Name: Tuan Ngo

CWID: 887416766

CPSC 355

Project04: Dynamic versus Exhaustive

1. **Dynamic Programming**
   Pseudocode:
   double max(double a, double b):
           if (a < b):   //1 T.U.
                   return b
           else:
                   return a
   s.c = max (1,0) = 1

   unique_ptr<ArmorVecctor> dynamic_max_defense(const ArmorVector& armors,
                                                   int total_cost) :
           unique_ptr<ArmorVector> todo(new ArmorVectotr(armors))   // 1 T.U.
           unique_ptr<ArmorVector> result(new ArmorVector)   // 1 T.U.
           EPSILON = 0.001   // 1 T.U.
           cost=0   // 1 T.U.
           vector<vector<double>> dp(todo->size()+1,vector<double> (total_cost+1)) // 1 T.U.
           for i=0 to (todo->size()) do: //Analysis later
                   for cost=0 to total_cost do: // ANALYSIS LATER
                           if (i==0 || cost ==0):   // 3 T.U.
                                   dp[i][cost]=0   // 1 T.U.
                           endif
                           else if (todo->at(i - 1)->cost() <= cost) :   // 1 T.U.
                                   dp[i][cost] = max(todo->at(i - 1)->defense() + dp[i - 1][cost -
                                           todo->at(i-1)->cost()]) , dp[i-1][cost]) // 3 T.U.
                           endelseif
                           else:
                                   d[i][cost]=dp[i-1][cost]   // 2 T.U.
                           endelse
                   endfor
           endfor
           defense = dp[todo->size()][total_cost]   // 1 T.U.
           cost = total_cost   // 1 T.U.
           for i=todo->size() down to 1 do: // ANALYSIS LATER
                   if(defense <= 0):   // 1 T.U.
                           break   // 1 T.U.
                   endif
                   if(std::abs(defense – dp[i-1][cost]) < EPSILON) :   // 2 T.U.
                           continue   // 1 T.U.

```
                    endif
                    if (defense != dp[i-1][cost]:                              // 2 T.U.
                            result->push_back(todo->at(i-1)                    // 2 T.U.
                            defense = defense – todo->at(i-1)->defense()       // 3 T.U.
                            cost =  cost – todo->at(i-1)->cost()               // 3 T.U.
                    endif
            endfor
            return result
```

Analysis:

Assume: the size of armos = n

m = total_cost

Functions at(),cost(),defense() in class Armorvector do not count time unit

Lookup vector array/vector/list does not count time unit

max function = 1 T.U.

std::abs() function does not count time unit.

Total_cost = unlimited

Step count:

s.c = $5 + (\sum_0^n (\sum_0^m 3 + \max(1, 1 + \max(3,2))) + (2 + \sum_1^n(1 + \max(1,0) + 2 + \max(1,0) + 2 + \max(8,0)))$

Calculate:

$\sum_0^n (\sum_0^m 3 + \max(1, 1 + \max(3,2)))$

$= \sum_0^n (\sum_0^m 7$

$= \sum_0^n (7*(m+1))$

$= 7*(m+1)(n+1)$

$= 7* (mn + m + n + 1)$

$= 7mn + 7m + 7n + 7$

$\sum_1^n(1 + \max(1,0) + 2 + \max(1,0) + 2 + \max(8,0)))$

$= \sum_1^n(1 + 1 + 2 + 1 + 2 + 8)$

$= 15n$

s.c = 5 + 7mn + 7m + 7n + 7 + 2 + 15n

s.c = 7mn + 7m + 22n + 14

Time complexity: O(n*m)

Prove:

Show 7mn + 7m + 22n + 14 belong to O(n*m)

Find c >= 0 and n_0 >=0 such that 7mn + 7m + 22n + 14 <= c * n*m withall n*m >= n_0

Choose c = 50

7mn + 7m + 22n + 14 <= 50 mn

7mn + 7m + 22n + 14 <= 7mn + 7mn + 22mn + 14mn

7mn – 7mn + 7mn – 7m+ 22mn – 22n + 14mn - 14 >=0

n*m >=1 and n_0 = 1

2. **Exhaustive**

   Pseudocode:

   psedocode of sum_armor_vector:

   def sum_armor_vector(const ArmorVector& armors,double& total_cost,double& total_defense):

           total_cost = total_defense = 0;   // 1 T.U.

           for i in armors do: // armors.size()

                   total_cost += armor->cost();   // 2 T.U.

                   total_defense += armor->defense();   // 2 T.U.

           endfor

   // This is a void function() => does not need return.

   assume : the size of armors = m

   Step count for above function:

   s.c = 1 + $\sum_1^m$ 4= 4m +1

   Unique_ptr<ArmorVector> exhaustive_max_defense(const ArmorVector& armors,

                                         double total_cost):

           const n = armors.size()   // 1 T.U.

           assert(n < 64)   // 1 T.U.

           unique_ptr<ArmorVector> best(new ArmorVector)   // 1 T.U.

           unique_ptr<ArmorVector> candidate (new ArmorVector)   // 1 T.U.

           candidate_cost = 0   // 1 T.U.

           candidate_defense = 0   // 1 T.U.

           best_armor = 0;   // 1 T.U.

           for i =0 to (2^n -1) do: // Analysis later

                   candidate->clear()   // 1 T.U.

                   for j = 0 to n-1 do: // Analysis later

                         if (((i>>j) & 1) == 1):   // 3 T.U.

                             candidate->push_back(armors[j]) // 1 T.U.

                         endif

                   endfor

                   sum_armor_vector(*candidate, candidate_cost, candidate_defense) //4m+1 t.u.

                   if candidate_cost <= total _cost :   // 1t.u.

                         if best == NULL || candidate_defense > best_armor :   //3 t.u.

                             *best = *candidate   // 1 t.u.

                             best_armor = candidate_ defense   // 1t.u.

                         endif

                   endif

           endfor

           return best

Analysis:

        Assume: the size of armos = n

Functions at(),cost(),defense() in class Armorvector do not count time unit

Lookup vector array/vector/list does not count time unit

Erase function in vector = 1 T.U.

Total_cost = unlimited

Assert(n<64) = 1 T.U.

The size of the candidate depends on the size of armors. Let size of candidate = m and m <=n

Step Count:

$$\text{s.c} = 7 + \sum_0^{2^n - 1} \ (\ 1 + (\sum_0^{n-1} \ 3 + max\ (1,0)\ ) + (4m+1) + 1 + max(\ 3 + max(2,0)\ )\ )$$

calculate:

$$\sum_0^{n-1} \ 3 + max\ (1,0)\ )\ = \sum_0^{n-1}\ 4 = 4n$$

$\text{s.c} = 7 + \sum_0^{2^n - 1} \ (\ 1 + (\sum_0^{n-1}\ 3 + max\ (1,0)\ ) + (4m+1) + 1 + max(\ 3 + max(2,0)\ ,\ 0)\ )$

$\text{s.c} = 7 + \sum_0^{2^n - 1} \ (\ 1 + 4n + 4m + 1 \ + 1 + max(\ 3 + max(2,0)\ ,\ 0\ )\ )$

$\text{s.c} = 7 + \sum_0^{2^n - 1} \ (\ 1 + \ 4n + \ 4m \ + 1 \ + \ 1 \ + (\ 3 + 2\ ,0)$

$\text{s.c} = \ 7 + \sum_0^{2^n - 1} \ (\ 1 + \ 4n + \ + 4m \ + \ 1 \ + \ 1 \ + 5)$

$\text{s.c} = 7 + \sum_0^{2^n - 1} \ 4n + 4m \ + \ 8$

$\text{s.c} = 7 + (\ 4n + 4m + 8\ )\ (\ 2^n)$

$\text{s.c} = 7 + 4n*2^n + 4m*2^n \ + 8*2^n$

m <= n because the size of the candidate is less or equal than the size of the armors. Maximum of m = n

=> Time complexity: $O(2^n * n\ )$

Prove:

Show $7 + 4n*2^n + 4m*2^n + 8 * 2^n$ belong to $O(2^n * n)$

m <= n because the size of the candidate depends on the size of the armors. Maximum of m = n

=> $7 + 4n*2^n + 4m*2^n + 8 * 2^n$

= $7 + \ 8n*2^n + 8 * 2^n$

Find c >= 0 and n_0 >=0 such that $7 + 8n * 2^n + 8 * 2^n <= c * 2^n *n$ with all $n >= n_0$

Choose c = 23

$8n * 2^n + 8 * 2^n + 7 <= 23 * 2^n * n$

$8n * 2^n + 8 * 2^n + 7 <= 8 * 2^n * n + 8 * 2^n * n + 7 * 2^n * n$

$8 * 2^n * n - 8n * 2^n + 8 * 2^n * n - 8 * 2^n + 7 * 2^n * n - 7 >= 0$
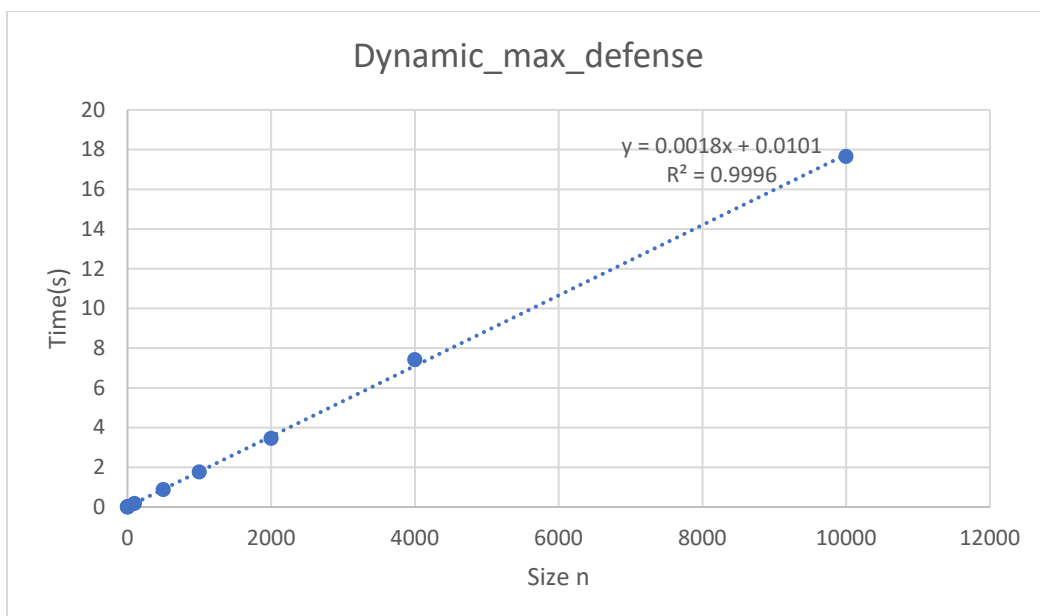
$n >= 1$ and $n_0 = 1$

Scatter Chart:

This is the data for size and time for both algorithms:
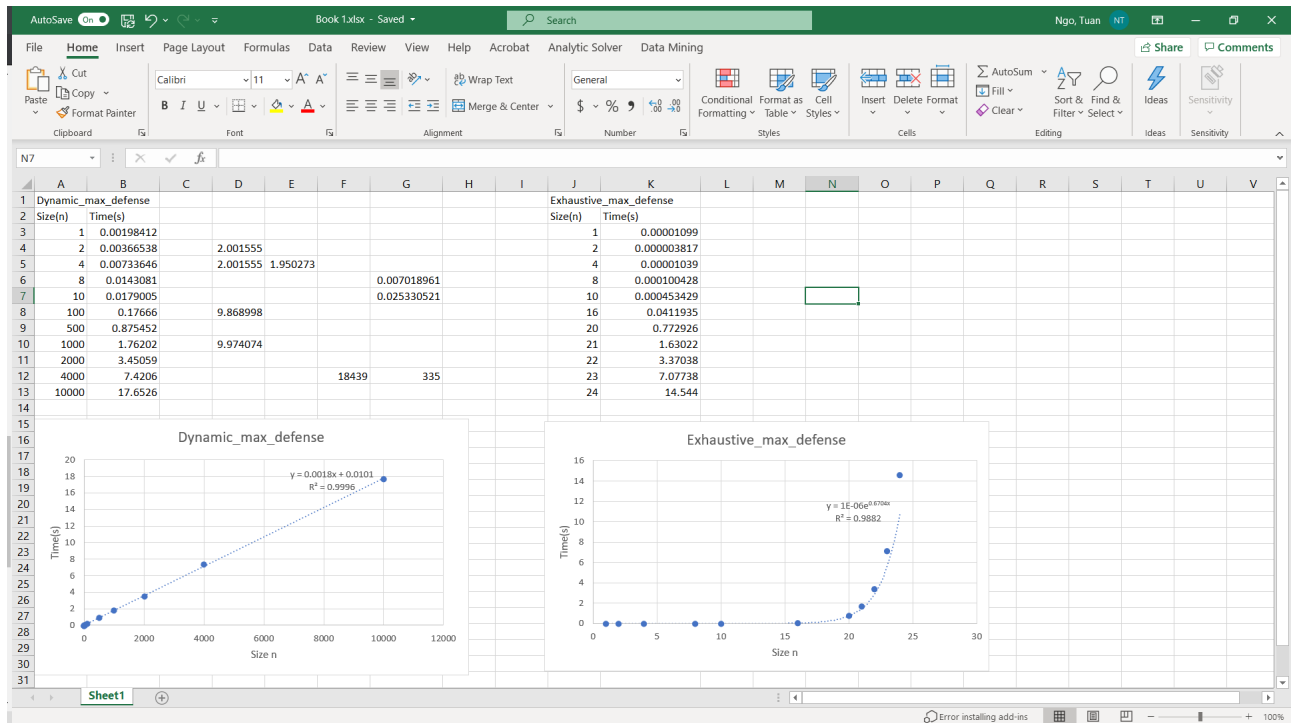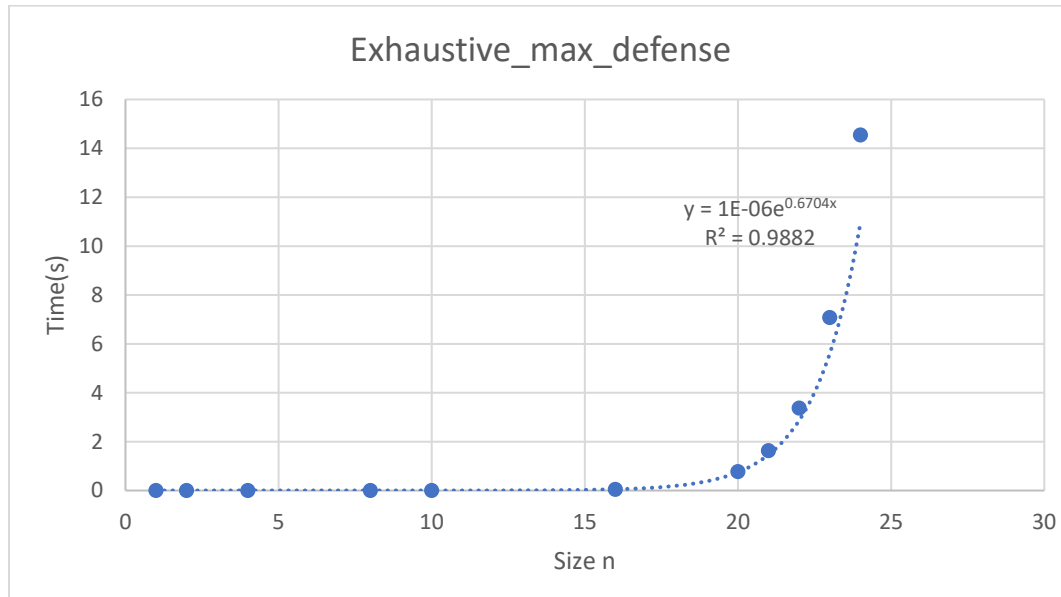

This is the scatter chart for the Dynamic_max_defense function:

The empirically-observed time efficiency data is consistent with my mathematically-derived big-O efficiency class for the dynamic_max_defense function.

This is the scatter chart for exhaustive_max_defense function:

The empirically-observed time efficiency data is consistent with my mathematically-derived big-O efficiency class for the exhaustive_max_defense function.

Answers to the following questions, using complete sentences.
a. Is there a noticeable difference in the performance of the two algorithms? Which is faster, and by how much? Does this surprise you?

Yes, there is a noticeable difference in the performance of the two algorithms. The dynamic_max_defense function is faster. The time complexity of the dynamic_max_defense function is $O(n*m)$, and the time complexity of enhaustive_max_defense is $O(n* 2^n)$. Dynamic_max_defense is faster than exhaustive_max_defense by $(n*2^n)/ (n*m)$ with n is the size of items in armors and m is the total_cost. It does surprise me because the exhaustive_max_defense function is so slow.

b. Are your empirical analyses consistent with your mathematical analyses? Justify your answer.

Yes, my empirical analyses are consistent with my mathematical analyses. I have increased the range a little more to see the difference in time for the dynamic_max_defense function. In contrast to the dynamic_max_defense function, a small change for the exhaustive function's size makes a big difference for the result.

c. Is this evidence consistent or inconsistent with hypothesis 1? Justify your answer.

Yes, this evidence is consistent with hypothesis 1. Based on my evidence from data and the scatter chart of dynamic_max_defense, the dynamic_max_defense function data almost fit the linear trendline. It means my evidence is consistent with hypothesis 1.

d. Is this evidence consistent or inconsistent with hypothesis 2? Justify your answer.

Yes, this evidence is consistent with hypothesis 2. Based on my evidence from data and the scatter chart of exhaustive_max_defense function, we can see it almost fits the exponential trendline. It means my evidence is consistent with hypothesis 2.