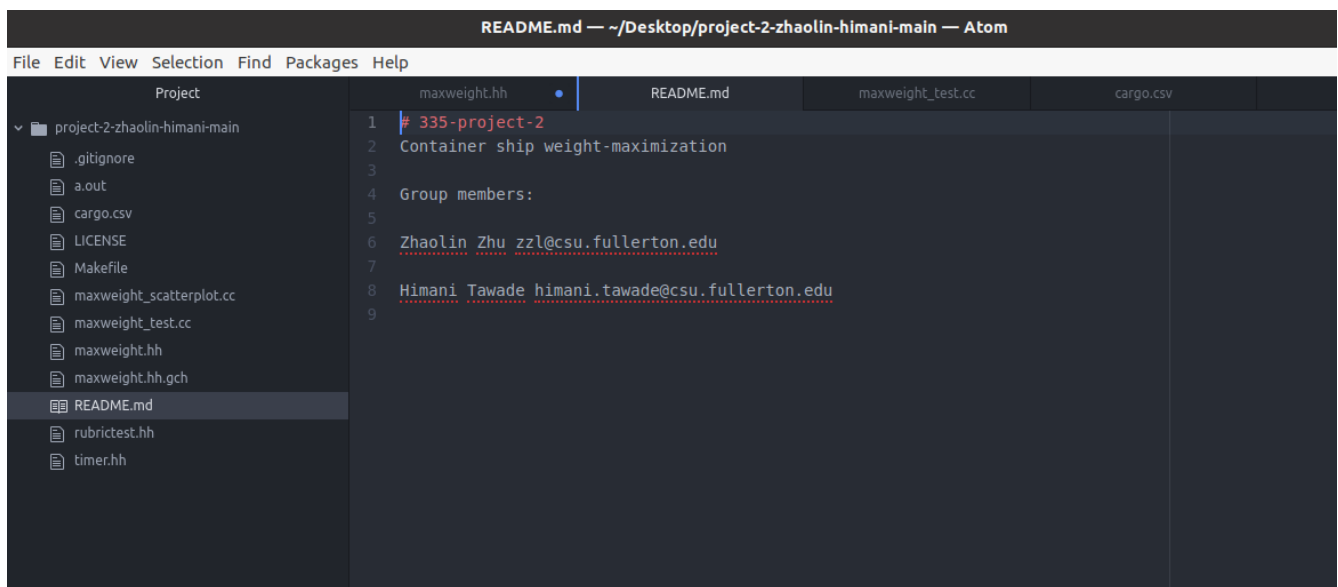


PROJECT 2 REPORT

Group Members:

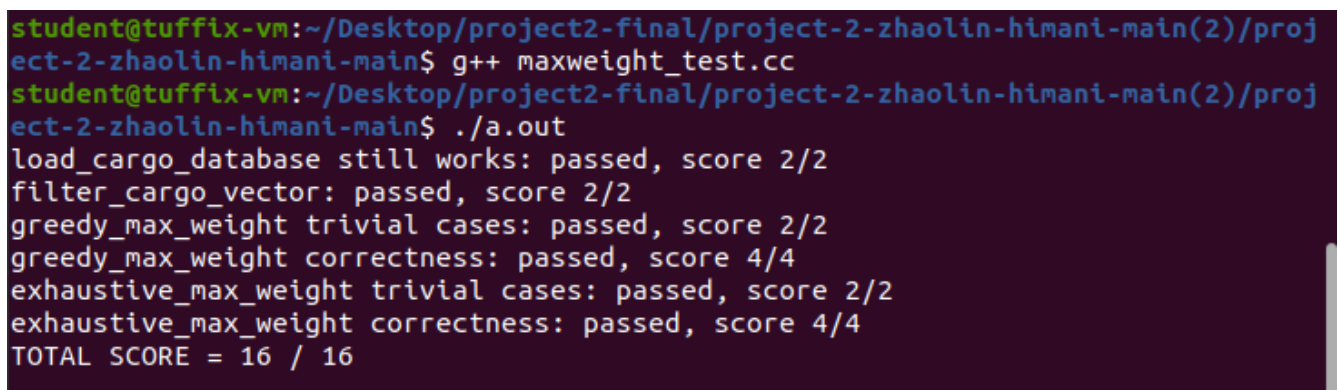
Name	Email ID
Himani Tawade	himani.tawade@csu.fullerton.edu
Zhaolin Zhu	zzl@csu.fullerton.edu

Read Me File:



```
README.md — ~/Desktop/project-2-zhaolin-himani-main — Atom
File Edit View Selection Find Packages Help
Project
project-2-zhaolin-himani-main
  .gitignore
  a.out
  cargo.csv
  LICENSE
  Makefile
  maxweight_scatterplot.cc
  maxweight_test.cc
  maxweight.hh
  maxweight.hh.gch
  README.md
  rubrictest.hh
  timer.hh
maxweight.hh
README.md
maxweight_test.cc
cargo.csv
1 # 335-project-2
2 Container ship weight-maximization
3
4 Group members:
5
6 Zhaolin Zhu zzl@csu.fullerton.edu
7
8 Himani Tawade himani.tawade@csu.fullerton.edu
9
```

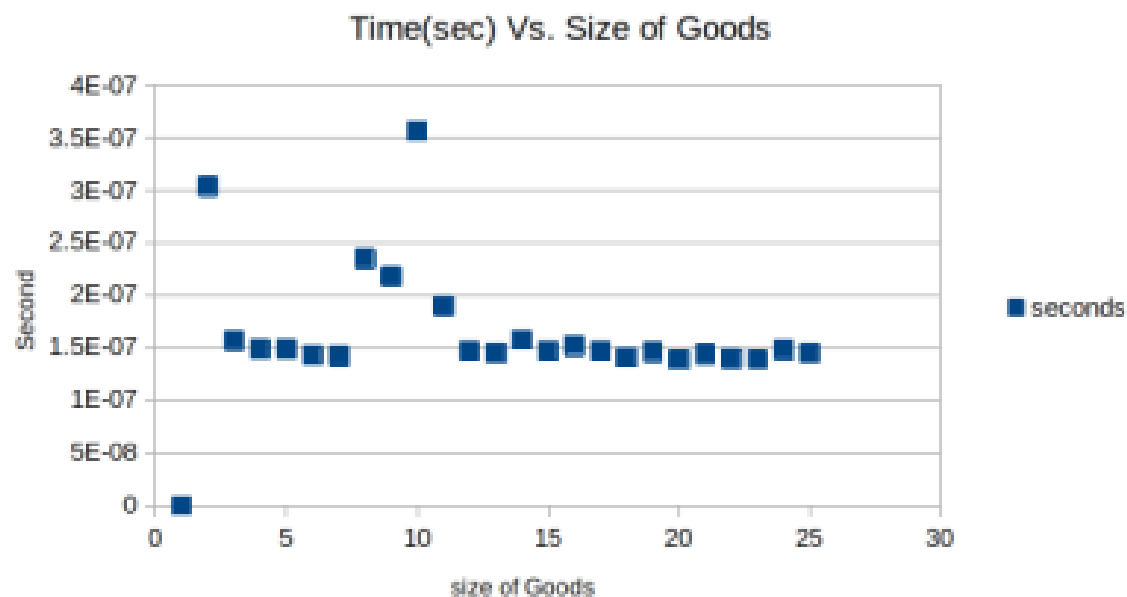
Code Compilation and Test Run Output:



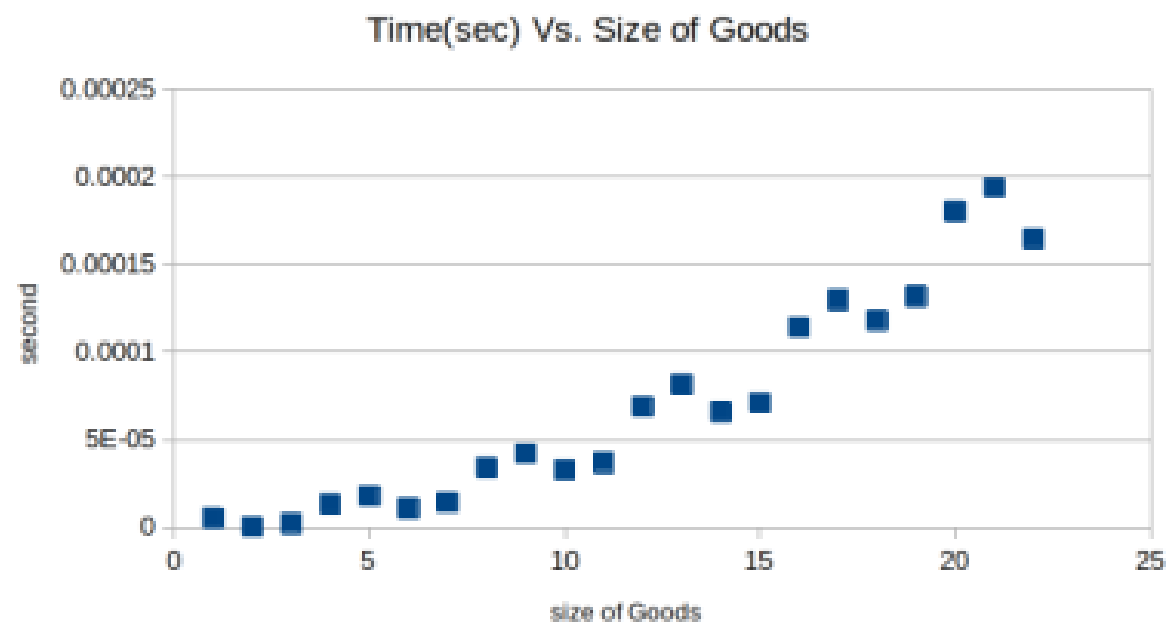
```
student@tuffix-vm:~/Desktop/project2-final/project-2-zhaolin-himani-main(2)/proj
ect-2-zhaolin-himani-main$ g++ maxweight_test.cc
student@tuffix-vm:~/Desktop/project2-final/project-2-zhaolin-himani-main(2)/proj
ect-2-zhaolin-himani-main$ ./a.out
load_cargo_database still works: passed, score 2/2
filter_cargo_vector: passed, score 2/2
greedy_max_weight trivial cases: passed, score 2/2
greedy_max_weight correctness: passed, score 4/4
exhaustive_max_weight trivial cases: passed, score 2/2
exhaustive_max_weight correctness: passed, score 4/4
TOTAL SCORE = 16 / 16
```

Scatter Plots:

Greedy Algorithm:



Exhaustive Optimization Algorithm:



Pseudo Code:

Greedy Algorithm:

```
greedy_max_time(V, goods):  
    todo = goods; 1  
    result = empty vector; 1  
    result_volume = 0; 1  
    while (todo is not empty): size(n)  
        {  
            index = 0; 1  
            counter = 0; 1  
            for each item in todo:{ size(n)  
                if (item(w)/ item(v) > todo(w(index))/todo(v(index))) 3  
                    index = counter; 1  
                endif  
                counter++; 1  
            endfor  
            if (result_volume + v(index)) <= V: 2  
                result.add(todo(index)) 1  
                result_volume += v(index) 1  
            endif  
            remove(todo(index)) 1  
        endwhile  
    return(result) 1  
}
```

S.C. Analysis

$= 1 + 1 + 1 + n * (1 + 1 + n * (3 + \max(1, 0) + 1) + 2 + \max(2, 0) + 1) + 1$
 $= 3 + n * (2 + n * (3 + 1 + 1) + 2 + 2 + 1) + 1$
 $= 3 + n * (2 + 5 * n + 5) + 1$
 $= 4 + 2 * n + 5 * n^2 + 5 * n$
 $= 5 * n^2 + 7 * n + 4$

Big O Proof

$5 * n^2 + 7 * n + 4$ belongs to $O(n^2)$

$c > 0$ and $n_0 > 0$

choose $c = 5 + 7 + 4 = 16$

$n_0 = 1$

$5 * n^2 + 7 * n + 4 \leq 16 * n^2$, for all $n \geq 1$

So, $5 * n^2 + 7 * n + 4$ belongs to $O(n^2)$

Exhaustive Optimization Algorithm:

```
initialize BEST;           1
initialize CANDIDATE;      1

for i from 0 to 2^size - 1    2 ^ size
    clear CANDIDATE;          1
    for j from 0 to size - 1    size
        if ((i >> j) & 1 == 1) 3
            CANDIDATE.add(todo[j]) 1
        endif
    endfor
    calculate CANDIDATE_weight size
    calculate CANDIDATE_volume size
    calculate BEST_weight size
    calculate BEST_volume size
    if (CANDIDATE_volume < total_volume) 1
        if (BEST == null || CANDIDATE_weight > BEST_weight) 3
            BEST = CANDIDATE 1
        endif
    endif
endfor

return BEST                1
```

S.C. Analysis

$= 1 + 1 + 2^n * (1 + n * (3 + \max(1, 0))) + n + n + n + n + 1 + \max(3 + \max(1, 0)) + 1$
 $= 2 + 2^n * (1 + 4n + 4n + 1 + 4) + 1$
 $= 3 + 2^n * (6 + 8n)$
 $= 8n * 2^n + 6 * 2^n + 3$

Big O Proof

$8n * 2^n + 6 * 2^n + 3$ belongs to $O(2^n * n)$
 $c > 0$ and $n_0 > 0$
choose $c = 8 + 6 + 3 = 17$
 $n_0 = 1$
 $8n * 2^n + 6 * 2^n + 3 \leq 17 * 2^n * n$, for all $n \geq 1$
So $8n * 2^n + 6 * 2^n + 3$ belongs to $O(2^n * n)$

Questions:

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 algorithm is much faster as compared to the exhaustive search algorithm which is proved by the time complexities of the two while greedy algorithm takes $O(n^2)$, exhaustive algorithm is an exponential i.e., $O(2^n * n)$. Yes, the huge difference in the time taken by the algorithms to solve the same problem is surprising and proves that every algorithm cannot be applied to all the problems, and it is critical to choose the right kind of algorithm to solve a problem.

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

Yes. From the Mathematical side, the Exhaustive Algorithm grows exponentially, if the size increases, the time for output will increase greatly. The Greedy Algorithm time complexity is quadratic, if the size increases, it will also increase greatly, but far less than the increase of the Exhaustive Algorithm. From the Empirical side, the Exhaustive Algorithm shows the great increase when the size increases. The Greedy Algorithm shows the increase but not that substantially.

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

Hypothesis 1: Exhaustive search algorithms are feasible to implement and produce correct outputs. It is correct because it tries all possible combinations to get the result, which will be a correct output. However, this is very time consuming.

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

Hypothesis 2: Algorithms with exponential running times are extremely slow, probably too slow to be of practical use.

It is correct because in the exhaustive search example, we need to run all possible combinations, which increase in an exponential way. When the size of input increases, the time will increase hugely.