KARAN GUPTA

SEPTEMBER 29, 2017

CSC600-02

HOMEWORK 2

/**

# HW2 QUESTION 1

The array a(1..n) contains sorted integers. Write a function maxlen(a,n) that returns the
length of the longest sequence of identical numbers (for example, if a=(1,1,1,2,3,3,5,6,6,6,6,7,9) then maxlen returns 4 because the longest sequence 6,6,6,6
contains 4 numbers. Write a demo main program for testing the work of maxlen. Explain
your solution, and insert comments in your program. The solution should have time
complexity O(n).
**/

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

//create an array maxlen of a specific size
int maxlen (int newArg[], int size){
    //create a variable which will be the streak number and a
comparing count
    int streakNumber=0, count=0;
    //To check the previous number in the array we declare it to a
variable
    int previousNum = newArg[0];

    //Iterating through the array to compare it with the previous
number is the array.
    for(int index=1; index<size; index++)
    {
        //since we started comparing from the next number in the array
we initialise it to a variable
        int currentNumber = newArg[index];
        //check if they the previous number and the current number are
same and if not then count remains the same
        if(previousNum == currentNumber)
        {
            count++;
            //check if the count is incremented if yes then make that
as the streakNumber
            if(count > streakNumber){streakNumber = count;}
        }else{count = 1;}
        previousNum = currentNumber;
    }
    return streakNumber;
}
```

```
//Testing code above with a testing array for the longest streak of
numbers
int main(){
    int testingArray[10] = {1,1,1,1,2,3,4,5,5,5};
    cout << "The longest streak is: "  << maxlen (testingArray,10);
    return 0;
}
/**
Since we use only one for loop in the code with iteration to n, this
seems to be the most efficient code
to find the longest streak in an array with time complexity of O(n).
**/
```

**MY OUTPUT**

```
"/Users/karangupta/Desktop/CSC600 hw2/cmake-build-debug/CSC600_hw2"
The longest streak is: 3
Process finished with exit code 0
```

```
/**
 HW 2 QUESTION 2
 We know three points on a curve: (x1, y1), (x2, y2), (x3, y3) .
 You have to create a program that for any value x1 <= x <= x3
computes the
 corresponding value y assuming that the segment of curve can be
 approximated with the parabola y = ax^2 + bx + c 2.
 Write a function that can be called as y(x1,y1,x2,y2,x3,y3,x)
 and a main program that reads x1,y1,x2,y2,x3,y3 and then displays
 y(x1,y1,x2,y2,x3,y3,x) in n=40 equidistant x points between x1 and
x3.
 Solve this problem in the following two ways:
 (a) Insert points (x1, y1), (x2, y2), (x3, y3) in parabola  y(x) =
ax^2 + bx + c . You
 will get three linear equations; solve them and find a,b,c. Then,
display the table of
 function y(x) from x1 to x3.
 (b) Solve the same problem using Lagrange interpolation formula.
 (c) Compare your two solutions and show your conclusions.
**/

#include <iostream>
```

```
using namespace std;

//Form a function for parabola and its equidistant point
void y(double x1, double y1, double x2, double y2, double x3, double
y3, float x){
    int a,b,c;
//Using the Lagrange interpolation formula
    a = (((y1-y2)*(x1-x3)) - ((x1-x2)*(y1-y3)))/(((x1*x1 - x2*x2)*(x1-
x3)) - ((x1-x2)*(x1*x1 - x3*x3)));
    b = ((y1-y2)-(((x1*x1) - (x2*x2))*a))/(x1-x2);
    c = y1 - ((x1*x1)*a) - (x1*b);
    //Print the parabola function and its equidistant point
    cout << "The equation is: y(x) = " << a << "x^2 + " << b << "x + "
<< c << endl;
    cout << "Equidistant point: "<< ((x3-x1)/40.0) << endl;
//We have to use a for loop to increment through and print
    for(int index = 0; index < 40; index++){
        float newCount = (float)index;
        x = x1 + ((x3-x1)/40.0) * newCount;
        printf("(%.2f, %.2f) \n", x, (a*x*x + b*x + c));
    }
}

// Test the parabola by asking for input and then call the above void
function
int main(){
    int x1, y1, x2, y2, x3, y3;
    float x;
//systematically ask for 6 inputs for the function to work
    cout<<"Please enter the following: \nx1:";
    cin >> x1;
    cout <<  "y1:";
    cin >> y1;
    cout <<  "x2:";
    cin >> x2;
    cout <<  "y2:";
    cin >> y2;
    cout <<  "x3:";
    cin >> x3;
    cout <<  "y3:";
    cin >> y3;
    y(x1,y1,x2,y2,x3,y3,x);
    return 0;
}
/**
 According to me this solution solve's all that was asked and in the
most efficient way.
 I broke the question into parts and the commented what I thought was
right then I wrote
 the main method so that I could test my code if it was wrong. After
```

writing the comments
 and the main method then I wrote the function. I had to refer internet to revise what
 Lagrange interpolation formula was before approaching it.
**/

## MY OUTPUT

```
"/Users/karangupta/Desktop/csc600 hw2 part 2/cmake-build-debug/CSC600_hw2_part_2"
Please enter the following:
x1:2
y1:4
x2:3
y2:5
x3:6
y3:7
The equation is: y(x) = 0x^2 + 1x + 2
Equidistant point: 0.1
(2.00, 4.00)
(2.10, 4.10)
(2.20, 4.20)
(2.30, 4.30)
(2.40, 4.40)
(2.50, 4.50)
(2.60, 4.60)
(2.70, 4.70)
(2.80, 4.80)
(2.90, 4.90)
(3.00, 5.00)
(3.10, 5.10)
(3.20, 5.20)
(3.30, 5.30)
(3.40, 5.40)
(3.50, 5.50)
(3.60, 5.60)
(3.70, 5.70)
(3.80, 5.80)
(3.90, 5.90)
(4.00, 6.00)
```

```
(4.00, 6.00)
(4.10, 6.10)
(4.20, 6.20)
(4.30, 6.30)
(4.40, 6.40)
(4.50, 6.50)
(4.60, 6.60)
(4.70, 6.70)
(4.80, 6.80)
(4.90, 6.90)
(5.00, 7.00)
(5.10, 7.10)
(5.20, 7.20)
(5.30, 7.30)
(5.40, 7.40)
(5.50, 7.50)
(5.60, 7.60)
(5.70, 7.70)
(5.80, 7.80)
(5.90, 7.90)

Process finished with exit code 0
```

/**

# HW2 QUESTION 3

The array a(1..n) contains arbitrary integers. Write a function reduce(a,n) that reduces
the array a(1..n) by eliminating from it all values that are equal to three largest different
integers. For example, if a=(9,1,1,6,7,1,2,3,3,5,6,6,6,6,7,9) then three largest different
integers are 6,7,9 and after reduction the reduced array will be a=(1,1,1,2,3,3,5), n=7. The

```
solution should have time complexity O(n).
 **/

#include <iostream>
using namespace std;
// Form a function that finds the largest number takes them out and
the prints the rest with its size
void reduce(int* a, int n){
//We need three variables for large numbers, a number for the left
over size and then an array
    int firstLargest =0, secondLargest=0, thirdLargest=0, size = 0;
    int *newArray=new int[size];

    // iterate through the loop to get the largest 3 value
    for(int index = 0; index< n; index++){
        if(firstLargest < a[index])
            firstLargest = a[index];  // first largest number
        if(secondLargest < a[index] && a[index] < firstLargest)
            secondLargest= a[index];   // second largest number
        if(thirdLargest < a[index] && a[index] < secondLargest)
            thirdLargest= a[index];   // third largest number
    }
    //print the three largest number
    cout<< "Three Largest different Integers: "
        << firstLargest << " " << secondLargest << " " << thirdLargest
<< endl << endl;

    //Form a new array removing the largest numbers and then print it
    cout<<"Array Elements After Reduction: a= ";
    for(int index=0; index< n; index++){
        if((a[index] != firstLargest) && (a[index] != secondLargest)
&& (a[index] != thirdLargest)) {
            newArray[size] = a[index];
            cout<<newArray[size]<<",";
            size++;
            //print the size after the loop ends
        }
    }
    cout<<" n = "<< size << endl;
}
//Test the function above with the same example in the question to be
sure about the output in the main function
int main(){
    int a[]={9,1,1,6,7,1,2,3,3,5,6,6,6,6,7,9};
    //We call the reduce function for separating and printing
    reduce(a, 16);
    return 0;
}
/**
I feel this is the best solution for the problem as it solves what is
```

asked with the most efficient
method. We used total of 2 for loops but not in a nested format therefore the n is iterated once
in both for loops that means the efficiency for this is O(n). The first for loop is separating the
largest numbers and the second for loop is used for reducing and printing them as sorted.
**/

**MY OUTPUT**

```
"/Users/karangupta/Desktop/csc600 hw2 part 3/cmake-build-debug/csc600_hw2_part_3"
Three Largest different Integers: 9 7 6

Array Elements After Reduction: a= 1,1,1,2,3,3,5, n = 7

Process finished with exit code 0
```

/**

# HW2 QUESTION 4

Write a program BigInt(n) that displays an arbitrary positive integer n using big
characters of size 7x7, as in the following example for BigInt(170):
**/

```
#include <iostream>
using namespace std;
//We create a function that makes the numbers in a 2D matrix way.
void BigInt(int n){
    //All the numbers created should be in 2D matrix of string variable.
    string numbers[7][10] = {
            { " @@@@@ "  ,  "    @   "  ,  "  @@@@@  "  ,  " @@@@@   "
,  "    @  "  ,  "  @@@@@ "  ,  " @@@@@  "  ,  "  @@@@@ "  ,  " @@@@@
"  ,  "  @@@  " } ,
            { " @   @ "  ,  "  @ @   "  ,  "      @  "  ,  "     @   "
,  "   @@  "  ,  "  @     "  ,  " @      "  ,  "      @ "  ,  " @   @
"  ,  " @   @ " } ,
            { " @   @ "  ,  "    @   "  ,  "      @  "  ,  "     @   "
,  "  @ @  "  ,  "  @     "  ,  " @      "  ,  "      @ "  ,  " @   @
"  ,  " @   @ " } ,
            { " @   @ "  ,  "    @   "  ,  "  @@@@@  "  ,  "  @@@@   "
,  " @  @  "  ,  "  @@@@@ "  ,  " @@@@@  "  ,  "      @ "  ,  " @@@@@
```

```
"   ,   "  @@@@ " } ,
            { " @   @ "   ,  "    @   "   ,  "  @      "   ,  "     @   "
,  " @@@@@ "   ,  "      @ "   ,  " @   @  "   ,  "      @ "   ,  " @   @
"   ,  "     @ " } ,
            { " @   @ "   ,  "    @   "   ,  "  @      "   ,  "     @   "
,  "    @  "   ,  "      @ "   ,  " @   @  "   ,  "      @ "   ,  " @   @
"   ,  " @   @ " } ,
            { " @@@@@ "   ,  "  @@@@@ "   ,  "  @@@@@  "   ,  " @@@@@   "
,  "    @  "   ,  "  @@@@@ "   ,  " @@@@@  "   ,  "      @ "   ,  " @@@@@
"   ,  "  @@@  " }
    };
    // Need to have a while loop so that we could segment the input
into the desired number that need to be printed
    int i = 0, j = 0, allNumbers[10] ;
    while(n > 0){
        //We divide the number by 10 which helps us giving us a single
number to print each time its iterated
        allNumbers[i] = n % 10;
        n /= 10;
        i++;
        j++;
    }
    // We print the numbers using a nested for loop.
    int index, index1;
    for(index=0; index<7; index++){
        for(index1= j-1; index1>=0 ; index1--){
            cout<< numbers [index][allNumbers[index1]];
        }
            cout << endl;
    }
}

// We test the above function in the main function by just giving it a
random input which I am taking from the question since we know the
output
int main(){
    //We call the function with any random output
    BigInt(170);
    return 0;
}

/**
I feel this is the best solution for this question, even though we
used a while loop and 2 for loops for
dividing the input and printing respectively, even then this is the
most efficient code for this question.
I was confused between two approaches for this question, I was first
thinking to store the number into switch
and each case would represent a number but then I decided to go with
2D matrix and then priting it accordingle
```

besides calling the switch cases.
**/

## MY SOURCE CODE

```cpp
/**
HW2 QUESTION 4
Write a program BigInt(n) that displays an arbitrary positive integer n using big
characters of size 7x7, as in the following example for BigInt(170):
**/

#include <iostream>
using namespace std;
//We create a function that makes the numbers in a 2D matrix way.
void BigInt(int n){
    //All the numbers created should be in 2D matrix of string variable.
    string numbers[7][10] = {
            { " @@@@@ " , "   @   " , " @@@@@ " , " @@@@@ " , "   @   " , " @@@@@ " , " @@@@@ " , " @@@@@ " , " @@@ " } ,
            { " @   @ " , "  @ @  " , "     @ " , "     @ " , "  @@@  " , " @     " , " @     " , "     @ " , " @   @ " } ,
            { " @   @ " , "   @   " , "     @ " , "     @ " , " @ @ @ " , " @     " , " @     " , "     @ " , " @   @ " } ,
            { " @   @ " , "   @   " , " @@@@@ " , " @@@@  " , " @@@@@ " , " @@@@@ " , " @@@@@ " , "  @@@@ " , "  @@@@ " } ,
            { " @   @ " , "   @   " , " @     " , "     @ " , "     @ " , "     @ " , " @   @ " , " @   @ " , " @   @ " } ,
            { " @   @ " , "   @   " , " @     " , " @@@@@ " , " @@@@@ " , "     @ " , "  @@@  " , "  @@@  " , "  @@@  " } ,
            { " @@@@@ " , " @@@@@ " , " @@@@@ " , " @@@@@ " , "   @   " , " @@@@@ " , " @@@@@ " , " @@@@@ " , " @@@ " } ,
    };
    // Need to have a while loop so that we could segment the input into the desired number that need to be printed
    int i = 0, j = 0, allNumbers[10] ;
    while(n > 0){
        //We divide the number by 10 which helps us giving us a single number to print each time its iterated
        allNumbers[i] = n % 10;
        n /= 10;
        i++;
        j++;
    }
    // We print the numbers using a nested for loop.
    int index, index1;
    for(index=0; index<7; index++){
        for(index1= j-1; index1>=0 ; index1--){
            cout<< numbers [index][allNumbers[index1]];
        }
            cout << endl;
    }
}

// We test the above function in the main function by just giving it a random input which I am taking from the question since we know the output
int main(){
    //We call the function with any random output
    BigInt(170);
    return 0;
}

/**
```

## MY OUTPUT

```
"/Users/karangupta/Desktop/csc600 hw2 part 4/cmake-build-debug/csc600_hw2_part_4"
    @     @@@@@  @@@@@
  @ @         @  @   @
    @         @  @   @
    @         @  @   @
    @         @  @   @
    @         @  @   @
  @@@@@       @  @@@@@

Process finished with exit code 0
```

/**

# HW2 QUESTION 5

Make a sorted integer array a[i]=i, i=0,…,n-1. Let bs(a,n,x) be a binary search

```
program that returns the index i of array a[0..n-1] where a[i]=x.
Obviously,
bs(a,n,x)=x, and the binary search function can be tested using the
loop
for(j=0; j<K; j++)
for(i=0; i<n; i++) if(bs(a,n,i) != i) cout << "\nERROR";
Select the largest n your software can support and then K so that this
loop with an
iterative version of bs runs 3 seconds or more. Then measure and
compare this run time
and the run time of the loop that uses a recursive version of bs.
Compare these run times
using maximum compiler optimization (release version) and the slowest
version
(minimum optimization or the debug version). If you use a laptop, make
measurements
using AC power, and then same measurements using only the battery.
What conclusions
can you derive from these experiments? Who is faster? Why?
 **/

#include <iostream>
using namespace std;

// Binary search function we use a while loop for better efficiency
int bs(int* a,int minimum, int maximum, int x){

    while (minimum <= maximum) {
        int mid;
        mid = (minimum + maximum) / 2;
        if (a[mid] > x) { maximum = mid - 1;}
        else if (a[mid] == x) { return mid;}
        else { minimum = mid + 1;}
    }
    return -1;
}

//Recursive version of the search, we dont need any loops to sort.
int bsRecursive(int* a, int minimum, int maximum, int x){

    if(minimum == maximum) {return -1;}
    else{
        int mid = minimum+((maximum-minimum) / 2);
        if (a[mid] < x) { return bsRecursive(a, mid + 1, maximum, x);}
        else if (a[mid] > x) { return bsRecursive(a,minimum, mid - 1,
x);}
        else { return mid;}
    }
}
//we use clock per second expression to get the time and initilise it
```

```
to a variable
double seconds(){
    return ((double)clock())/CLOCKS_PER_SEC;
}
//We use the above functions to read its run time in the main
function. We would probably need a nested for loop
int main(){

    int K = 1000000, n= 100, a[n];
    double timeStart; // starting the clock
    double timeEnd; // ending the clock

    // Instatiating the array
    for(int index=0;index<n;index++) {
        a[index] = index;
    }
    // START TIME
    timeStart= seconds();
    //We will go through the above mentioned function and print an
error message if needed
    for(int j=0; j<K;j++){
        for(int i=0;i<n;i++){
            if(bs(a,0,n,i)!=i)
                cout<<"ERROR" << endl;
        }
    }
    //END TIME
    timeEnd = seconds()- timeStart;
    cout<<"Iterative seconds:"<< timeEnd<< endl;

    //START TIME
    timeStart=seconds();
    //We will go through the above mentioned function and print an
error message if needed
    for(int j=0; j<K;j++){
        for(int i=0;i<n;i++){
            if(bs(a,0,n,i)!=i)
                cout<<"\nERROR"<<endl;
        }
    }
    //END TIME
    timeEnd= seconds()- timeStart;
    cout<<"Recursive seconds:"<< timeEnd<< endl;

    return 0;
}

/**
I feel this code can find time needed and it first explains both
binary search function and then recursive function
```

then we create the function of time. In the main file we use our function to test for time. According to me recursive
method has proved itself to be a faster approach than other iterative method. I also tried testing it and I feel
that it does when the charger is plugged than it is just on the battery.
**/

**MY OUTPUT**

```
"/Users/karangupta/Desktop/csc600 hw2 part 5/cmake-build-debug/csc600_hw2_part_5"
Iterative seconds:5.05003
Recursive seconds:4.99763

Process finished with exit code 0
```