CS 456 : Advanced Algorithms
Programming Assignment #03
Total Points: 150

Assigned Date : Monday, April 18, 2016
Due Date : Monday, May 2, 2016 @ 02:59:59 p.m.

Overview

For your last programming assignment, you will implement three different algorithms to solve the **0-1 Knapsack Problem** – the *exhaustive (Sec 3.4 in Levitin)*, the *dynamic (Sec 8.2 in Levitin)*, and the *branch-and-bound (Sec 12.2 in Levitin)* – and empirically validate their asympototic runtime behavior using **computer generated results**. More specifically, you are expected to think about and address the following questions:

a. At what size n_0 does your implementation start to exhibit asymptotic complexity?
b. How do you **plan** to generate the appropriate input?
c. How does the measured run time correspond to the abstract complexity analysis using operation counting (as discussed in class)?
d. In the case of the three optimal solutions, at what sizes do the algorithms perform better than the others? What are the characteristics of the input that could influence the choice of algorithm to use?
e. How do you **plan** to show the inherent differences in the algorithms' complexity using visual representations?
f. How to create the algorithms so that they will be extensible and reusable for future projects.

Instructions

- This is an individual assignment. **Do your own work**.
- **Start early!!**
- **Take backups of your code often!!**.
- Make sure to test your program properly before your final submission.
- You may use any programming language of your choice. However, you **must** make sure that your code compiles and runs on a typical Linux machine. Absolutely **DO NOT** include executables with your submissions. A Makefile is **mandatory**. Please test your makefile on Linux machine prior to sending. Do not assume that your program can be designed in Visual Studio, Eclipse, or NetBeans on Windows and then transition to a Linux machine without problems.
- The report part of your solution must be produced using a word processor. Any figures, graphs, plots, etc., should also be produced using appropriate computer applications. Be professional with your reports; properly label and title your graphs; properly caption and cross-reference your figures; make sure to include all sections/subsection mentioned below.

- Your final report **must** be in **PDF** format. No exceptions.
- Follow a good coding standard. Use the Google C++ coding standard found here `http://goo.gl/1rC1o`, if you don't already follow one.
- For input, your program should be able to read in a file with the following format: The first line of your file signifies the knapsack capacity; The subsequent lines lists the **`weight<space>value`** of each item per line. When run, your program should prompt the user to type-in the input file name (and the path, if not in **`"./"`**). Here's an example input file:

```
10
5 $10
4 $40
6 $30
3 $50
```

- For output, produce a file named output.txt. The first line should have the maximum weight. The second line is a list of all items inputted in the format: weight value. All items should be on the same line. In addition, it should contain the times of all algorithms, the optimal weight and the optimal value found for the algorithms. (See example below)
- Total points: [**150 points**]

Deliverables

The due date of this assignment is **Monday, May 2, 2016 @ 02:59:59 p.m.** A dropbox will be opened for submission on Moodle before the due date. A complete solution comprises of:

- [**56 points**] A report that includes the followings:
  - Motivation and background of the experiment. This includes real world examples of this problem [**5 points**].
  - Pseudocode of your algorithm appropriately annotated with the theoretical runtime analysis. It is also necessary at this point to provide a detailed description of the various algorithms to illustrate their similarities and differences including the runtimes of the algorithms[**5 points**].
  - Testing Plan and Test Results. Thoroughly discuss how you intend to generate the input that will provide you with a full understanding of the characteristics of the algorithms [**10 points**].
  - A correctness proof of your programs [**5 points**].
  - Problems Encountered/Key insights [**5 points**].
  - Justification of your observations. You must be able to justify and/or argue the empirical asymptotic behavior you are observing. This section should focus on how your experiment's timing ran compared to the theoretical expectations [**13 points**].
  - Conclusion and performance comparisons of 3 Optimized Algorithms. This section should emphasize the differences and similarities between the 3 optimized algorithms based on results. Attention should be paid to how your testing plan failed or succeeded in enabling correct conclusions to be drawn from your experiment [**13 points**].
- [**94 points**] A compressed tarball of the directory containing your source code and Makefile. Do not include executables in this tarball; we will do a fresh compile of your code using your Makefile. To create a compressed tarball of the directory source, use the following command: `tar -zcvf name-111-pr1.tar.gz source/`. I will only be using a Linux machine so a zipped Windows file is not acceptable. Obviously, change the name to your last name and 111 to the last three digits of your SIUE ID.

Bonus Possibility [15 points]

P. 454 of the Levitin book discusses an approximation algorithm for the Knapsack Problem. Implement this algorithm in your code and run it just like you did the other three. Include in your report:

- What is the characteristic of your input required to generate results closest to optimal? Worst approximation results?
- Analysis of the algorithm including it's approximation ratio
- Graphs representing accuracy and time complexity
- Detailed discussion of why and when an approximation algorithm would be useful?

Sample output file

If you chose not to implement the bonus portion, it will not be necessary to include greedy approx, approximate weight, approximate value, and approximation accuracy in the output file.

output.txt
Maximum Weight: 10
2 2 4 6 6 5 8 1
Exhaustive: 1.4s
Dynamic: .053s
Branch and Bound: .87s
Greedy Approx: .005
Optimal Weight: 10
Optimal Value: 11
Approximate Weight: 6
Approximate Value: 8
Approximation Accuracy: .72