 782 783 783 784 786 786 787 789 791 796 797 797 802 803 803 804 805 805 806 809 810
810 810 812 813 814 816 819 819 820 821 821 821 823 823 828 828 829 829 829 830 832 833 833 834 835 836 836 836 836 837 839 842 846
848 848 849 850 850 851 852 854 857 859 859 860 861 861 861 861 862 863 864 865 865 871 871 872 873 873 873 874 875 877 879 883 883
884 887 888 888 891 892 894 897 897 901 904 904 906 906 908 908 909 910 911 911 915 915 915 915 915 915 916 916 917 918 919 921 921
922 924 926 931 931 932 932 932 933 934 935 935 936 937 937 938 938 938 939 940 941 941 943 944 945 945 947 947 947 947 948 948 949
949 952 953 953 953 954 955 955 956 958 962 962 963 966 968 968 969 969 969 972 972 973 973 973 974 974 976 977 977 978 982 986 986
987 987 987 987 989 989 989 992 994 994 995 995 996 997 999 999 1000 1000
The median of the given data is: 481

[Done] exited with code=0 in 1.074 seconds
```

10000 Data *Ascending* **(Selection Sort)**



```
9707 9708 9708 9709 9709 9711 9711 9712 9713 9714 9714 9716 9716 9717 9718 9719 9719 9722 9722 9722 9723 9723 9724 9724 9727 9727
9727 9727 9729 9730 9730 9732 9732 9732 9735 9736 9737 9738 9739 9741 9741 9742 9743 9744 9744 9745 9746 9746 9746 9747 9749 9750
9751 9752 9752 9755 9755 9756 9757 9757 9758 9758 9761 9765 9766 9767 9770 9771 9772 9776 9778 9779 9780 9781 9782 9783 9784 9785
9786 9786 9787 9787 9789 9789 9791 9792 9792 9793 9796 9799 9800 9800 9802 9802 9805 9806 9806 9806 9809 9809 9809 9809 9810 9811
9812 9812 9813 9814 9814 9815 9815 9817 9818 9819 9819 9819 9820 9823 9824 9826 9829 9830 9831 9832 9832 9833 9837 9839 9840 9841
9843 9843 9845 9846 9846 9847 9847 9850 9851 9851 9853 9853 9855 9856 9858 9860 9860 9861 9863 9865 9866 9867 9871 9872 9873 9875
9876 9877 9878 9879 9882 9883 9883 9886 9887 9887 9892 9893 9894 9894 9897 9897 9897 9899 9900 9900 9901 9901 9903 9904 9906 9906
9908 9910 9912 9913 9913 9915 9915 9918 9918 9920 9921 9921 9922 9923 9924 9924 9924 9925 9925 9926 9926 9927 9928 9928 9929 9930
9930 9930 9930 9931 9931 9932 9932 9934 9935 9935 9936 9937 9940 9941 9941 9942 9942 9944 9947 9947 9948 9950 9954 9956 9958 9959
9960 9962 9964 9964 9966 9967 9968 9970 9972 9973 9973 9974 9975 9975 9976 9977 9979 9979 9979 9981 9981 9982 9983 9983 9983 9985
9985 9985 9986 9989 9990 9992 9993 9996 9996 9996 9996 9996 9997 9997 9998 9999 9999 9999 10000
The median of the given data is: 4.64e+03

[Done] exited with code=0 in 1.305 seconds
```

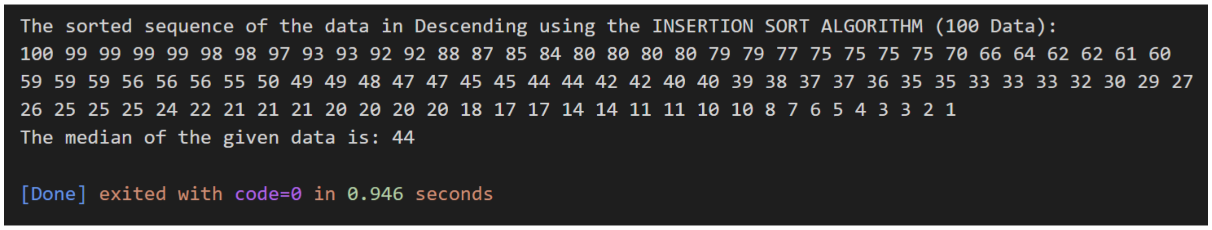
100.000 Data *Ascending* **(Selection Sort)**



```
32693 32693 32693 32694 32694 32694 32694 32694 32694 32694 32694 32695 32695 32695 32696 32696 32697 32697 32698 32698 32698 32698
32698 32698 32699 32699 32699 32699 32700 32700 32700 32700 32701 32701 32702 32703 32703 32704 32705 32705 32706 32706 32706 32706
32707 32707 32707 32708 32708 32708 32708 32708 32708 32708 32709 32709 32710 32710 32710 32711 32711 32712 32713 32713 32714 32714
32715 32716 32716 32716 32716 32716 32716 32717 32717 32718 32719 32720 32720 32720 32721 32721 32722 32722 32723 32723 32723 32724
32724 32724 32724 32725 32726 32726 32726 32726 32727 32727 32728 32728 32728 32728 32728 32728 32729 32730 32730 32730 32730 32732
32732 32732 32732 32733 32734 32734 32734 32734 32734 32735 32735 32736 32736 32737 32737 32737 32738 32738 32738 32738 32738 32738
32738 32738 32739 32740 32740 32740 32740 32740 32740 32741 32741 32741 32741 32742 32742 32742 32743 32744 32744 32745 32745 32745
32745 32745 32746 32746 32746 32746 32747 32747 32748 32748 32749 32749 32749 32750 32750 32750 32751 32751 32751 32751 32751 32752
32752 32752 32753 32753 32753 32753 32753 32754 32755 32755 32755 32756 32756 32756 32756 32757 32757 32757 32758 32758 32758 32760
32760 32761 32761 32761 32761 32761 32761 32761 32761 32762 32762 32762 32762 32763 32763 32763 32763 32763 32763 32764 32765 32765
32765 32766 32766 32767 32767 32768 32768 32768
The median of the given data is: 1.64e+04

[Done] exited with code=0 in 15.363 seconds
```

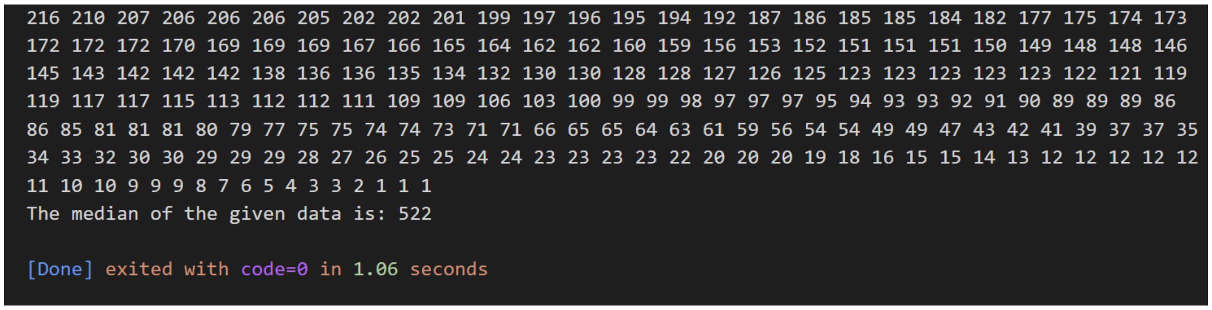
Sorted in Descending Order [Insertion]

100 Data *Descending* **(Insertion Sort)**



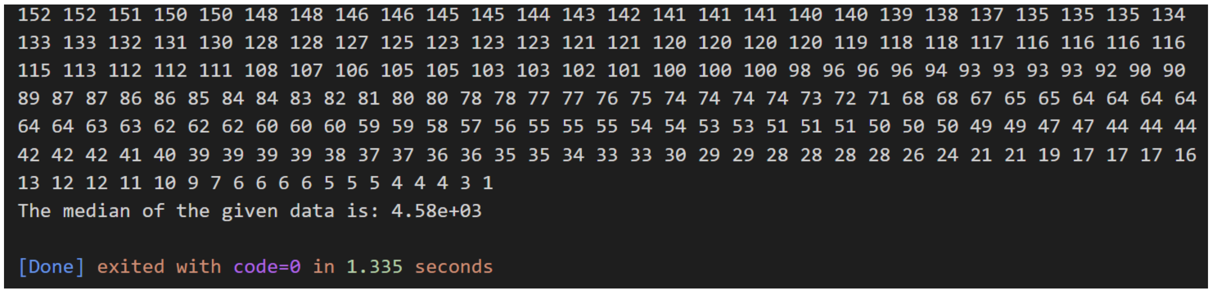
```
The sorted sequence of the data in Descending using the INSERTION SORT ALGORITHM (100 Data):
100 99 99 99 99 98 98 97 93 93 92 92 88 87 85 84 80 80 80 80 79 79 77 75 75 75 75 70 66 64 62 62 61 60
59 59 59 56 56 56 55 50 49 49 48 47 47 45 45 44 44 42 42 40 40 39 38 37 37 36 35 35 33 33 33 32 30 29 27
26 25 25 25 24 22 21 21 21 20 20 20 20 18 17 17 14 14 11 11 10 10 8 7 6 5 4 3 3 3 2 1
The median of the given data is: 44

[Done] exited with code=0 in 0.946 seconds
```

1.000 Data *Descending* **(Insertion Sort)**



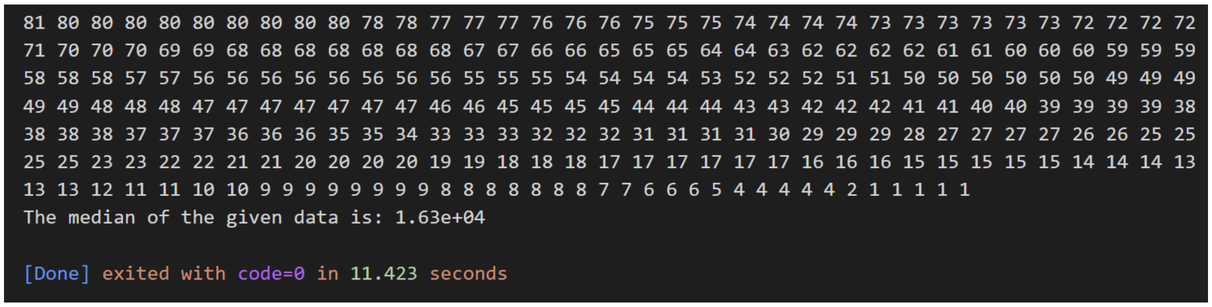
```
 216 210 207 206 206 206 205 202 202 201 199 197 196 195 194 192 187 186 185 185 184 182 177 175 174 173
 172 172 172 170 169 169 169 167 166 165 164 162 162 160 159 156 153 152 151 151 151 150 149 148 148 146
 145 143 142 142 142 138 136 136 135 134 132 130 130 128 128 127 126 125 123 123 123 123 123 122 121 119
 119 117 117 115 113 112 112 111 109 109 106 103 100 99 99 98 97 97 97 95 94 93 93 92 91 90 89 89 89 86
 86 85 81 81 81 80 79 77 75 75 74 74 73 71 71 66 65 65 64 63 61 59 56 54 54 49 49 47 43 42 41 39 37 37 35
 34 33 32 30 30 29 29 29 28 27 26 25 25 24 24 23 23 23 23 22 20 20 20 19 18 16 15 15 14 13 12 12 12 12 12
 11 10 10 9 9 9 8 7 6 5 4 3 3 2 1 1 1
 The median of the given data is: 522

 [Done] exited with code=0 in 1.06 seconds
```

10.000 Data *Descending* **(Insertion Sort)**



```
152 152 151 150 150 148 148 146 146 145 145 144 143 142 141 141 141 140 140 139 138 137 135 135 135 134
133 133 132 131 130 128 128 127 125 123 123 123 121 121 120 120 120 120 119 118 118 117 116 116 116 116
115 113 112 112 111 108 107 106 105 105 103 103 102 101 100 100 100 98 96 96 96 94 93 93 93 93 92 90 90
89 87 87 86 86 85 84 84 83 82 81 80 80 78 78 77 77 76 75 74 74 74 74 73 72 71 68 68 67 65 65 64 64 64 64
64 64 63 63 62 62 62 60 60 60 59 59 58 57 56 55 55 55 54 54 53 53 51 51 51 50 50 50 49 49 47 47 44 44 44
42 42 42 41 40 39 39 39 39 38 37 37 36 36 35 35 34 33 33 30 29 29 28 28 28 28 26 24 21 21 19 17 17 17 16
13 12 12 11 10 9 7 6 6 6 6 5 5 5 4 4 4 3 1
The median of the given data is: 4.58e+03

[Done] exited with code=0 in 1.335 seconds
```

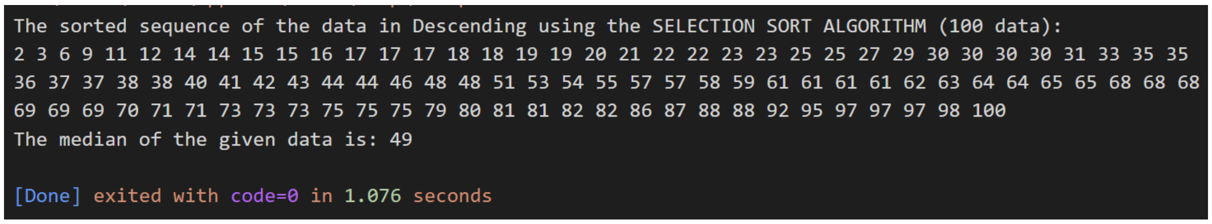
100.000 Data *Descending* **(Insertion Sort)**



```
81 80 80 80 80 80 80 80 80 80 78 78 77 77 77 76 76 76 75 75 75 74 74 74 74 73 73 73 73 73 73 72 72 72 72
71 70 70 70 69 69 68 68 68 68 68 68 68 67 67 66 66 65 65 65 64 64 63 62 62 62 62 61 61 60 60 60 59 59 59
58 58 58 57 57 56 56 56 56 56 56 56 56 55 55 55 54 54 54 54 53 52 52 52 51 51 50 50 50 50 50 50 49 49 49
49 49 48 48 48 47 47 47 47 47 47 47 46 46 45 45 45 45 44 44 44 43 43 42 42 42 41 41 40 40 39 39 39 39 38
38 38 38 37 37 37 36 36 36 35 35 34 33 33 33 32 32 32 31 31 31 31 30 29 29 29 28 27 27 27 27 26 26 25 25
25 25 23 23 22 22 21 21 20 20 20 20 19 19 18 18 18 17 17 17 17 17 17 16 16 16 15 15 15 15 15 14 14 14 13
13 13 12 11 11 10 10 9 9 9 9 9 9 9 9 8 8 8 8 8 8 8 7 7 6 6 6 5 4 4 4 4 4 2 1 1 1 1 1
The median of the given data is: 1.63e+04

[Done] exited with code=0 in 11.423 seconds
```

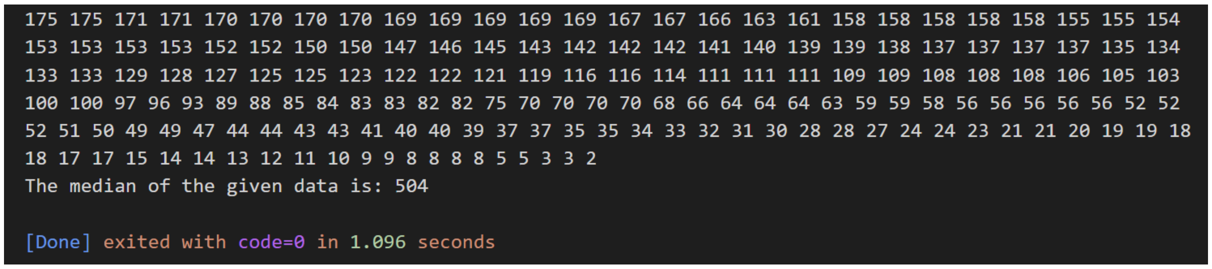
Sorted in Descending Order [Selection]

100 Data *Descending* **(Selection Sort)**



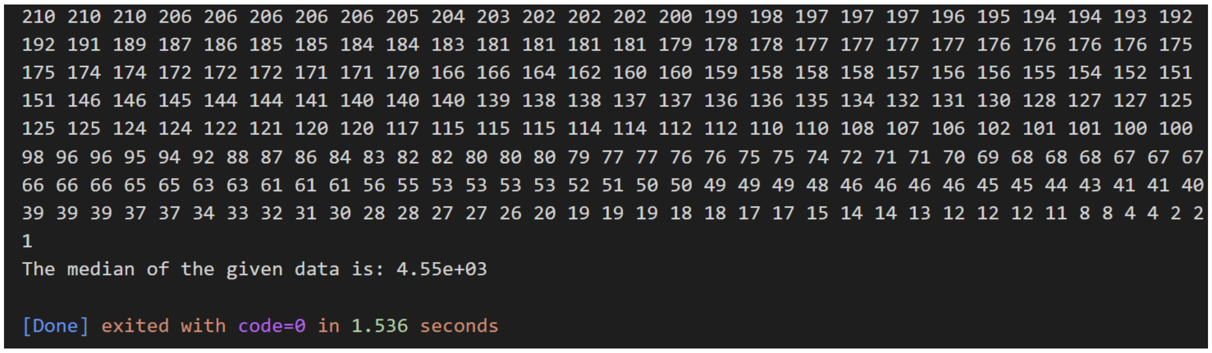
```
The sorted sequence of the data in Descending using the SELECTION SORT ALGORITHM (100 data):
2 3 6 9 11 12 14 14 15 15 16 17 17 17 18 18 19 19 20 21 22 22 23 23 25 25 27 29 30 30 30 30 31 33 35 35
36 37 37 38 38 40 41 42 43 44 44 46 48 48 51 53 54 55 57 57 58 59 61 61 61 61 62 63 64 64 65 65 68 68 68
69 69 69 70 71 71 73 73 73 75 75 75 79 80 81 81 82 82 86 87 88 88 92 95 97 97 97 98 100
The median of the given data is: 49

[Done] exited with code=0 in 1.076 seconds
```

1.000 Data *Descending* **(Selection Sort)**



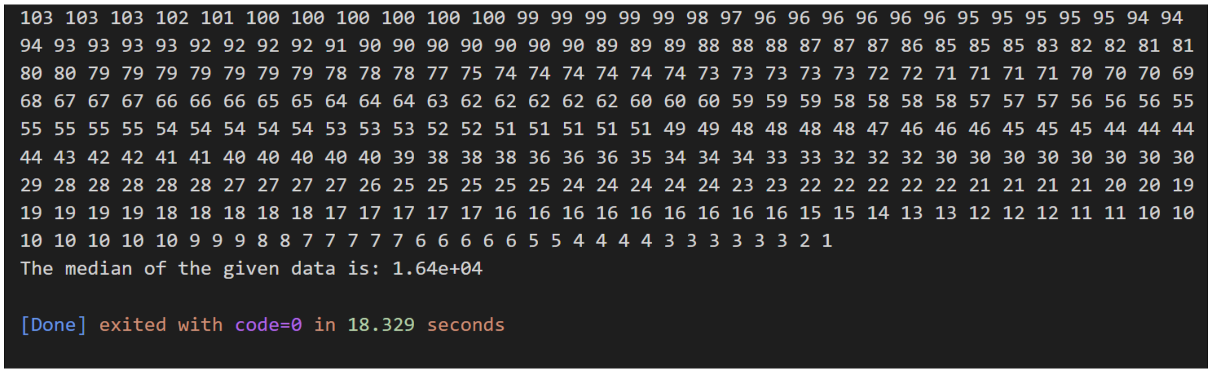
```
 175 175 171 171 170 170 170 170 169 169 169 169 169 167 167 166 163 161 158 158 158 158 158 155 155 154
 153 153 153 153 152 152 150 150 147 146 145 143 142 142 142 141 140 139 139 138 137 137 137 137 135 134
 133 133 129 128 127 125 125 123 122 122 121 119 116 116 114 111 111 111 109 109 108 108 108 106 105 103
 100 100 97 96 93 89 88 85 84 83 83 82 82 75 70 70 70 70 68 66 64 64 64 63 59 59 58 56 56 56 56 56 52 52
 52 51 50 49 49 47 44 44 43 43 41 40 40 39 37 37 35 35 34 33 32 31 30 28 28 27 24 24 23 21 21 20 19 19 18
 18 17 17 15 14 14 13 12 11 10 9 9 8 8 8 8 5 5 3 3 2
 The median of the given data is: 504

 [Done] exited with code=0 in 1.096 seconds
```

10.000 Data *Descending* **(Selection Sort)**



```
 210 210 210 206 206 206 206 206 205 204 203 202 202 202 200 199 198 197 197 197 196 195 194 194 193 192
 192 191 189 187 186 185 185 184 184 183 181 181 181 181 179 178 178 177 177 177 177 176 176 176 176 175
 175 174 174 172 172 172 171 171 170 166 166 164 162 160 160 159 158 158 158 157 156 156 155 154 152 151
 151 146 146 145 144 144 141 140 140 140 139 138 138 137 137 136 136 135 134 132 131 130 128 127 127 125
 125 125 124 124 122 121 120 120 117 115 115 115 114 114 112 112 110 110 108 107 106 102 101 101 100 100
 98 96 96 95 94 92 88 87 86 84 83 82 82 80 80 80 79 77 77 76 76 75 75 74 72 71 71 70 69 68 68 68 67 67 67
 66 66 66 65 65 63 63 61 61 61 56 55 53 53 53 53 52 51 50 50 49 49 49 48 46 46 46 46 45 45 44 43 41 41 40
 39 39 39 37 37 34 33 32 31 30 28 28 27 27 26 20 19 19 19 18 18 17 17 15 14 14 13 12 12 12 11 8 8 4 4 2 2
 1
 The median of the given data is: 4.55e+03

 [Done] exited with code=0 in 1.536 seconds
```

100.000 Data *Descending* **(Selection Sort)**



```
 103 103 103 102 101 100 100 100 100 100 100 99 99 99 99 99 98 97 96 96 96 96 96 96 95 95 95 95 95 94 94
 94 93 93 93 93 92 92 92 92 91 90 90 90 90 90 90 90 89 89 89 88 88 88 87 87 87 86 85 85 85 83 82 82 81 81
 80 80 79 79 79 79 79 79 79 78 78 78 77 75 74 74 74 74 74 74 73 73 73 73 73 72 72 71 71 71 71 70 70 70 69
 68 67 67 67 66 66 66 65 65 64 64 64 63 62 62 62 62 62 60 60 60 59 59 59 58 58 58 58 57 57 57 56 56 56 55
 55 55 55 55 54 54 54 54 54 53 53 53 52 52 51 51 51 51 51 49 49 48 48 48 48 47 46 46 46 45 45 45 44 44 44
 44 43 42 42 41 41 40 40 40 40 40 39 38 38 38 36 36 36 35 34 34 34 33 33 32 32 32 30 30 30 30 30 30 30 30
 29 28 28 28 28 28 27 27 27 27 26 25 25 25 25 25 24 24 24 24 24 23 23 22 22 22 22 22 21 21 21 21 20 20 19
 19 19 19 19 18 18 18 18 18 17 17 17 17 17 16 16 16 16 16 16 16 16 16 15 15 14 13 13 12 12 12 11 11 10 10
 10 10 10 10 10 9 9 9 8 8 7 7 7 7 7 7 6 6 6 6 6 5 5 4 4 4 4 3 3 3 3 3 3 3 2 1
 The median of the given data is: 1.64e+04

 [Done] exited with code=0 in 18.329 seconds
```

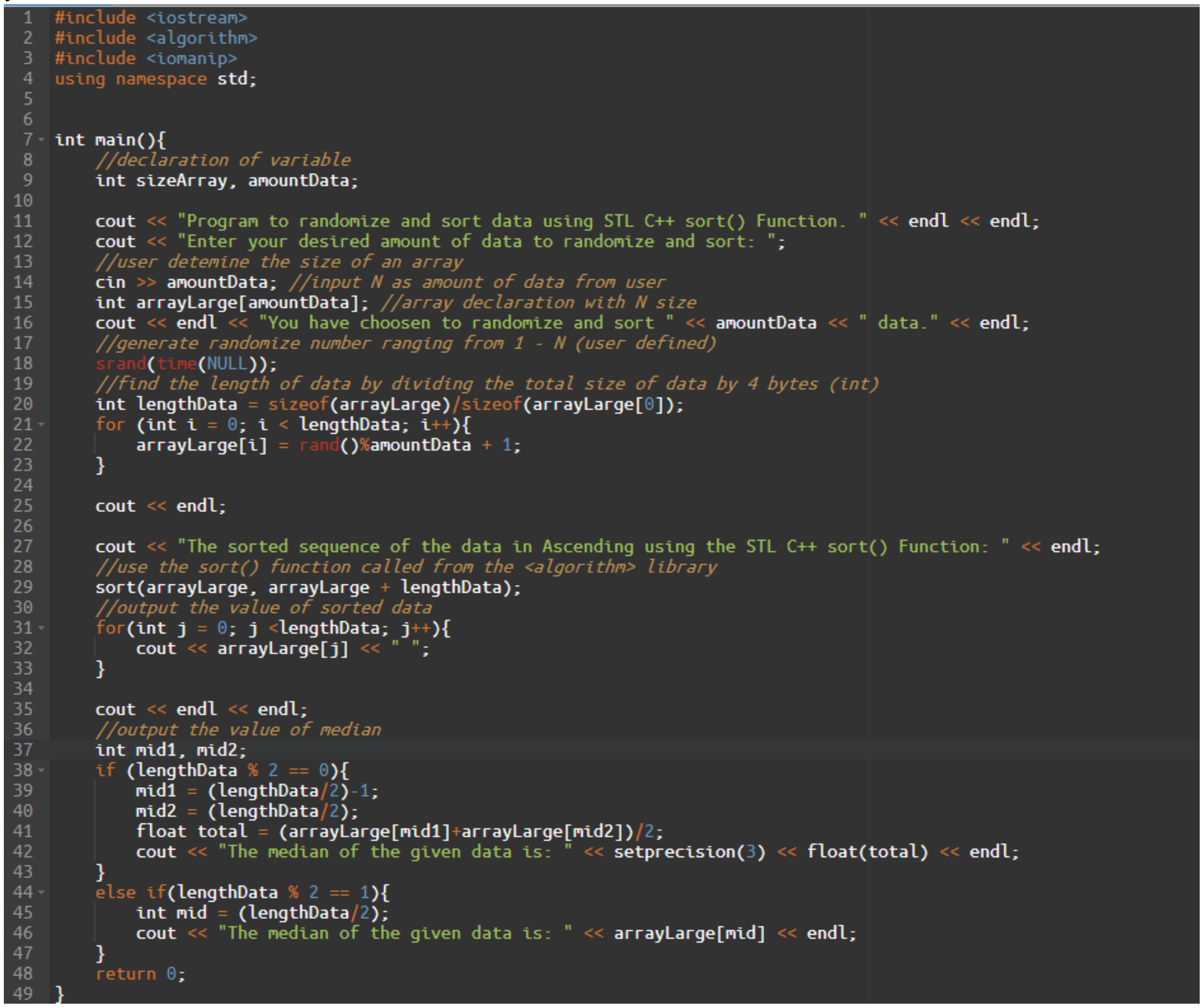
Program 3

3. Create a program to implement the STL C++ sort() function. Try with 100 data, 1000 data, 10000 data, and 100000 data. Compare the running time with the result of the 1st problem.

Source Code :

```
#include <iostream>
#include <algorithm>
#include <iomanip>
using namespace std;


int main(){
   //declaration of variable
   int sizeArray, amountData;

   cout << "Program to randomize and sort data using STL C++ sort() Function. " << endl <<
     endl;
   cout << "Enter your desired amount of data to randomize and sort: ";
   //user detemine the size of an array
   cin >> amountData; //input N as amount of data from user
   int arrayLarge[amountData]; //array declaration with N size
   cout << endl << "You have choosen to randomize and sort " << amountData << " data." <<
     endl;
   //generate randomize number ranging from 1 - N (user defined)
   srand(time(NULL));
   //find the length of data by dividing the total size of data by 4 bytes (int)
   int lengthData = sizeof(arrayLarge)/sizeof(arrayLarge[0]);
   for (int i = 0; i < lengthData; i++){
      arrayLarge[i] = rand()%amountData + 1;
   }

   cout << endl;

   cout << "The sorted sequence of the data in Ascending using the STL C++ sort() Function:
     " << endl;
   //use the sort() function called from the <algorithm> library
   sort(arrayLarge, arrayLarge + lengthData);
   //output the value of sorted data
   for(int j = 0; j <lengthData; j++){
      cout << arrayLarge[j] << " ";
   }

   cout << endl << endl;
```

```
    //output the value of median
    int mid1, mid2;
    if (lengthData % 2 == 0){
        mid1 = (lengthData/2)-1;
        mid2 = (lengthData/2);
        float total = (arrayLarge[mid1]+arrayLarge[mid2])/2;
        cout << "The median of the given data is: " << setprecision(3) << float(total) << endl;
    }
    else if(lengthData % 2 == 1){
        int mid = (lengthData/2);
        cout << "The median of the given data is: " << arrayLarge[mid] << endl;
    }
    return 0;
}
```



```
#include <iostream>
#include <algorithm>
#include <iomanip>
using namespace std;


int main(){
    //declaration of variable
    int sizeArray, amountData;

    cout << "Program to randomize and sort data using STL C++ sort() Function. " << endl << endl;
    cout << "Enter your desired amount of data to randomize and sort: ";
    //user detemine the size of an array
    cin >> amountData; //input N as amount of data from user
    int arrayLarge[amountData]; //array declaration with N size
    cout << endl << "You have choosen to randomize and sort " << amountData << " data." << endl;
    //generate randomize number ranging from 1 - N (user defined)
    srand(time(NULL));
    //find the length of data by dividing the total size of data by 4 bytes (int)
    int lengthData = sizeof(arrayLarge)/sizeof(arrayLarge[0]);
    for (int i = 0; i < lengthData; i++){
        arrayLarge[i] = rand()%amountData + 1;
    }

    cout << endl;

    cout << "The sorted sequence of the data in Ascending using the STL C++ sort() Function: " << endl;
    //use the sort() function called from the <algorithm> library
    sort(arrayLarge, arrayLarge + lengthData);
    //output the value of sorted data
    for(int j = 0; j <lengthData; j++){
        cout << arrayLarge[j] << " ";
    }

    cout << endl << endl;
    //output the value of median
    int mid1, mid2;
    if (lengthData % 2 == 0){
        mid1 = (lengthData/2)-1;
        mid2 = (lengthData/2);
        float total = (arrayLarge[mid1]+arrayLarge[mid2])/2;
        cout << "The median of the given data is: " << setprecision(3) << float(total) << endl;
    }
    else if(lengthData % 2 == 1){
        int mid = (lengthData/2);
        cout << "The median of the given data is: " << arrayLarge[mid] << endl;
    }
    return 0;
}
```

Program Runtime Result compared between problem 1 and 3

Program 3

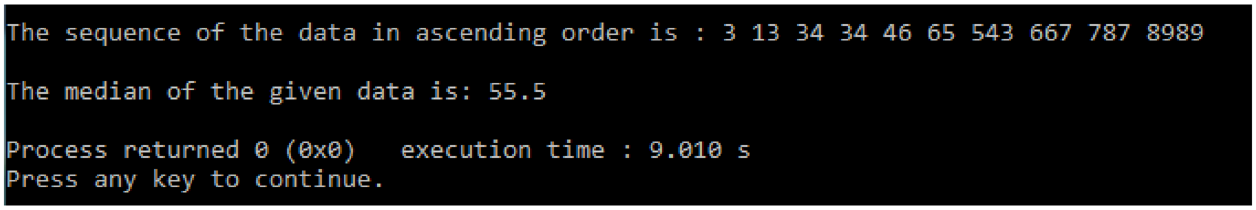
Amount of Data	STL C++ sort() Function Algorithm (Ascending)
100 Data	0,924 Seconds
1.000 Data	1,089 Seconds
10.000 Data	1,315 Seconds
100.000 Data	1,276 Seconds

Program 1

Amount of Data (User Defined)	Bubble Sort Algorithm (Ascending)
10 Data	9,010 Seconds
20 Data	18,089 Seconds
30 Data	32,381 Seconds
40 Data	46,037 Seconds

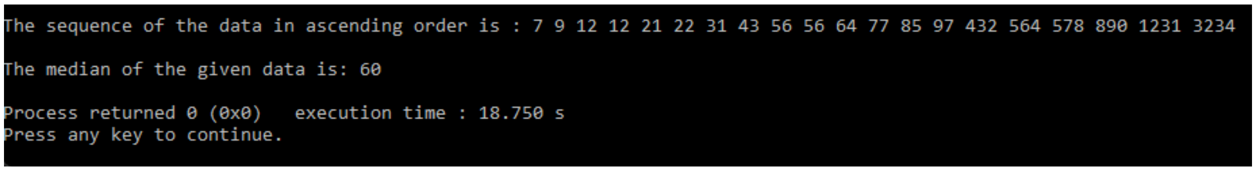
Input & Output

N = 10 Data *Ascending* **(Bubble Sort)**



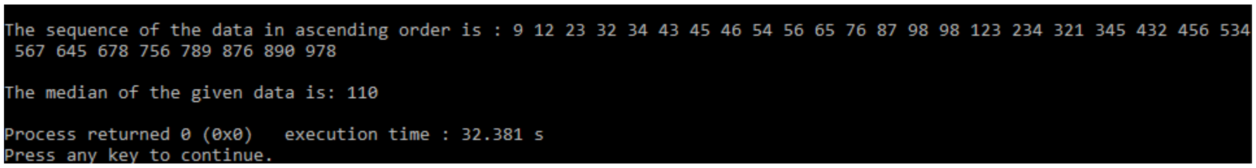
```
The sequence of the data in ascending order is : 3 13 34 34 46 65 543 667 787 8989

The median of the given data is: 55.5

Process returned 0 (0x0)   execution time : 9.010 s
Press any key to continue.
```

N = 20 Data *Ascending* **(Bubble Sort)**



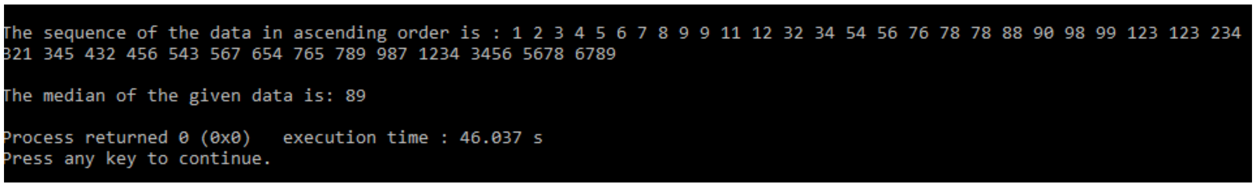
```
The sequence of the data in ascending order is : 7 9 12 12 21 22 31 43 56 56 64 77 85 97 432 564 578 890 1231 3234

The median of the given data is: 60

Process returned 0 (0x0)   execution time : 18.750 s
Press any key to continue.
```

N = 30 Data *Ascending* **(Bubble Sort)**



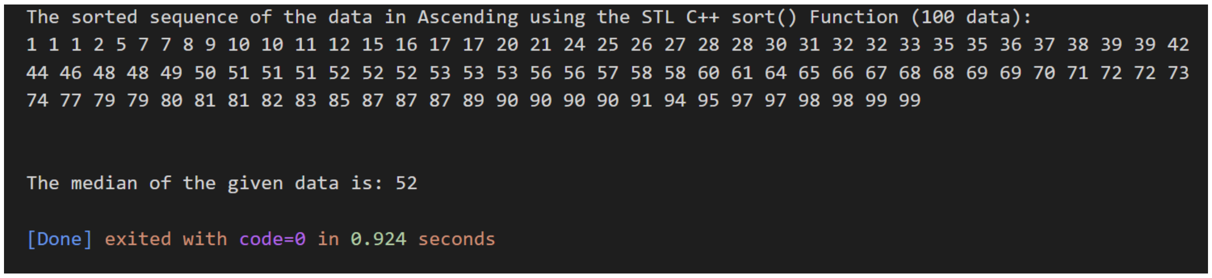
```
The sequence of the data in ascending order is : 9 12 23 32 34 43 45 46 54 56 65 76 87 98 98 123 234 321 345 432 456 534
 567 645 678 756 789 876 890 978

The median of the given data is: 110

Process returned 0 (0x0)   execution time : 32.381 s
Press any key to continue.
```

N = 40 Data *Ascending* **(Bubble Sort)**



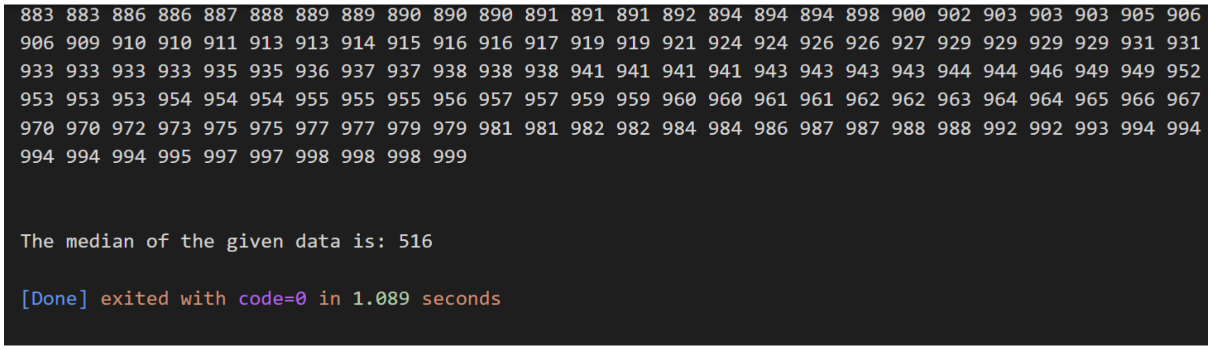
```
The sequence of the data in ascending order is : 1 2 3 4 5 6 7 8 9 9 11 12 32 34 54 56 76 78 78 88 90 98 99 123 123 234
321 345 432 456 543 567 654 765 789 987 1234 3456 5678 6789

The median of the given data is: 89

Process returned 0 (0x0)   execution time : 46.037 s
Press any key to continue.
```

100 Data *Ascending* **(STL Sort)**



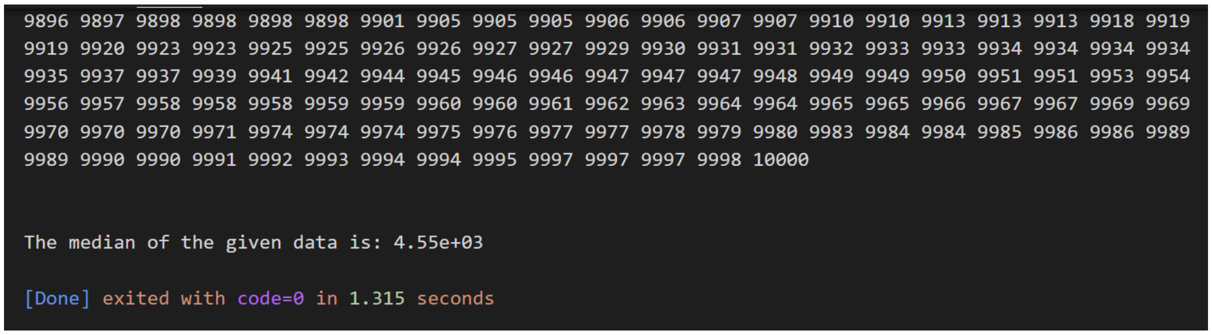
```
The sorted sequence of the data in Ascending using the STL C++ sort() Function (100 data):
1 1 1 2 5 7 7 8 9 10 10 11 12 15 16 17 17 20 21 24 25 26 27 28 28 30 31 32 32 33 35 35 36 37 38 39 39 42
44 46 48 48 49 50 51 51 51 52 52 52 53 53 53 56 56 57 58 58 60 61 64 65 66 67 68 68 69 69 70 71 72 72 73
74 77 79 79 80 81 81 82 83 85 87 87 87 89 90 90 90 90 91 94 95 97 97 98 98 99 99


The median of the given data is: 52

[Done] exited with code=0 in 0.924 seconds
```

1.000 Data *Ascending* **(STL Sort)**



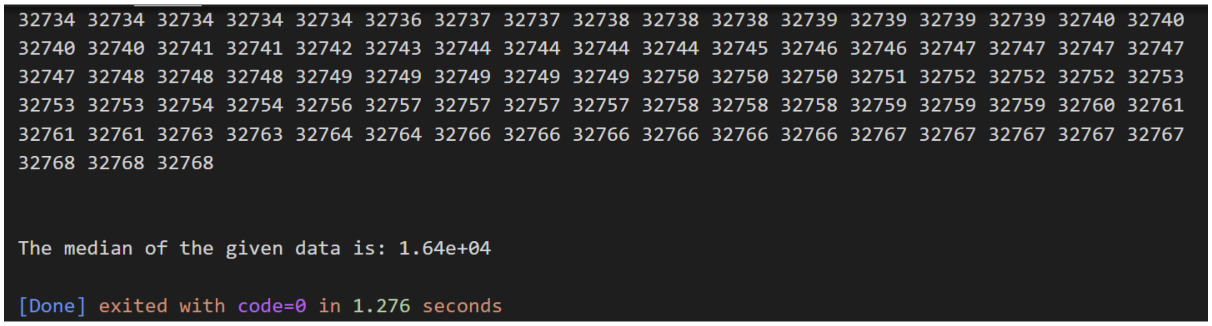
```
883 883 886 886 887 888 889 889 890 890 890 891 891 891 892 894 894 894 898 900 902 903 903 903 905 906
906 909 910 910 911 913 913 914 915 916 916 917 919 919 921 924 924 926 926 927 929 929 929 929 931 931
933 933 933 933 935 935 936 937 937 938 938 938 941 941 941 941 943 943 943 943 944 944 946 949 949 952
953 953 953 954 954 954 955 955 955 956 957 957 959 959 960 960 961 961 962 962 963 964 964 965 966 967
970 970 972 973 975 975 977 977 979 979 981 981 982 982 984 984 986 987 987 988 988 992 992 993 994 994
994 994 994 995 997 997 998 998 998 999


The median of the given data is: 516

[Done] exited with code=0 in 1.089 seconds
```

10.000 Data *Ascending* **(STL Sort)**



```
9896 9897 9898 9898 9898 9898 9901 9905 9905 9905 9906 9906 9907 9907 9910 9910 9913 9913 9913 9918 9919
9919 9920 9923 9923 9925 9925 9926 9926 9927 9927 9929 9930 9931 9931 9932 9933 9933 9934 9934 9934 9934
9935 9937 9937 9939 9941 9942 9944 9945 9946 9946 9947 9947 9947 9948 9949 9949 9950 9951 9951 9953 9954
9956 9957 9958 9958 9958 9959 9959 9960 9960 9961 9962 9963 9964 9964 9965 9965 9966 9967 9967 9969 9969
9970 9970 9970 9971 9974 9974 9974 9975 9976 9977 9977 9978 9979 9980 9983 9984 9984 9985 9986 9986 9989
9989 9990 9990 9991 9992 9993 9994 9994 9995 9997 9997 9997 9998 10000


The median of the given data is: 4.55e+03

[Done] exited with code=0 in 1.315 seconds
```

100.000 Data *Ascending* **(STL Sort)**



```
32734 32734 32734 32734 32734 32736 32737 32737 32738 32738 32738 32739 32739 32739 32739 32740 32740
32740 32740 32741 32741 32742 32743 32744 32744 32744 32744 32745 32746 32746 32747 32747 32747 32747
32747 32748 32748 32748 32749 32749 32749 32749 32749 32750 32750 32750 32751 32752 32752 32752 32753
32753 32753 32754 32754 32756 32757 32757 32757 32757 32758 32758 32758 32759 32759 32759 32760 32761
32761 32761 32763 32763 32764 32764 32766 32766 32766 32766 32766 32766 32767 32767 32767 32767 32767
32768 32768 32768


The median of the given data is: 1.64e+04

[Done] exited with code=0 in 1.276 seconds
```