

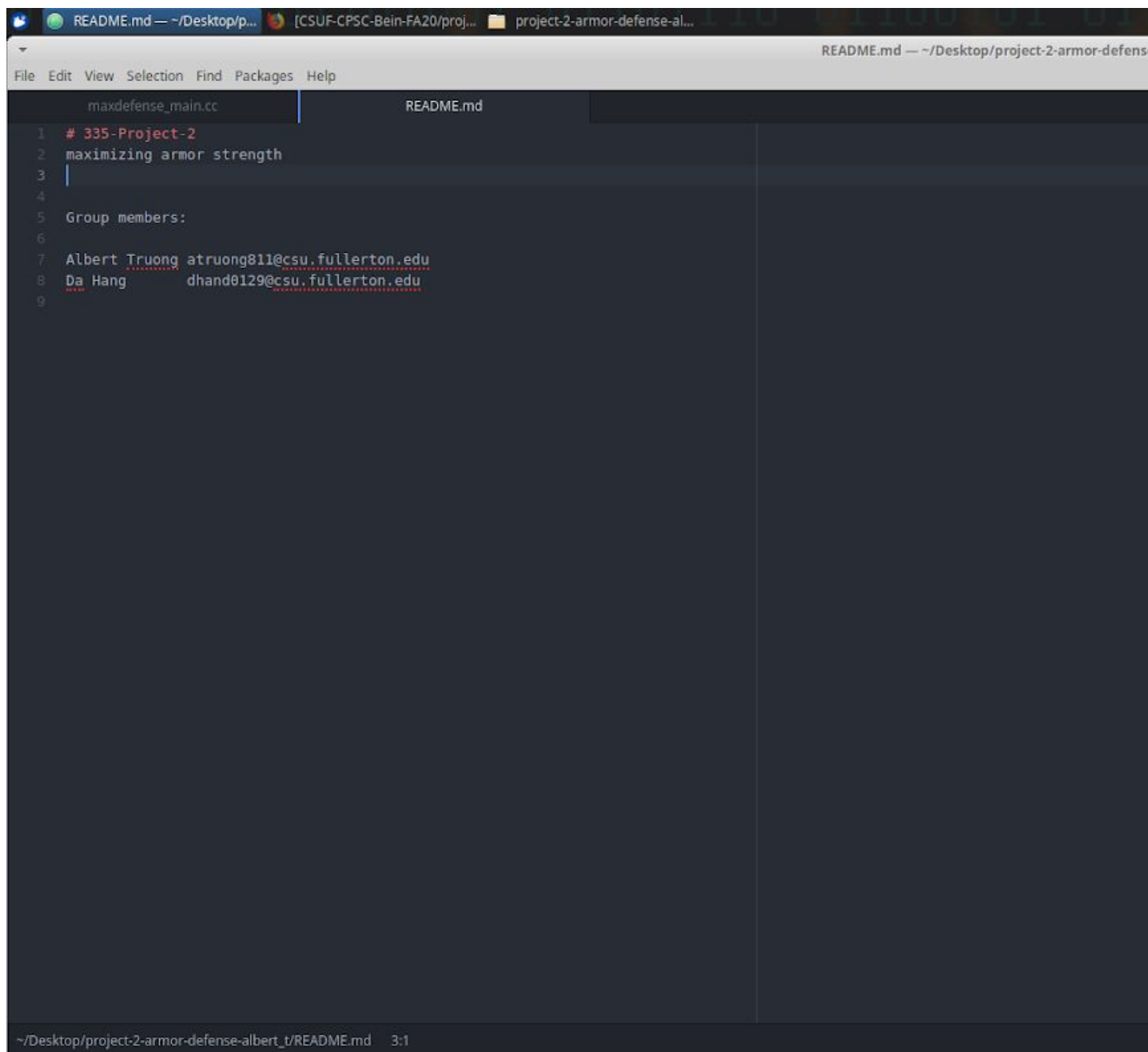
CPSC 335 Project 2 Analysis

Albert Truong

atruong811@csu.fullerton.edu

Da Hang

dhang0129@csu.fullerton.edu

A screenshot of a code editor window. The title bar shows the file path: README.md — ~/Desktop/p... [CSUF-CPSC-Bein-FA20/proj... project-2-armor-defense-al... The menu bar includes File, Edit, View, Selection, Find, Packages, and Help. The editor has two tabs: maxdefense_main.cc and README.md. The README.md tab is active, showing the following content:

```
1 # 335-Project-2
2 maximizing armor strength
3
4
5 Group members:
6
7 Albert Truong atruong811@csu.fullerton.edu
8 Da Hang      dhang0129@csu.fullerton.edu
9
```

The status bar at the bottom shows the file path: ~/Desktop/project-2-armor-defense-albert_t/README.md and the line number 3:1.

File Edit View Terminal Tabs Help

```
student@tuffix-vm:~/Desktop/project-2-armor-defense-albert_t$ make
```

```
g++ -std=c++17 -g maxdefense_main.cc -o experiment
```

```
./maxdefense_test
```

```
load armor database still works: passed, score 2/2
```

```
filter_armor_vector: passed, score 2/2
```

```
greedy_max_defense trivial cases: passed, score 2/2
```

```
greedy_max_defense correctness: passed, score 4/4
```

```
exhaustive_max_defense trivial cases: passed, score 2/2
```

```
exhaustive_max_defense correctness: passed, score 4/4
```

```
TOTAL SCORE = 16 / 16
```

```
student@tuffix-vm:~/Desktop/project-2-armor-defense-albert_t$ ./experiment
```

```
Starting to collect Greed Algorithm measurements
```

```
1 Items: 2.318e-06s
```

```
2 Items: 2.41e-07s
```

```
3 Items: 1.48e-07s
```

```
4 Items: 1.37e-07s
```

```
5 Items: 1.39e-07s
```

```
10 Items: 1.91e-07s
```

```
15 Items: 1.24e-07s
```

```
20 Items: 1.54e-07s
```

```
25 Items: 1.37e-07s
```

```
Starting to collect Exhaustive Optimization Algorithm measurements
```

```
1 Items: 1.91e-07s
```

```
2 Items: 1.77e-07s
```

```
3 Items: 1.51e-07s
```

```
4 Items: 1.34e-07s
```

```
5 Items: 1.44e-07s
```

```
10 Items: 1.47e-07s
```

```
15 Items: 2.1248e-05s
```

```
20 Items: 4.436e-06s
```

```
25 Items: 5.267e-06s
```

Greedy Algorithm:

todo = armor_items //SC: 1

double g;

int a;

result =empty vector //SC: 1

result_cost =0 //SC: 1

while todo is not empty: //SC: n

 For int x=0 to n-1 do //SC: (n-1)-0+1 = n

 If (defense(todo[x])/cost(todo[x]))> best &&

 (result_cost + cost(todo[x]))<=total_cost //SC:5

 Best = defense(todo[x])/cost(todo[x]);//SC: 2

 g=cost(todo[x]) //SC: 1

 a=x; //SC: 1

 End if

End for

If result_cost + cost(todo[a]) <= total_cost//SC: 2

 result.add_back(todo[a]) //SC: 1

 result_cost += g //SC: 1

End if

todo.erase(a) //SC: 1

End while

return result

SC: $1+1+1 + \text{SC}(\text{while})*\text{SC}(\text{for})$

$3+n(n(5+ \max(2+1+1,0)) + 2 + \max(2,0) + 1)$

$3 + n(n(5+4) + 2 + 2 + 1)$

$3 + 9n^2 + 5n = 9n^2 + 5n + 3$

$O(n^2)$

Exhaustive Optimized Algorithm:

$n = |\text{armor_items}|$ //SC: 1

best = None //SC: 1

double totalGold= 0.0 //SC: 1

totalDef=0.0 //SC: 1

totalDefB =0.0 //SC: 1

for bits = 0 to $(2^n - 1)$: //SC: $((2^n - 1) - 0 + 1) = 2^n$

candidate = empty vector //SC: 1

for j = 0 to n-1: //SC: $(n-1) - 0 + 1 = n$

if $(\text{bits} \gg j) \& 1 == 1$: //SC: 3

```

        candidate.add_back(armor_items[j]) //SC: 1

        sum_armor_vector(candidate,totalGold,totalDef); //SC: 1

    End if

End for

if totalGold <= total_cost //SC: 1

    if best is None || totalDef > totalDefB && candidate is not empty //SC: 3

        best = candidate //SC: 1

        totalDefB = totalDef //SC: 1

    End if

End if

End for

return best

```

SC: $1+1+1+1+1+2^n(1+n(3+\max(1+1,0)) + 1 + \max(3 + \max(1+1,0), 0))$

$5 + 2^n(1+n(3+2) + 1 + \max(3+2,0))$

$5 + 2^n(1+5n + 1 + 5)$

$5 + 2^n(7 + 5n)$

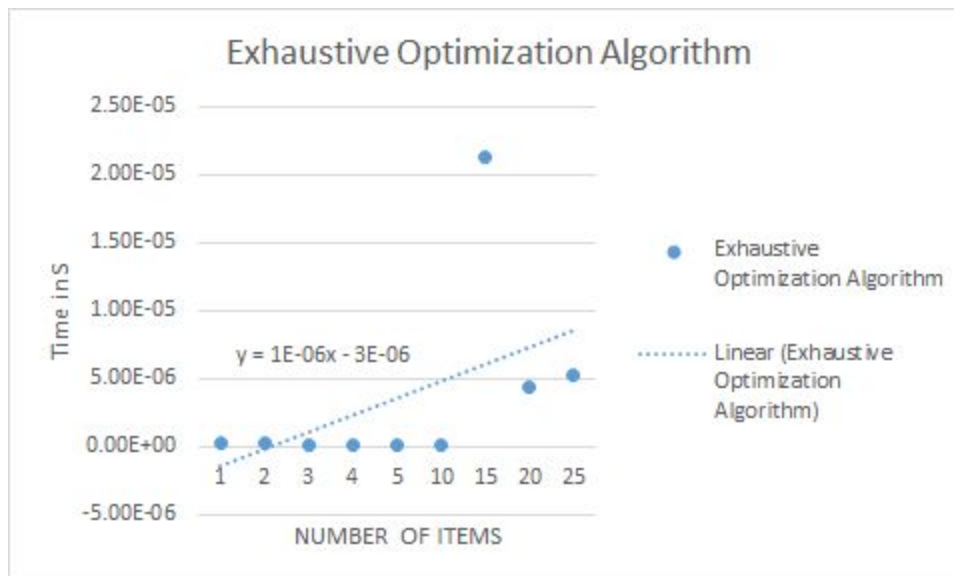
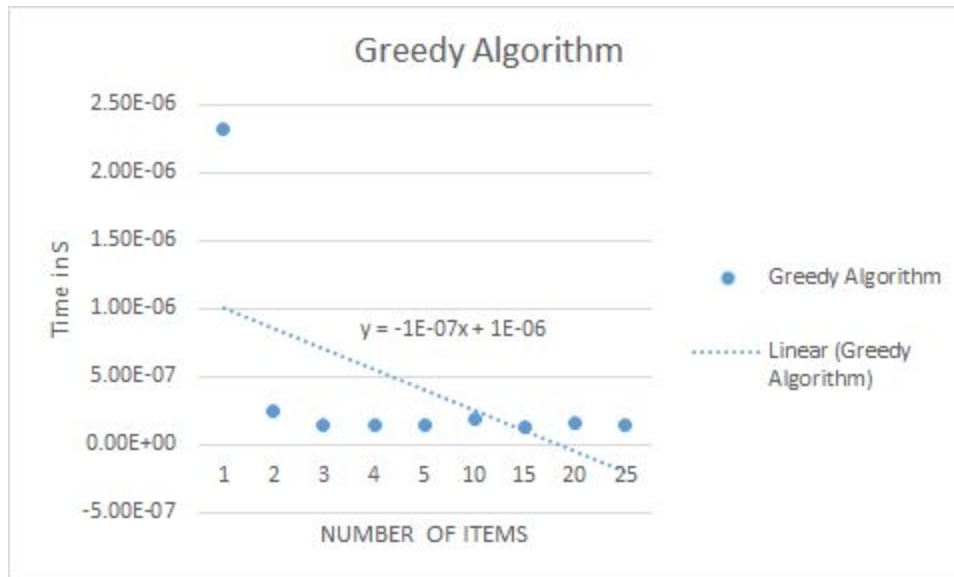
$= 5n(2^n) + 7(2^n) + 5$

$O(2^n * n)$

Scatter Plot Graphs:

Data:

	1	2	3	4	5	10	15	20	25
Greedy Algorithm	2.32E-06	2.41E-07	1.48E-07	1.37E-07	1.39E-07	1.91E-07	1.24E-07	1.54E-07	1.37E-07
Exhaustive Optimization Algorithm	1.91E-07	1.77E-07	1.51E-07	1.34E-07	1.44E-07	1.47E-07	2.12E-05	4.44E-06	5.27E-06



- a. There is a noticeable difference in the performance of the two algorithm. It only becomes noticeable once the size of n goes over 10. Other then the spike in 15 items in the

exhaustive search where it may be explained by a background process slowing the computer, at 20 or 25 items the process may be 40 times slower than the greedy algorithm. When comparing the mathematically-derived big O efficiency class for each algorithm $O(2^n * n)$ vs $O(n^2)$ it is not surprising that the $O(2^n * n)$ of the Exhaustive Optimization Algorithm would be the slower algorithm.

- b. The empirical analyses are not consistent with the mathematical analyses of the greedy algorithm but may be consistent with the exhaustive search algorithm. The reason for this is most likely due to the size of n chosen for the graphs. When doing the empirical analyses it was determined that the max size of n would be chosen to be 25 due to the time it would take to obtain the data above 25. That size may be too small to obtain a empirical analyses data that are consistent with the mathematical analyses.
- c. Based on the graph of the Greedy Algorithm, the empirically-observed time efficiency data is inconsistent, with the mathematically-derived big O efficiency class for the algorithm. It may be because of background process slowing the first test of n . The size of n may not be large enough to obtain an accurate look at the big O efficiency. The top size of n at 25 was chosen because above 25 the time it took for exhaustive optimization algorithm was too long.
- d. Based on the graph of the Exhaustive Optimization Algorithm, the empirically-observed time efficiency data is potentially consistent. As the size of n went up it a considerable amount was above 10. At size 15 there appears to be a spike that may have been caused by a background process slowing the computer. Above that at 20 and 25 both times were climbing which is relatively consistent with the big O efficiency class of $O(2^n * n)$. If the size of n went above 25, it would be graphically more consistent with the Big O, but due to amount of time it took above 25. It was determined that n of 25 would be the largest size of n .