CSCE 221 Cover Page

Homework #1

Due June 2, at midnight to eCampus

First Name Abdullah Last Name Ahmad UIN:927009064

User Name Abdullah2808 E-mail address abdullah2808@tmau.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)	https://www.geeksforgeeks.org/binary-search/	https://opendsa-server.cs.vt.edu/ODSA/Boo
Printed material		
Other Sources		

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 Abdullah Ahmad Date 5/31/2020

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) 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? Provide a formula on the number of scalar multiplications in terms of n. What is the classification of this algorithm in terms of the Big-O notation?

```
vector<double> scalar_mult(vector<double> a, vector<double> b) {
        vector<double> c;
        for (int i = 0; i < a.size(); i++) {
                c.push_back(a.at(i) * b.at(i));          }
        return c; }
```

The amount of scalar multiplications required to create elements of the vector c of size n is n. The formula for the number of scalar multiplications in terms of n is n and the classifcation in terms of the Big-O notation is O(n).

2. (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 size of the vector, to solve the problem in the best and worst cases. Describe the situations of getting the best and worst cases, give the samples of the input at each case and check if your formula works. What is the classification of the algorithm in the best and worst cases in terms of the Big-O notation?

```
bool odd_prod(vector<int> a) {
        for (int i = 0; i < a.size(); i++) {
                for (int j = 0; j < i; j++) {
                        if (a[i] * a[j] % 2 == 1) {
                                return true;
                        }
                }
        }
        return false; }
```

The formula in terms of the size of the vector in the worst case scenario is $\frac{n(n+1)}{2}$for the best case scenario the formula is 1. The Big-O notation for the worst case scenario is $O(n^2)$ and O(1) for the best case scenario. If the input is 1 the best scenario is achieved, if you enter an n greater than 1 the worst case scenario will happen.

3. (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 formulas on the number of comparisons in terms of n, the length of the vector, when searching for a target in the best and 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?

```
template <typename T>
int binarysearch(vector<T>vec, int l, int r, T search){
                    while (l <= r) {
                            int m = l + (r - l) / 2;
                            if (vec[m] == search)
                                    return m;
                            if (vec[m] < search)
                                    l = m + 1;
                            else
                                    r = m - 1;
                    }               return -1;
        }
```

In the best case when searching for a target the formula in terms of n is 1, and in the worst case is $(\frac{n}{2} + 1)$. To get the best case you would have to find the middle term, and to get the worst case you would find nothing. The classifcation of the algorithm in the best case is O(1), this occurs O(log(n)).

4. (10 points) 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 faster than B for $n \geq n_0$.

   Using algebra to solve

$$\begin{aligned} 8n\,log\,n &= 2n^2 \\ 4log\,n &= n \\ 4 &= \frac{n}{log\,n} \end{aligned}$$

   Solving for n you get 16, so for A to be faster than B you have to have an n_0 of 17.

5. (10 points) Two friends are arguing about their algorithms. The first one claims his $O(n \log n)$-time algorithm is **always** faster than the second friend's $O(n^2)$-time algorithm. To settle the issue, they perform a set of experiments. Show that it is possible to find an input $n < 100$ that the $O(n^2)$-time algorithm runs faster, and only when $n \geq 100$ that the $O(n \log n)$-time algorithm is faster.

   Big-O notation calculates complexity when n is arbitrarily large, so $O(n \log n)$ will be faster whenever the number of inputs grows larger. However it is possible for $O(n^2)$ to be faster at smaller inputs, because of variables like better memory allocation or other non-algorithmic effects. Big-O notation is more useful whenever n grows very large, but in real life cases where factors such size of the iterations or memory allocation comes into affect, it is possible for $O(n^2)$to be faster. This can be proven by measuring exectution time of both algorithms for different n's and you will find that for n < 100 $O(n^2)$ is faster, but for $n \geq 100$ $O(n \log n)$ is faster.

6. (10 points) An algorithm takes 0.5ms for an input of size 100. Assuming that lower-order terms are negligible, how long will it take for an input of size 500 if the running time is:

   (a) linear, $O(n)$?

   $$\frac{500}{100} = \frac{N}{.5ms}$$

   Solving for N you get 2.5ms.

   (b) cubic, $O(n^3)$?

   $$\frac{500^3}{100^3} = \frac{N}{.5ms}$$

   Solving for N you get 62.5ms.

7. (10 points) An algorithm takes 0.5ms for an input of size 100. Assuming that lower-order terms are negligible, how large a problem can be solved in 1 minute if the running time is:

   (a) $O(n \log n)$?
   Since 1 minute is 60,000ms

   $$\frac{xlogx}{100log100} = \frac{60,000}{0.5ms}$$

   Solving for x you get an input size of rougly 3,650,000. Using base 10 for log.

   (b) quadratic, $O(n^2)$?

   $$\frac{x^2}{100^2} = \frac{60,000}{0.5ms}$$

   Solving for x you get an input size of rougly 34700.

8. (20 points) Find running time functions for the algorithms below and write their classification using Big-O asymptotic notation. A 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 $n \geq 1$ integers.
  Output: The sum of the elements in A.
$s \leftarrow A[0]$
for $i \leftarrow 1$ to $n-1$ **do**
  $s \leftarrow s + A[i]$
end for
return s

The running time function is 1+(n-1), since there is 1 assignment operation performed before, the loop is done n-1 times, and there is 1 operation in the loop. The big classifcation is O(n).

Algorithm Ex2(A):
  Input: An array A storing $n \geq 1$ integers.
  Output: The sum of the elements at even positions in A.
$s \leftarrow A[0]$
for $i \leftarrow 2$ **to** $n-1$ **by** increments of 2 **do**
  $s \leftarrow s + A[i]$
end for
return s

The running time function is 1+($\frac{n-1}{2}$), since there is 1 assignment before, and it increments by 2. So the Big-O notation is O(n) still since constants are ignored.

Algorithm Ex3(A):
  Input: An array A storing $n \geq 1$ integers.
  Output: The sum of the partial sums in A.
$s \leftarrow 0$
for $i \leftarrow 0$ **to** $n-1$ **do**
  $s \leftarrow s + A[0]$
  for $j \leftarrow 1$ **to** i **do**
    $s \leftarrow s + A[j]$
  end for
end for
return s

The running time function is 1+$\frac{(n+1)n}{2}$, since there is 1 operation performed before, the outer loop has 1 operation, and the inner loop has 1 operation. The Big-O for the algorithm is $O(n^2)$.

Algorithm Ex4(A):
  Input: An array A storing $n \geq 1$ integers.
  Output: The sum of the partial sums in A.
$t \leftarrow 0$
$s \leftarrow 0$
for $i \leftarrow 1$ **to** $n-1$ **do**
  $s \leftarrow s + A[i]$
  $t \leftarrow t + s$
end for
return t

The running time for the function is 2(n-1)+1, because there is two operations performed each loop on s and there is 1 operation performed before the loop on s, and it loops n-1 times. The Big-O notation is O(n).