CSCE 221 Cover Page

Homework #1

Due February 10 at midnight to eCampus

Ryan Wenham 627002098

ryanjw0903@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	Sam Jefferies	
Web pages (provide URL)	C++.com Geeksofgeeks.com	
Printed material	Your notes	
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 Ryan Wenham Date 2/2/20

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?

```
#include <iostream>
#include <vector>
using namespace std;
int main() {
     vector<double> v1 {1,4,6,8,9};
     vector<double> v2 {10,6,2,20,4};
     vector<double > v3;
     for(int i = 0; i<v1.size(); i ++){
       v3[i] = v1[i]*v2[i];
     }
     for(int i = 0; i<v3.size(); i ++){
       cout << v3[i] << " ";
     }
   }
```

   (a) There is n number of multiplications and n is based on size of vector c. The eqaiton if f(n) = n

   (b) Big O notaion is O(N)

1. (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?

```
#include <iostream>
#include <vector>
using namespace std;
int main() {
   bool odd = false;
   vector<int> v1 {2,4,6,8,2};
   for(int i = 0; i<v1.size(); i ++) {
      for (int j = 1; j < v1.size(); j++) {
        if (v1[i] * v1[j] % 2 != 0) {
          odd = true;
          break;
        }
      }
   }
   cout << odd;
}
```

(a) The formula for scalre multipications f(n) = n(n-1)/2

(b) Best Case is the first multiplication betwwen v[0] and v[1] is odd. f(n) = 1. Input ex is {1,3,5,8,9}, will take 1 multpiclation bc 1 times 3 is odd.

(c) Worst Case is takes till v[size-2] and v[size-1} to be the odd case, meaning it will go through all multipications possbile. The f(n) = n(n-1)/2. Input ex is {2,2,4,3,3}.

(d) For best case Big-Oa is O(n) and worst the Big-O is O(n^2)

2. (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?

```
#include <iostream>
#include <vector>
using namespace std;
template <typename T>
T binarySearch(T arr[], int l, int r, T x) {
if (r >= l) {
   int mid = l + (r - l) / 2;
   if (arr[mid] == x)
      return mid;
      if (arr[mid] > x)
        return binarySearch(arr, l, mid - 1, x);
return binarySearch(arr, mid + 1, r, x); }
return -1;
}
int main() {
   vector<T> v1 {2,4,6,8,2};
   binarySearch(v1,0,v1.size(),8);
}
```

(a) Formula for number of comparions is f(n) = log n.

(b) For best case the middle vaule of the vector is the value wanted there for only one comparssion is needed.

(c) The worst case is that you have to take log n comparrsions to find the value

(d) Best case big-O is O(l) and worst case big-O is O(log n)

1. (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 $n \geq n_0$

   (a) 8n log n = 2n^2

   (b) 4 log n = n

   (c) 4 = n /logn

   (d) n=16

   (e) So n_0= 17 since n >=17 for A to be faster.

2. (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.

   (a) For n<100 n^2 will prove to be fatser has it iterates faster over its loop, while as the data size gets bigger an bigger over 100, O(n log n) that takes slower to iterate will be faster .

3.

4. (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, $O(n)$?

      i. 500/100 = N/.5

      ii. 2.5 ms

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

      i. 500^3/(100^3) = N/.5

      ii. 62.5ms

   (c)

5. (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) $O(n \log n)$?

      i. x log x / 100 log100 = 60,000/.5

      ii. x = 12,000,000

      iii.

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

      i. x^2 / 100^2 = 60,000/.5

      ii. x = 34,641

(c)

6. (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 ≥ 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
f(n) = 2n+2
Big O = 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
f(n) = 2(n/2) + 2
Big O = 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
```

```
Big O = O(n^2)
```

```
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
f(n) = 4n+3
Big O = O(n)
```