Michael Gresham

Greshammichael@csu.fullerton.edu

Project 4: Dynamic vs. Exhaustive

Dynamic Algorithm

a) Mathematical Analysis

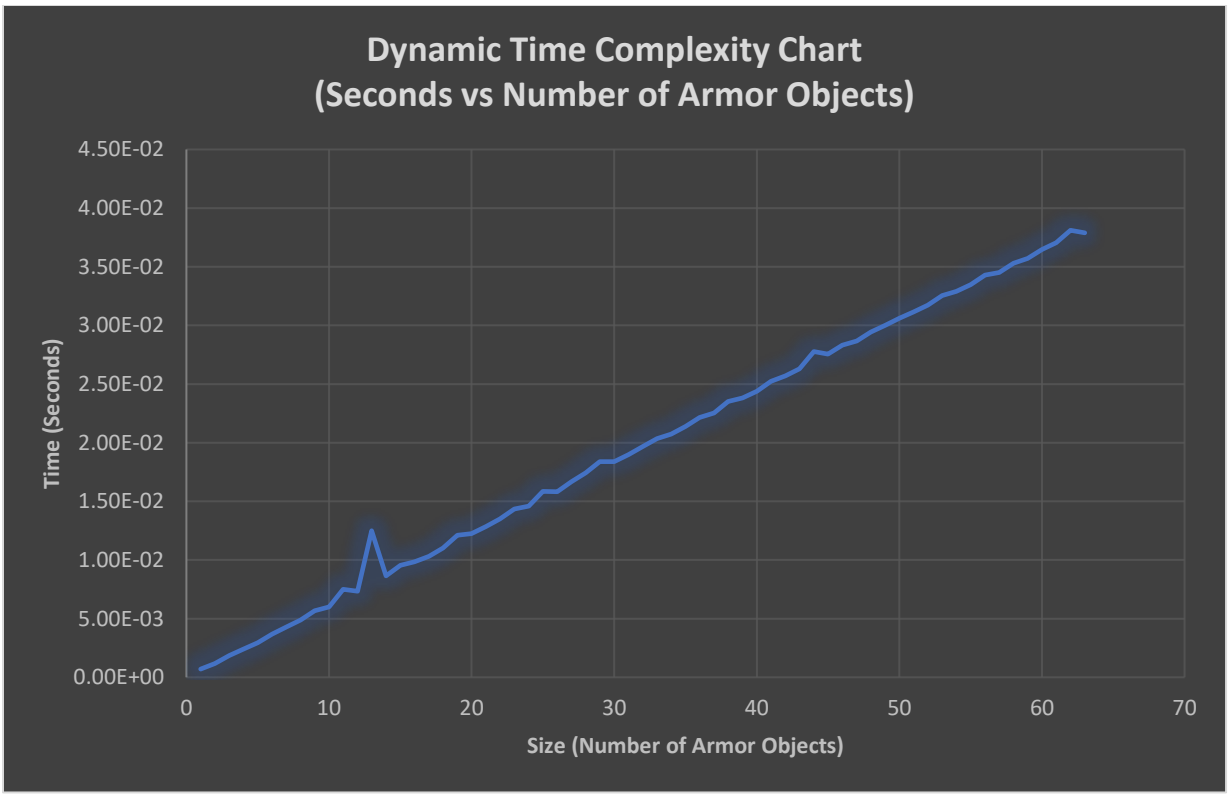
```
def dynamic_max_defense(const ArmorVector armors, int total_cost)
{

	int n = armors.size() // 2 t.u.
	int v = total_cost   // 2 t.u.
	// first set the 2d cache for finding the optimal solution
	vector< vector<double>> cache (n, v); //Set up blank 2d array of doubles filled
	//with zero's based on armor count and total_cost will take O(V x N) time
	//complexity
	double up;
	double left_up;


	for ( int i = 1; i <= n; i++) { // n - 1 + 1 = n

		for ( int j = 1; j <= v; j++) {  // v-1+1 = v

			up = cache.at(i - 1).at(j); // 2 t.u.

			if ((j - armors.at(i - 1)->cost()) < 0) {  // 3.tu.
				left_up = -1;  // 1 t.u.

			}

			else {

				left_up = cache.at(i-1).at(j - armors.at(i - 1)->cost()) +
				armors.at(i - 1)->defense();  // 6 t.u.
			}

		    if (left_up >= up) {   // 1.tu.
				cache.at(i).at(j) = left_up;   // 1 t.u.

			}

			else {
				cache.at(i).at(j) = up;    // 1 t.u.
			}




		}
```

```
        }

        // n* (v * ( 2 + 3 + max(1, 6) + 1 + max(1, 1)) = n * (v * (13) ) = 13nv

        std::unique_ptr<ArmorVector> replacement(new ArmorVector);
        // Now to find the items.

        int vertical = total_cost;   // 1 t.u.
        for (int i = n; i >= 0; i--) {  // n+1 t.u.

                if (cache.at(i).at(vertical) == 0) { // 1 t.u.
                        break;
                }

                if (cache.at(i - 1).at(vertical) != cache.at(i).at(vertical)){//2 t.u.


                        vertical = vertical - armors.at(i - 1)->cost(); // 3 t.u,
                        replacement->push_back(armors.at(i - 1));    // 1 t.u



                }


        }

        // (n+1) + 1 + 1 + max(0,0) + 2 + 4 = n+1+8 = n + 9


        return replacement;   // 1 t.u.
}
```

total step count : n + 9 + 13nv + 1 + 1 + 2+ 2 = 13nv + n + 15

O(N * V). The expected scatterplot will be linear since the test case will have constant total_cost but a changing number of objects. If total cost was not constant then it would be more of polyrhythmic shape.

b) Scatter Plot Graph:



Exhaustive Optimization Algorithm

a) Mathematical Analysis

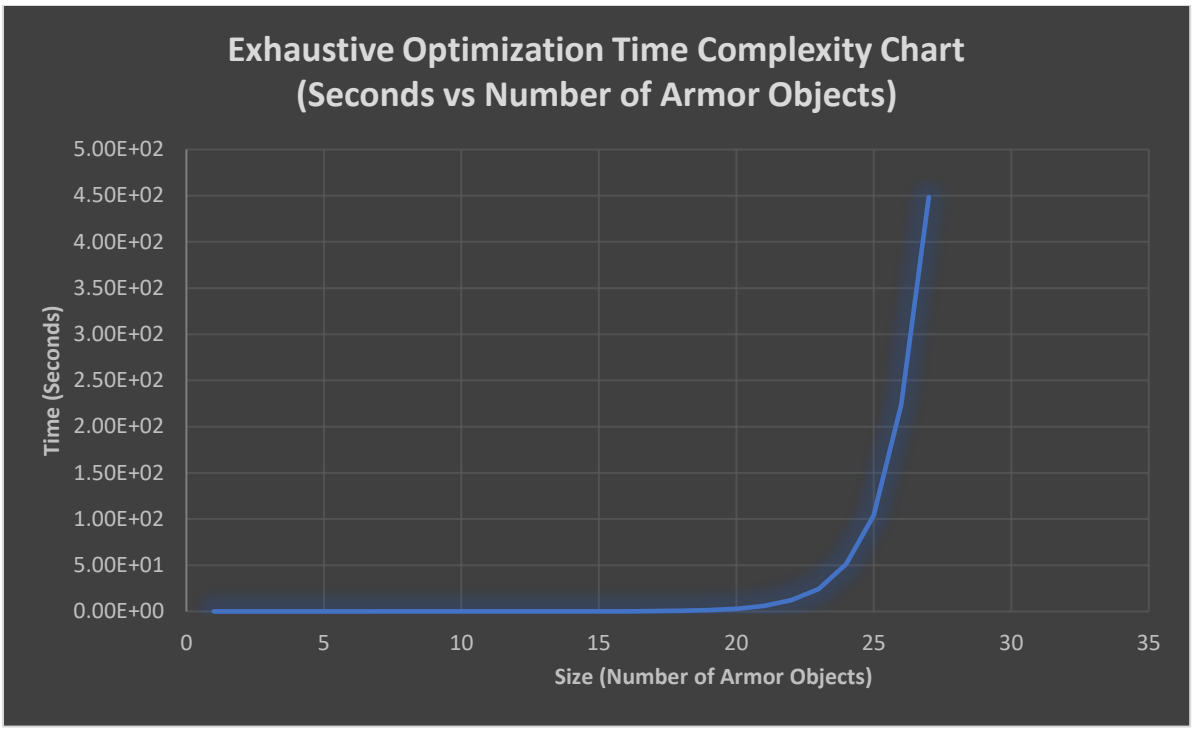
def Exhaustive(armor, total_cost):

```
const int n = armors.size();  // 1 t.u.

int counter = 0;   // 1 t.u.

ArmorVector best;

for int i = 0 to 2^n -1 do:        // 2^n times
       ArmorVector candidate;


       for int j = 0 to n-1 do:    // n times

              if (((i >> j)) & 1  == 1):   // 3 t.u.
                     candidate.push_back(armors.at(j));
```

```
            double candidate_cost = 0;          // 1 t.u.
            double candidate_defense = 0;       // 1 t.u
            sum_armor_vector(candidate, candidate_cost, candidate_defense);  // n times

            double best_cost = 0;               // 1 t.u.
            double best_defense = 0;            // 1 t.u.

            if(!best.empty()):   // 1 t.u
                  sum_armor_vector(best, best_cost, best_defense);// O(n) times

            if (candidate_cost <= total_cost) {       // 1 t.u.
                  if (candidate_defense > best_defense) {     // 1 t.u.
                              best = candidate;              // 1 t.u.


                  }

            }


      }

      std::unique_ptr<ArmorVector> best_subset(new ArmorVector);


      for (int i = 0; i < best.size(); i++) {    //  n - 1 - 0 + 1 = n

            best_subset->emplace_back(best.at(i));      // O(1)

      }



      // TODO: implement this function, then delete the return statement below
      return best_subset;  // 1 t.u.
```

total step count = 2+n+2^n(5n)+8(2^n) → O(n(2^n))

b) Scatterplot Graph



Questions

a) Comparing the dynamic and exhaustive algorithms I would say that the dynamic algorithm is much faster than the exhaustive one. An example being that for 27 items it took the exhaustive approach 448 seconds to return the right answer whereas it only took about 0.0167 seconds to give the correct answer. Which means that the exhaustive method runs about 26826 times slower than the dynamic method. Overall I was not surprised about this outcome considering that check all the subsets would at least take O(2^(n)) times whereas the dynamic method only took o(mn) time.

b) Yes, I do believe that the empirical results turns out to be similar to the mathematical analysis. The time complexity of the dynamic algorithm was O(mn) which is reflective to the scatterplot. Since total cost of armors remain constant throughout, and only the number of armors are changing then it makes sense that the graph is linear. If both the number of armors and total cost of armor was changing then it wouldn't be linear, but at the same time we wouldn't be able to compare the dynamic scatterplot to the exhaustive scatterplot. As the test variables would be different since in this case the scatterplot graph is only testing the effect of increasing the number of armors, has on the speed of the program. The exhaustive search's scatterplot clearly shows that the exhaustive algorithm's time complexity is O(2^n). For example, the data points on the exhaustive graph increases by about double between each point which is what you would expect from an exponential function with 2 as the base number (2^n). Not only that but the curve is very steep showing a clear rapid rise as the number of armor objects passes the 20 mark. As the times between armors sizes become greater the more items we have. Overall, both scatter points does an excellent job showing the time complexity of the functions and is definitely consistent to the mathematical analysis.

c) I think this experiment does show that exhaustive search algorithms are feasible to implement and produce correct outputs, but should only be used in cases where there are no better possible implementations. In this case however the dynamic method produces the correct answer in a fraction of the time that exhaustive can. Which means that while its feasible to implement an exhaustive search algorithm its practical use is small. As you would be limited to a very small number of armor objects to make the program run quickly. An example being the dynamic algorithm can calculate an answer for 1000 objects in .6 seconds whereas the exhaustive algorithm might take days or even weeks to calculate a correct result. Overall while it may be feasible to implement an exponential algorithm I wouldn't recommend using one unless you have no other algorithm to solve the problem.

d) The second hypothesis is definitely true. In the experiment when I was gathering empirical data it took a little over seven minutes to perform an exhaustive search on 27 different items. Which is impractical in the real-world as people need to get results quickly or risk losing money. If GPS systems in cars took exponential time to find the best possible path you would likely see that just finding the best route would be longer than taking a random path to your destination. Likewise, in the armor case if you had to search through hundreds of different armors you probably would be dead or have switched to another game by the time you figured out the best armor configuration for your character. So, the actual real-world applicability of function with exponential time complexity is very small if not non-existent. As you would only be able to effectively use the function when given a very small number of objects. Which isn't common in real world applications that has to deal with a tremendous number of objects like Facebook. Which if you consider every profile to be a user object has over 2 billion active users.