Name: Tuan Ngo

CWID: 887416766

CPSC 355

Project02: Greedy versus Exhaustive

1. **Greedy**
   Pseudocode:

```
unique_ptr<ArmorVecctor> greedy_max_defense(const ArmorVector& armors,
                                                    double total_cost) :
    unique_ptr<ArmorVector> todo(new ArmorVectotr(armors))      // 1 T.U.
    unique_ptr<ArmorVector> result( new ArmorVector)            // 1 T.U.
    result_gold = 0                                             // 1 T.U.
    max = 0                                                     // 1 T.U.
    index = 0                                                   // 1 T.U.
    c=0                                                         // 1 T.U.
    while( !todo->empty() && result_gold < total_cost):         //Analysis later
        max = 0                                                 // 1 T.U.
        index = 0                                               // 1 T.U.
        for i=0 to (todo->size() -1) do:                        // ANALYSIS LATER
            if (max <= todo->at(i)->defense() / ( todo->at(i)->cost() ) ): // 2 T.U.
                index = I                                                  // 1 T.U.
                max = todo->at(i)->defense() / ( todo->at(i)->cost() ) //2 T.U.
            endif
        endfor
        c = todo->at(index)->cost()                             // 1 T.U.
        if ( (result_gold + c) <= total_cost):                  // 2 T.U.
            result->push_back(todo->at(index) )                 // 1 T.U.
            result_gold = result_gold + c                       // 2 T.U.
        endif
        todo->erase(todo->begin() + index)                      // 1 T.U.

    endwhile

    return result
```

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

Step count:

s.c = 6 + $\sum_0^{n-1}$ 2 +($\sum_0^{n-1}$ $2 + max\ (3,0)$) + 1 + 2 + max(3,0) + 1

Calculate:

$\sum_0^{n-1}$ $2 + max\ (3,0)$ = $\sum_0^{n-1}$ 5 = 5n

$\sum_0^{n-1}$ 2 +($\sum_0^{n-1}$ $2 + max\ (3,0)$) + 1 + 2 + max(3,0) + 1

= $\sum_0^{n-1}$ 2 + 5n + 1 + 2 + 3 + 1

=$\sum_0^{n-1}$ $5n + 9$

= (5n + 9) n

= $5n^2$ +9

s.c = 6 + $\sum_0^{n-1}$ 2 +($\sum_0^{n-1}$ $2 + max\ (3,0)$) + 1 + 2 + max(3,0) + 1

s.c = 6 + $5n^2$ + 9

s.c= $5n^2$ + 15

Time complexity: O(n^2)

Prove:

Show $5n^2$ + 15 belong to O(n^2)

Find c >= 0 and n_0 >=0 such that $5n^2$ + 15 <= c * n^2 withall n >= n_0

Choose c = 20

$5n^2$ + 15 <= $20n^2$

$5n^2$ + 15 <= $5n^2$ + $15n^2$

$5n^2$ – $5n^2$ + $15n^2$ -15 >=0

n >=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:

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

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))

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

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

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

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

s.c = 7 + (4n + 4m + 8) (2^n)

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 withall n >= n_0

Choose c = 23

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

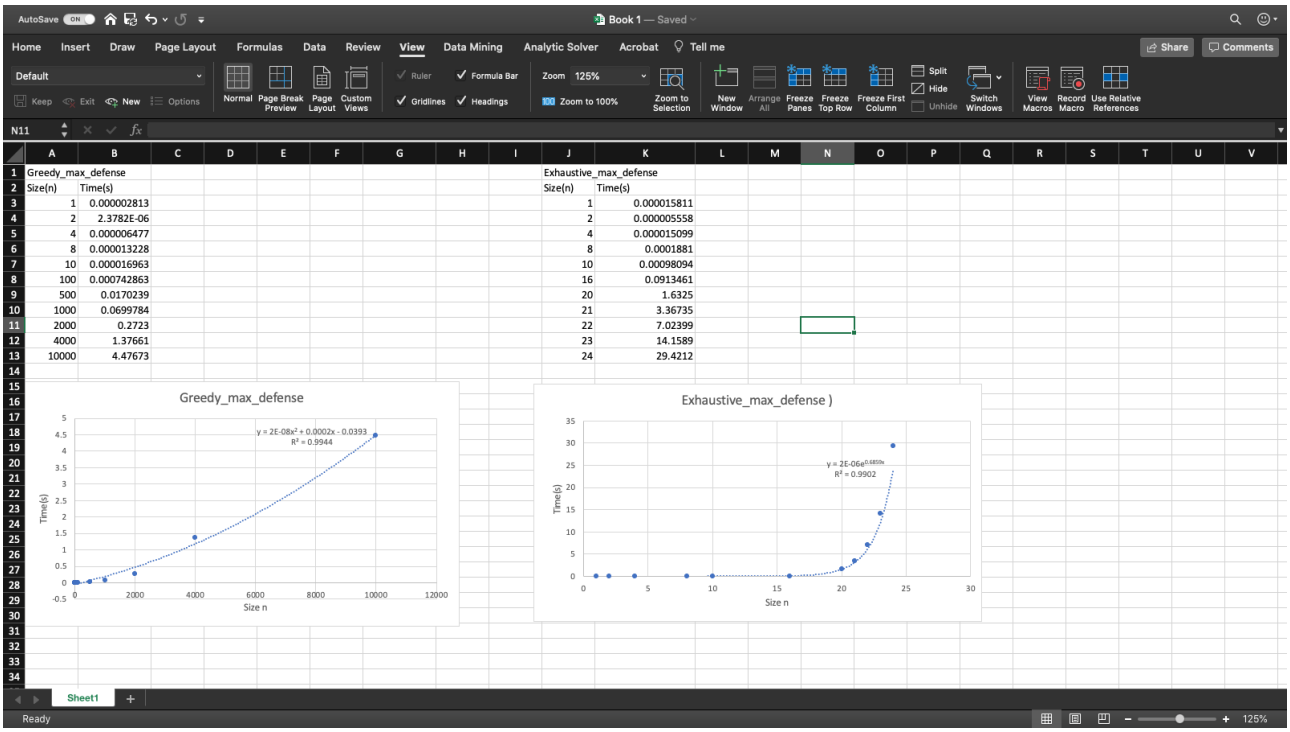
8n * 2^n + 8 * 2^n +7 <= 8 * 2^n * n + 8 * 2^n * n + 7 * 2^n * n

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

$n >= 1$ and $n_0 = 1$

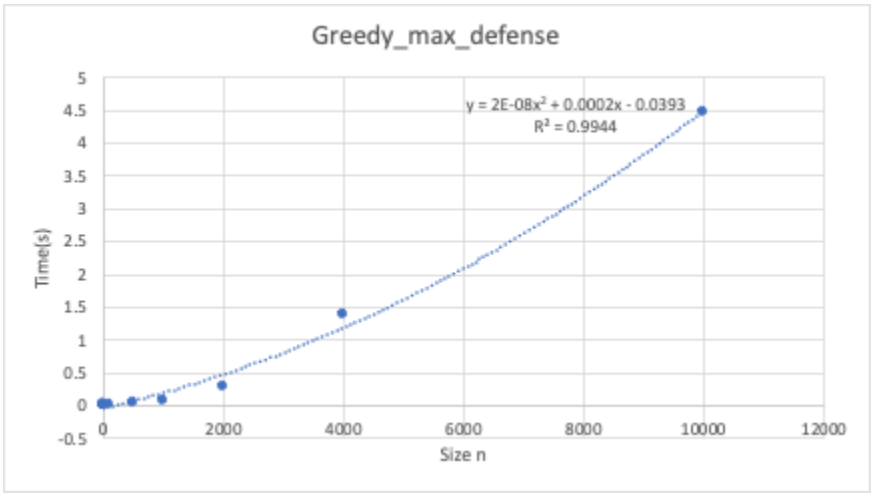
Scatter Chart:

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



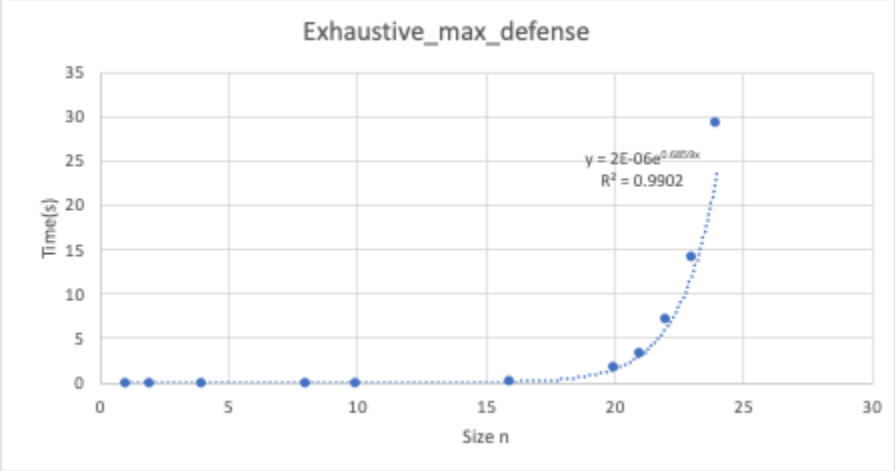
Greedy_max_defense		Exhaustive_max_defense	
Size(n)	Time(s)	Size(n)	Time(s)
1	0.000002813	1	0.000015811
2	2.3782E-06	2	0.000005558
4	0.000006477	4	0.000015099
8	0.000013228	8	0.0001881
10	0.000016963	10	0.00098094
100	0.000742863	16	0.0913461
500	0.0170239	20	1.6325
1000	0.0699784	21	3.36735
2000	0.2723	22	7.02399
4000	1.37661	23	14.1589
10000	4.47673	24	29.4212

This is the scatter chart for the Greedy_max_defense function:



The empirically-observed time efficiency data is consistent with my mathematically-derived big-O efficiency class for the greedy_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 greedy_max_defense function is faster. The time complexity of the greedy_max_defense function is O(n^2), and the time complexity of enhaustive_max_defense is O(n* 2^n). Greedy_max_defense is faster than exhaustive_max_defense by (n*2^n)/ (n^2). 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 greedy_max_defense function. In contrast to the greedy_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 greedy_max_defense, the greedy_max_defense function data almost fit the quadratic polynomial 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.