```
//
//  main.cpp
//  Assignment 2 3d5a
//
//  Created by daylin on 17/11/2016.
//  Copyright © 2016 daylin. All rights reserved.
//

#include <iostream> //used for cout,cin functions
#include <iomanip> // used for setw function
#include <fstream> // used for outFile function

using namespace std;


// reads in a sequences of numbers inputed by the user and stores it on an array
void readinarray(double a[], int capacity, double sentinel, int& asize);
// outputs the sorted sequence of numbers into a .text file for future reference
void readOut(double a[],int size);
// function that uses the buble sort algorithm to sort the sequence of numbers
void bubbleSort(double arr[], int n, int& e);
// function that uses the quicksort algorithm to sort the sequence of numbers
void quickSort(double arr[], int left, int right, int& e);
// prints the sorted sequence of numbers
void print(double a[], int size);


int main()

{
    int beffic=0, qeffic=0;
    const int MAX_SIZE = 1000;// determines the maximum space that will be alocate in memory; preatty big in order to avoid                truncation of the sequence of numbers in case of insufficient space allocated for the array.
    int asize;//used to determine the size of the array
    char choice1=0, choice2=0;
    const double SENTINEL = 9999;// used to determine the end of the sequence so that while loop can be exited.
    double numbers[MAX_SIZE];// used as data structure to store the sequence of numbers.
```

```
    cout<< "Do you want to input the sequence to be sorted? (y-yes,n-no):"<<endl;
    cin >>choice1;

    if (choice1=='y')
      {
         for(int i=0;i<MAX_SIZE;i++)//Initialize array.
           numbers[i]=0;

         cout << "Please enter a sequence of up to 1000 numbers to be sorted (end with 9999): " <<endl;
         readinarray  (numbers,MAX_SIZE,SENTINEL,asize);//stores  sequence of numbers into the array

         cout<< "The sequence has "<<asize<<" terms." <<endl;//prints out n as size of the array for the user in order to calculate O(n)

         cout << "Sorted using BubbleSort: "<<endl;
         bubbleSort (numbers, asize, beffic); // sorting fucntion that uses the bubble algorithm
         cout <<"Probe count: "<<beffic<<endl;

         cout << "Sorted using QuickSort: "<<endl;
         quickSort(numbers,0,  asize-1,  qeffic);// sorting function that uses the quicksort alogorithm
         cout <<"Probe count: "<<qeffic<<endl;

         print (numbers, asize);
         readOut(numbers, asize);

      }

    else
      {
          cout<<"Choose the type of array you wish to sort for this exercise?"<<endl;
          cout<<" a - randomnly distributed"<<endl; // 35 against 11 average case  for both bubblesort and quicksort
          cout<<" b - mostly sorted"<<endl;          // 35 against  7 best case for both bubblesort O(n^2) and quicksort O(n)
          cout<<" c - sorted but in inverse"<<endl; // 45 against 11 worst case for both bubblesort O(n^2) and quicksort O(n log n)
```

```
          cout<<" d - sorted"<<endl;// this option is only given to evidence
that bubble sort will stop after one pass if the array given is already
sorted: 9 probes = 1 full pass over the array - adaptive property
          cin>> choice2;
          // this 4 options are offered to make it easier to show how the
complexity O() greatly varies with the type of data sequence the algorith
is presented with.

          if (choice2=='a')
          {
              double array1[10]= {1,9,2,8,3,7,4,6,5,10};

              cout<< "The sequence has 10 terms." <<endl;
              print ( array1, 10);

              cout << "Sorted using BubbleSort: "<<endl;
              bubbleSort ( array1, 10, beffic);
              cout <<"Probe count: "<<beffic<<endl;

              cout << "Sorted using QuickSort: "<<endl;
              quickSort(array1,0, 10-1, qeffic);
              cout <<"Probe count: "<<qeffic<<endl;

              print (array1, 10);
              readOut( array1, 10);
          }

          else if (choice2=='b') // by using else if and else instead of
another if statment the program is more efficent as it does not need to
check all the recurrent if statments
          {
              double array2[]= {1,2,3,8,5,6,7,4,9,10};
              cout<< "The sequence has 10 terms." <<endl;
              print ( array2, 10);

              cout << "Sorted using BubbleSort: "<<endl;
              bubbleSort ( array2, 10, beffic);
              cout <<"Probe count: "<<beffic<<endl;

              cout << "Sorted using QuickSort: "<<endl;
              quickSort(array2,0, 10-1, qeffic);
              cout <<"Probe count: "<<qeffic<<endl;

              print (array2, 10);
              readOut( array2, 10);
          }
```

```
          else if (choice2=='d')
          {
              double array2[]= {1,2,3,4,5,6,7,8,9,10};
              cout<< "The sequence has 10 terms." <<endl;
              print ( array2, 10);

              cout << "Sorted using BubbleSort: "<<endl;
              bubbleSort ( array2, 10, beffic);
              cout <<"Probe count: "<<beffic<<endl;

              cout << "Sorted using QuickSort: "<<endl;
              quickSort(array2,0, 10-1, qeffic);
              cout <<"Probe count: "<<qeffic<<endl;

              print (array2, 10);
              readOut( array2, 10);
          }

          else
          {
              double array3[]= {10,9,8,7,6,5,4,3,2,1};
              cout<< "The sequence has 10 terms." <<endl;
              print ( array3, 10);

              cout << "Sorted using BubbleSort: "<<endl;
              bubbleSort ( array3, 10, beffic);
              cout <<"Probe count: "<<beffic<<endl;

              cout << "Sorted using QuickSort: "<<endl;
              quickSort(array3,0, 10-1, qeffic);
              cout <<"Probe count: "<<qeffic<<endl;

              print (array3, 10);
              readOut( array3, 10);
          }
      }
}

void readinarray(double a[], int capacity, double sentinel, int& size)
//function used to store the sequence of numbers inputed by the user into
an array of up to 1000 elements
{
    size = 0;
    double x=0;
    cin >> x;
```

```
    while (size < capacity && x != sentinel)// use of sentinel so that the
size of the array would be adaptable
    {
        a[size] = x;
        size = size+1;
        cin >> x; //makes sure that one of the end of while loop conditions
is reached in order to avoid infinite loops
    }
}

void  readOut(double a[], int size)
{
    int count=0;
    ofstream outFile;
    outFile.open("Dataout.txt");

    if (outFile.fail())// checks if the function is able to opend the output
file
        cout << "Unable to save sorted list" << endl;

    for(int i=0;i<size;i++)
    {
        outFile<<left<<setw(12)<<a[i]<<" ";
        count++;
        if (count==10)
        {
            cout<<endl;
            count=0;
        }
    }
    outFile << "test on output function" << endl;
    outFile.close();


}

void bubbleSort(double a[], int size, int& e)
{
    bool swapped = true;
    double temp_a[size];// used to coppy the sequence of numbers to be
sorted such that the original array is unaltered when the quikSort function
is called afterwards in main()
    for (int i=0;i<size;i++)
        temp_a[i]=a[i];
    int j = 0;
    double tmp;
```

```
    while (swapped) // checks if any swaps have been made
    {
        swapped = false;
        j++; //

        for (int i = 0; i < size - j; i++)// this line makes the algorithm
repeat the checks but not check again the top elements of the array as they
have been already sorted;this increases efficiency.
        {
            if (temp_a[i] > temp_a[i + 1])// this loop swaps the elements
if previous element is bigger then the next element
            {
                tmp = temp_a[i];
                temp_a[i] = temp_a[i + 1];
                temp_a[i + 1] = tmp;
                swapped = true;
            }
            e++; // probe count
        }
    }
    print (temp_a, size);
}

void quickSort(double a[], int indleft, int indright, int& e)// the left
and right indeces were chosen as the first and last element of the array
respectively
{

    int i = indleft, j = indright;
    double tmp;// facilitates the swap of elements of the array
    double pivot = a[(indleft + indright) / 2];

    // partition or divide and conquer:  this part of the code separates
all elements < pivot into the left side and all elements > pivot into the
right side.
    while (i <= j)
    {
        e++;// probe count
        while (a[i] < pivot)
            i++;
        while (a[j] > pivot)
            j--;
        if (i <= j)
        {
            tmp = a[i];
            a[i] = a[j];
```

```
            a[j] = tmp;
            i++;
            j--;
        }
    };
    //recursion the algorithm treats the right and left side of the array as new arrays and starts spliting them around a new individual pivot in an recursive fashion
    if (indleft < j)
        quickSort(a, indleft, j, e);
    if (i < indright)
        quickSort(a, i, indright, e);

}

void print(double a[], int size)

{
   int count=0;
   for(int i=0;i<size;i++)
     {
       cout<<setw(4)<<a[i]<<" ";
       count++;
       if (count==10) // this loop checks if 10 elementa have already been printed and then introduces in a new line so that the data is printed as rows of 10 elements
        {
            cout<<endl;
            count=0;
        }
     }
}
```