Brian Tan - brian388@csu.fullerton.edu

Hammad Qureshi - qureshi434@csu.fullerton.edu

Project 2 Analysis

Greedy Max Defense – Pseudocode:

```
greedy_max_defense(G, armor_items):
        todo = armor_items    // 1tu
        result = empty vector  //1tu
        result_cost = 0                    //1tu

        //bubblesort to sort armor in todo vector from highest defense/cost to lowest.
        for each i in todo       //  same as for i =0 to n -1
                for each j in todo   // same as for j = 0 to n - i - 1
                        if((todo[j]->defense / todo[j]->cost) < todo[j+1]->defense / todo[j+1]->cost))
        //3tu
                                swap(todo[j], todo[j+1]) // same as using temp procedure. 3tu
                        endif
                endfor
        endfor

        while todo is not empty: // n tu   same as while n > 0
                shared_ptr a = todo[0]  // 1tu
                float g = a->cost   // 1tu
                todo.erase(a)  // 1tu    n shrinks by 1 every iteration
                if result_cost + g <= G then  //2tu
                        result.push_back(a)  //1tu
                        result_cost += g   //1tu
        endwhile
        return result
```

Time Complexity:

bubblesort

$$\sum_{i=0}^{n-1} \sum_{j=0}^{n-i-1} 6$$

$$\sum_{j=0}^{n-i-1} 6 = 6(n-i-1+1) = 6n - 6i$$

$\sum_{i=0}^{n-1} 6n - 6i = \sum_{i=0}^{n-1} 6n - \sum_{i=0}^{n-1} 6i$

$6n(n-1+1) - 6 \frac{n-1(n-1+1)}{2} = 6n^2 - 3(n^2 - n)$

$= 6n^2 - 3n^2 + 3n = 3n^2 + 3n$

While loop:

$n * 7 = 7n$

Total:

$3 + 3n^2 + 3n + 7n = 3n^2 + 10n + 3$

$\therefore O(n^2)$

Proof $3n^2 + 10n + 3 \in O(n^2)$

find $C > 0$ and $n_0 \geq 0$ such that $3n^2 + 10n + 3 \leq C * n^2 \ \forall \ n_0$

$C = 3 + 10 + 3 = 16$

$3n^2 + 10n + 3 \leq 16n^2$

$16n^2 - 3n^2 - 10n - 3 \geq 0$

$13n^2 - 10n - 3 \geq 0$ true for $n = 1$

choose $n_0 = 1$

$\therefore 3n^2 + 10n + 3 \in O(n^2)$

Exhaustive Max Defense – Pseudocode:

```
exhaustive_max_defense(G, armor_items):
        n = armor_items.size()  //1tu
        best = None                  // 1tu
        end = pow(2,n)           //1tu
        for bits = 0 to (end-1):  // 2^n - 1 tu
                candidate = empty vector  // 1tu
                for j = 0 to n-1:                    // n tu
                        if ((bits >> j) & 1) == 1:  // 2tu
                                candidate.add_back(armor_items[j])  // 1tu
                endfor
                if total_gold_cost(candidate) <= G: // 1tu
                        if best == None or total_defense(candidate) > total_defense(best): // 2tu
                                best = candidate //1tu
                        endif
                endif
        endfor
        return best
```

Time Complexity

inner for loop:

$\frac{end - start}{step} + 1$

$\frac{n-1-0}{1} + 1 = n$

$n * 3 = 3n$

outer for loop:

$\frac{2^n - 1 - 0}{1} + 1 = 2^n$

$2^n(1 + 3n + 1 + 2 + 1) = 2^n(3n + 5)$

Total:

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

$= 3 + 2^n (3n + 5)$

$\therefore O(2^n n)$

Proof $3 + 2^n (3n + 5) \in O(2^n n)$

L'Hopital

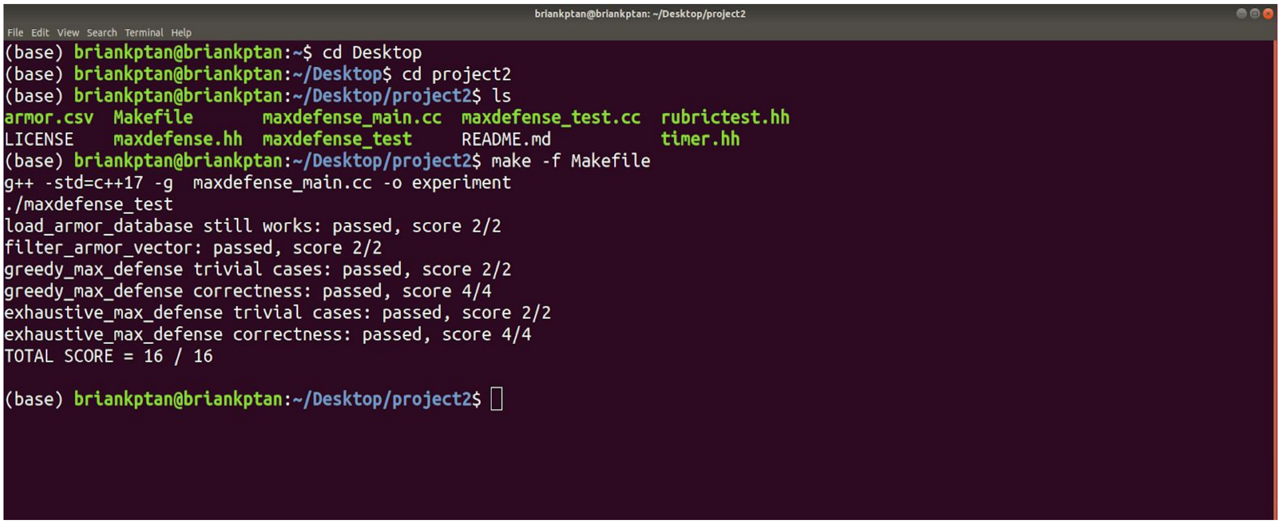
$$\lim_{n\to\infty} \frac{f'(n)}{g'(n)} = \lim_{n\to\infty} \frac{(3 + 2n\,(3n + 5))'}{(2^n\, n)'} = \lim_{n\to\infty} \frac{2^n(3ln(2)n + 5ln(2) + 3)}{2^n(ln(2)n + 1)}$$

$$= \lim_{n\to\infty} \frac{(3ln(2)n + 5ln(2) + 3)}{(ln(2)n + 1)} = \lim_{n\to\infty} \frac{ln(2)n\,(3 + \frac{5}{n} + \frac{3}{ln(2)n})}{ln(2)n(1 + \frac{1}{ln(2)n})}$$

$$= \lim_{n\to\infty} \frac{(3 + \frac{5}{n} + \frac{3}{ln(2)n})}{(1 + \frac{1}{ln(2)n})} = \frac{3}{1} = 3 \geq 0\text{, constant, and not } \infty$$

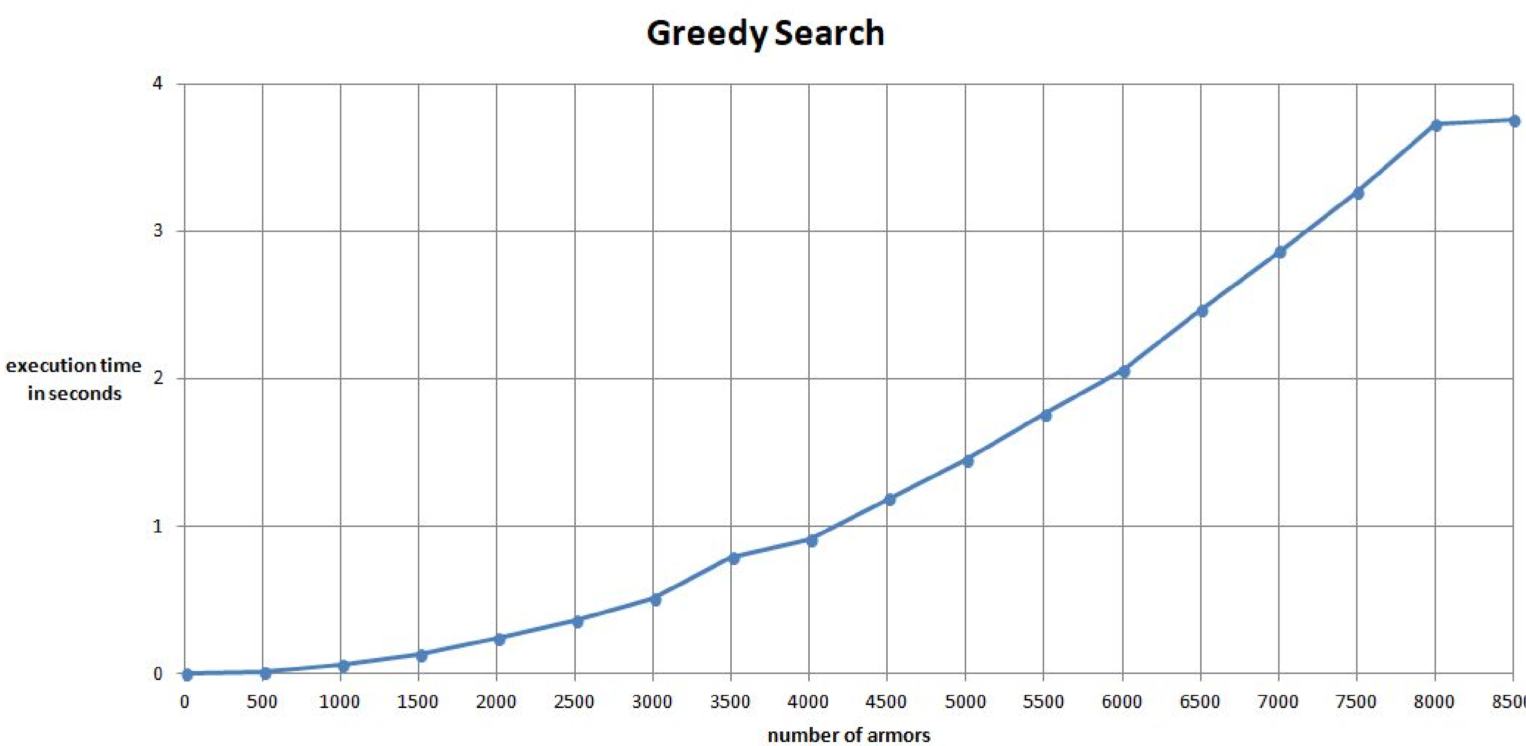
$\therefore 3 + 2^n (3n + 5) \in O(2^n n)$

Screenshot


briankptan@briankptan: ~/Desktop/project2
File Edit View Search Terminal Help
(base) briankptan@briankptan:~$ cd Desktop
(base) briankptan@briankptan:~/Desktop$ cd project2
(base) briankptan@briankptan:~/Desktop/project2$ ls
armor.csv Makefile maxdefense_main.cc maxdefense_test.cc rubrictest.hh
LICENSE maxdefense.hh maxdefense_test README.md timer.hh
(base) briankptan@briankptan:~/Desktop/project2$ make -f Makefile
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
(base) briankptan@briankptan:~/Desktop/project2$

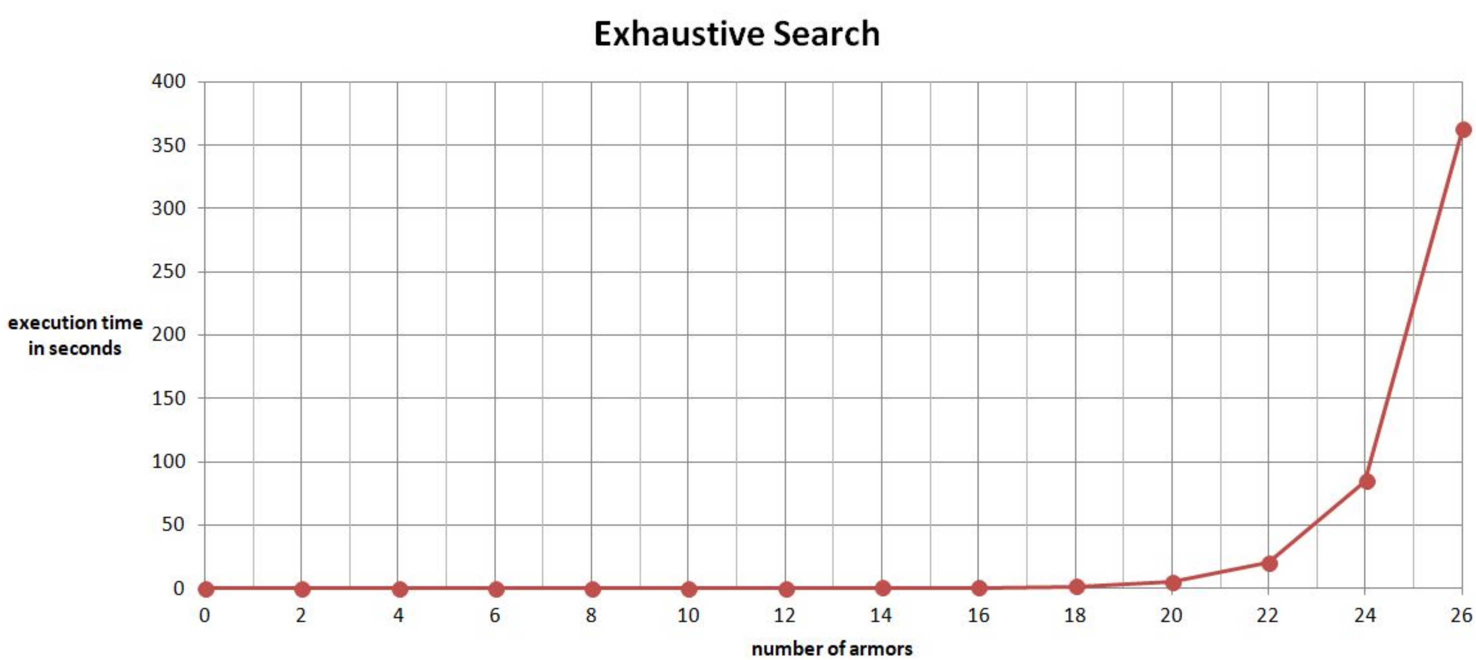
Greedy Search

n armors	time in sec			n armors	time in sec
0	0			4500	1.1843
500	0.0149			5000	1.4471
1000	0.0589			5500	1.7569
1500	0.1309			6000	2.0608
2000	0.2388			6500	2.4646
2500	0.3587			7000	2.8623
3000	0.5097			7500	2.8623
3500	0.7895			8000	3.7268
4000	0.9104			8500	3.7558



Exhaustive Search

n armors	time in sec			n armors	time in sec
0	0			14	0.06096
2	0			16	0.25862
4	0			18	1.19581
6	0			20	4.76593
8	0.001			22	20.1209
10	0.002			24	84.8068
12	0.01897			26	362.677



3a) 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. The Greedy search algorithm is exponentially faster than the Proper Exhaustive search. The time it takes for the exhaustive search algorithm to complete with n = 26 is in the 5 minute range compared to a mere 3 to 4 seconds it takes for the greedy search algorithm to complete with n = 8000.
- It does surprise us. While we did expect a difference in terms of wait time, we did not expect such a huge difference. The amount of time it would take for the exhaustive search to iterate through 30+ items, let alone 63, would be in the thousands of seconds. Possibly tens of thousands.

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

- Yes. Our mathematical analysis for the greedy search algorithm indicated a time complexity of $O(n^2)$. The empirical data reflected as such. The growth in time was gradual and slow.
- Analysis of the exhaustive algorithm indicated a time complexity of $O(2^n * n)$. The empirical data reflected this complexity as well. The growth in time shot up very quickly past a certain threshold.

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

- The evidence is partially consistent with hypothesis 1. It is not feasible to implement exhaustive search algorithms with a list size past a certain amount. With a small value of n, between 10 and 18, the algorithm performs adequately efficient.
- By design, the outputs produced are expected to be correct and more precise in terms of pinpointing the best item. But in terms of efficiency, the time it takes to perform the exhaustive search within a *reasonable* value of n, say n = 25, is too long to be considered feasible.

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

- The evidence is consistent with hypothesis 2. The proper exhaustive search algorithm we tested reflected precisely what was hypothesized; exponential running time, too slow to be of practical use. We had a list of over 8000 armor items. It would already be impractical to use the proper exhaustive search algorithm to find the best armor in a list of 60 armor items, let alone 8000.