CSCE 221 Cover Page

Homework #1

Due February 11 at midnight to eCampus

First Name Hunter Last Name Cleary UIN 625001547

User Name hncleary E-mail address hncleary@tamu.edu

Please list all sources in the table below including web pages which you used to solve or implement the current homework. If you fail to cite sources you can get a lower number of points or even zero, read more: Aggie Honor System Office

Type of sources			
People			
Web pages (provide URL)	URLS Below Table		
Printed material	Data Structures and Algorithms (Textbook)		
Other Sources			

URLS

https://en.wikibooks.org/wiki/C%2B%2B$_{P}rogramming/Classes/Abstract_{C}lasses$
$https://en.wikipedia.org/wiki/Best,_{w}orst_{a}nd_{a}verage_{c}ase$
$https://stackoverflow.com/questions/2236197/whatistheeasiestway-toinitializeastdvectorwithhardcoded-elements$
$https://www.geeksforgeeks.org/binarysearch/$
$https://rob-bell.net/2009/06/a-beginners-guide-to-big-o-notation/$

I certify that I have listed all the sources that I used to develop the solutions/codes to the submitted work.

"On my honor as an Aggie, I have neither given nor received any unauthorized help on this academic work."

Your Name Hunter Cleary Date 2/11/18

Type solutions to the homework problems listed below using preferably LyX/LaTeX word processors, see the class webpage for more information about their installation and tutorials.

1. (10 points) Write the C++ classes called `ArithmeticProgression` and `GeometricProgression` that are derived from the abstract class `Progression`, with two pure virtual functions, `getNext()` and `sum()`, see the course textbook p. 87–90 for more details. Each subclass should implement these functions in order to generate elements of the sequences and their sums. Test your program for the different values of d, r and the number of elements n in each progression.
What is the classification of those functions: `getNext()` and `sum()` in terms of the Big-O notation?

   Recall the definitions of the arithmetic and geometric progressions.

   Definition: An *arithmetic progression* with the initial term a and the common real difference d is a sequence of the form

   $a, a+d, a+2d, \ldots, a+nd, \ldots$

   Definition: A *geometric progression* with the initial term a and the common real ratio r is a sequence of the form

   $a, ar, ar^2, \ldots, ar^n, \ldots$

Listing 1: Progression

```
1  #include<iostream>
2  #include<math.h> //power function for geometric progression
3
4  using namespace std;
5
6  class Progression{
7      public:
8      virtual int getnext() = 0;
9      virtual int sum() = 0;
10
11     void setCurrenVal(int a){
12         currentVal = a;
13     }
14     void setIncrement(int d){
15         increment = d;
16     }
17
18     void setRatio(int r){
19         ratio = r;
20     }
21     void setFirstValue(int f){
22         firstValue = f;
23     }
24     void setElementNumber(int n){
25         elementNumber = n;
26     }
27     protected:
28     int currentVal;
29     int elementNumber = 0;
30     int increment;
31     int ratio;
32     int firstValue;
33 };
34
35 class ArithmeticProgression : public Progression{
36     public:
```

```
37      int getnext(){
38          currentVal = (currentVal + (increment*elementNumber));
39          ++elementNumber;
40          return currentVal;
41      }
42      //adds up values until provided element number
43      int sum(int s){
44          int total;
45          for(int i = 1; i < s; ++i){
46              total = currentVal + getnext(i);
47          }
48      }
49  };
50  class GeometricProgression : public Progression{
51      public:
52      int getnext(){
53          currentVal = (firstValue * (pow(ratio,(elementNumber+1))));
54          ++elementNumber;
55          return currentVal;
56      }
57      //adds up values until provided element number
58      int sum(int s){
59          int total;
60          for(int i = 1; i < s; ++i){
61              total = currentVal + getnext(i);
62          }
63      }
64  };
65  int main(void){
66      ArithmeticProgression linear;
67      linear.setCurrenVal(0);
68      linear.setIncrement(2);
69      lineear.setFirstValue(1);
70      cout << linear.getnext() << endl;
71      cout << linear.sum() << endl;
72
73      GeometricProgression ratio;
74      ratio.setCurrenVal(0);
75      ratio.setRatio(2);
76      ratio.setFirstValue(2);
77      cout << ratio.getnext() << endl;
78      cout << ratio.sum() << endl;
79      return 0;
80  }
```

Classifications

The classification of the getnext() function is O(1). The function only accesses what number comes next in the series. the classification of the sum() function is O(n). This function adds each numbers in a series to the previous. Run time is dependent on the length of the series.

2. (10 points) Use the STL class `vector<double>` to write a C++ function that takes two vectors, `a` and `b`, of the same size and returns a vector `c` such that $c[i] = a[i] \cdot b[i]$. How many scalar multiplications are used to create elements of the vector `c` of size n? What is the classification of this algorithm in terms of the Big-O notation?

Listing 2: Vector Multiplication

```
#include <iostream>
#include <vector>
using namespace std;
vector<double> vectorMultiply(vector<double> a, vector<double> b){
    vector<double> c;
    if(a.size()==b.size()){
        for(int i = 0; i < a.size(); ++i){
        c.push_back(a[i]*b[i]);
        }
    }
    else{
        c.push_back(0);
        cout << "Invalid vector sizes." << endl;
    }
    return c;
}

int main()
{
    vector<double> a{ 10, 20, 35};
    vector<double> b{ 15, 35, 35};
    for (int x : a)
        cout << x << " ";
    cout << endl;
    for (int x : b)
        cout << x << " ";
    cout << endl;

    for( int y : vectorMultiply(a,b))
        cout << y << " ";

}
```

Classification

The number of scalar multiplications used to create the elements in vector c of size n is equal to n. The number of operations scales directly with the size of the original two vectors and is therefore classified as O(n).

3. (10 points) Use the STL class `vector<int>` to write a C++ function that returns true if there are two elements of the vector for which their product is odd, and returns false otherwise. Provide a formula on the number of scalar multiplications in terms of n, the length of the vector, to solve the problem in the best and the worst cases. Describe the situations of getting the best and worst cases. What is the classification of the algorithm in the best and worst cases in terms of the Big-O notation?

Listing 3: Vector Product Test

```
#include <iostream>
#include <vector>
using namespace std;
bool vectorMultiplyTest(vector<int> a, vector<int> b){
    bool j;
    int i = 0;
    vector<int> c;
        while( j == false && i < a.size() ){
                for(int q = 0; q < b.size(); ++q){
                    if( (a[i]*b[q])%2 == 1){
                    j = true;
                    ++i;
                    }
                    else{
                        ++i;
                    }
                }
            }
    return j;
}
```

Classification

In the best case, the function will find an odd value after multiplying the first values of both vectors. In the worst case, the function will onyl find even values or find an odd value by multiplying the last points of both vectors. Best case = 1.(O(1)) Worst case = (size of vector 1) * (size of vector 2)$\rightarrow n*n \rightarrow n^2$. This results in the classifcation O(n^2).

4. (20 points) Write a templated C++ function called `BinarySearch` which searches for a target `x` of any numeric type `T`, and test it using a sorted vector of type `T`. Provide the formulae on the number of comparisons in terms of n, the length of the vector, when searching for a target in the best and the worst cases. Describe the situations of getting the best and worst cases. What is the classification of the algorithm in the best and worst cases in terms of the Big-O notation?

Listing 4: Templated Search

```
#include<iostream>
#include<iostream>
#include<vector>
using namespace std;
template<typename T>
int BinarySearch(T x, vector<T> v){
    int p;
    int midpoint = (v.size() / 2);
    bool j = false;
    int end = v.size();
    int start = 0;
    int temp;
    while ( j == false){
        if( v[midpoint] == x ){
            j = true;
            p = midpoint;
        }
        else if( v[midpoint+1] == x ){
            j = true;
            p = midpoint+1;
        }
        else if( v[midpoint-1] == x ){
            j = true;
            p = midpoint-1;
        }
        else if(v[midpoint] > x){
            temp = midpoint;
            midpoint = (end-midpoint)/2;
            start = temp;
        }
        else if((v[midpoint] < x)){
            temp = midpoint;
            midpoint = (start-midpoint);
            end = temp;
        }
    }
    return p;
}

int main(){
vector<int> v1{1,2,3,5,7,9,11,15};

cout << "The value is located at: " << BinarySearch(9, v1) << " ";

}
```

Classification

In the best case scenario, the binary search function will find the value located at the midpoint of the vector. The best case performance is O(1). In the worst case, the value will be found using the highest number of midpoints. This results in a worst case of $log_2(n)$ operations, classified at O(logn).

5. (10 points) **(R-4.7 p. 185)** The number of operations executed by algorithms A and B is $8n \log n$ and $2n^2$, respectively. Determine n_0 such that A is better than B for $n \geq n_0$.

   Solve for n:
   $8nlogn = 2n^2$
   $8logn = 2n$
   $4logn = n$
   $4 = (n)/(logn)$
   $n = 16$
   $n >= 16$

6. (10 points) **(R-4.21 p. 186)** Bill has an algorithm, `find2D`, to find an element x in an $n \times n$ array A. The algorithm `find2D` iterates over the rows of A, and calls the algorithm `arrayFind`, see Code Fragment 4.5, p. 184, on each row, until x is found or it has searched all rows of A. What is the worst-case running time of `find2D` in terms of n? What is the worst-case running time of `find2D` in terms of N, where N is the total size of A? Would it be correct to say that `find2D` is a linear-time algorithm? Why or why not?

   Answer
   The worst case running time for find2d in terms of n is $O(n^2)$. It will execute operations for rows * columns (n*n). The worst case running time in terms of N is O(N). The algorithm will have to perform the operation on all points in the array (N) once. It would be correct to say that find2d is a linear time algorithm. The operation is used for N elements in an array.

7. (10 points) **(R-4.39 p. 188)** Al and Bob are arguing about their algorithms. Al claims his $O(n \log n)$-time method is **always** faster than Bob's $O(n^2)$-time method. To settle the issue, they perform a set of experiments. To Al's dismay, they find that if $n < 100$, the $O(n^2)$-time algorithm runs faster, and only when $n \geq 100$ is the $O(n \log n)$-time algorithm better. Explain how this is possible.

   Answer
   Bob's algorithm is only effective for small inputs. If the input is large (n >= 100), then Al's will be more efficient because $100log100 < 100^2$.

8. (20 points) Find the running time functions for the algorithms below and write their classification using Big-O asymptotic notation. The running time function should provide a formula on the number of operations performed on the variable s. Note that array indices start from 0.

```
Algorithm Ex1(A):
  Input: An array A storing n ≥ 1 integers.
  Output: The sum of the elements in A.
s ← A[0]
for i ← 1 to n − 1 do
    s ← s + A[i]
end for
return s
```

Answer

Total Operations = (n-1) * 2 + 1 = n-1

Notation = O(n)

```
Algorithm Ex2(A):
  Input: An array A storing n ≥ 1 integers.
  Output: The sum of the elements at even positions in A.
s ← A[0]
for i ← 2 to n − 1 by increments of 2 do
  s ← s + A[i]
end for
return s
```

Answer

Total Operations = n/2 + n/2 -1 + 1 + 1 = n + 1

Notation = O(n)

```
Algorithm Ex3(A):
   Input: An array A storing n ≥ 1 integers.
   Output: The sum of the partial sums in A.
s ← 0
for i ← 0  to n − 1 do
   s ← s + A[0]
   for j ← 1 to i do
     s ← s + A[j]
   end for
end for
return s
```

Answer

Total Operations = $2n + (2(1 - 2^n - 1)/(1 - 2) * n = 2n + 2^n - 2n = 2^n * n$

Notation = $O(n2^n)$

```
Algorithm Ex4(A):
   Input: An array A storing n ≥ 1 integers.
   Output: The sum of the partial sums in A.
t ← 0
s ← 0
for i ← 1 to n − 1 do
   s ← s + A[i]
   t ← t + s
end for
return t
```

Answer

Total Operations = 2 + 4(n-1) = 4n -4 +2 = 4n -2

Notation = O(n)