# Problem Set 7

**Both theory and programming questions** are due on **Tuesday, December 6** at **11:59PM**. Please download the .zip archive for this problem set. Refer to the `README.txt` file for instructions on preparing your solutions.

We will provide the solutions to the problem set 10 hours after the problem set is due. You will have to read the solutions, and write a brief **grading explanation** to help your grader understand your write-up. You will need to submit the grading guide by **Thursday, December 8, 11:59PM**. Your grade will be based on both your solutions and the grading explanation.

---

**Collaborators:** None.

---

**Problem 7-1.** [30 points] **Seam Carving**

In a recent paper, "Seam Carving for Content-Aware Image Resizing", Shai Avidan and Ariel Shamir describe a novel method of resizing images. You are welcome to read the paper, but we recommend starting with the YouTube video:

`http://www.youtube.com/watch?v=vIFCV2spKtg`

Both are linked from the Problem Sets page on the class website. After you've watched the video, the terminology in the rest of this problem will make sense.

If you were paying attention around time 1:50 of the video, then you can probably guess what you're going to have to do. You are given an image, and your task is to calculate the best vertical seam to remove. A *vertical seam* is a connected path of pixels, one pixel in each row. We call two pixels *connected* if they are vertically or diagonally adjacent. The *best* vertical seam is the one that minimizes the total "energy" of pixels in the seam.

The video didn't spend much time on dynamic programming, so here's the algorithm:

**Subproblems:** For each pixel $(i, j)$, what is the lower-energy seam that starts at the top row of the image, but ends at $(i, j)$?

**Relation:** Let `dp[i, j]` be the solution to subproblem $(i, j)$. Then

```
dp[i, j] = min(dp[i, j-1],dp[i-1, j-1],dp[i+1, j-1]) + energy(i, j)
```

**Analysis:** Solving each subproblem takes $O(1)$ time: there are three smaller subproblems to look up, and one call to `energy()`, which all take $O(1)$ time. There is one subproblem for each pixel, so the running time is $\Theta(A)$, where $A$ is the number of pixels, i.e., the area of the image.

Download `ps7_code.zip` and unpack it. To solve this problem set, you will need the Python Imaging Library (PIL), which you should have installed for Problem Set 4. If you wish to view your results, you will additionally need the Tkinter library.

In `resizeable_image.py`, write a function `best_seam(self)` that returns a list of coordinates corresponding to the cheapest vertical seam to remove, e.g., $[(5,0),(5,1),(4,2),(5,3),(6,4)]$. You should implement the dynamic program described above in a bottom-up manner.

The class `ResizeableImage` inherits from `ImageMatrix`. You should use the following components of `ImageMatrix` in your dynamic program:

- `self.energy(i,j)` returns the energy of a pixel. This takes $O(1)$ time, but the constant factor is sizeable. If you call it more than once, you might want to cache the results.
- `self.width` and `self.height` are the width and height of the image, respectively.

Test your code using `test_resizable_image.py`, and submit `ResizeableImage.py` to the class website. You can also view your code in action by running `gui.py`. Included with the problem set are two differently sized versions of the same sunset image. If you remove enough seams from the sunset image, it should center the sun.

Also, please try out your own pictures (most file formats should work), and send us any interesting before/after shots.

**Problem 7-2.** [70 points] **HG Fargo**

You have been given an internship at the extremely profitable and secretive bank HG Fargo. Your immediate supervisor tells you that higher-ups in the bank are very interested in learning from the past. In particular, they want to know how much money they *could* have made if they had invested optimally.

Your supervisor gives you the following data on the prices[1] of select stocks in 1991 and in 2011:

| **Company** | **Price in 1991** | **Price in 2011** |
|---|---|---|
| Dale, Inc. | $12 | $39 |
| JCN Corp. | $10 | $13 |
| Macroware, Inc. | $18 | $47 |
| Pear, Inc. | $15 | $45 |

As a first step, you decide to examine what the optimal decision is for a couple of small examples:

**(a)** [5 points] If you had $20 available to purchase stocks in 1991, how much of each stock should you have bought to maximize profits when you sell everything in 2011? Note that you do not need to invest all of your money — if it is more profitable to keep some as cash, you do not need to invest it.

**Answer:**

| **Company** | **Number of Shares** |
|---|---|
| Dale, Inc. | 0 |
| JCN Corp. | 0 |
| Macroware, Inc. | 1 |
| Pear, Inc. | 0 |

**(b)** [5 points] If you had $30 available to purchase stocks in 1991, how much of each stock should you have bought?

**Answer:**

| **Company** | **Number of Shares** |
|---|---|
| Dale, Inc. | 0 |
| JCN Corp. | 0 |
| Macroware, Inc. | 0 |
| Pear, Inc. | 2 |

**(c)** [5 points] If you had $120 available to purchase stocks in 1991, how much of each stock should you have bought?

**Answer:**

[1] Note that for the purposes of this problem, you should ignore some of the intricacies of the real stock market. The only income you can make is from purchasing stocks in 1991, then selling those same stocks at market value in 2011.

| Company | Number of Shares |
|---|---|
| Dale, Inc. | 10 |
| JCN Corp. | 0 |
| Macroware, Inc. | 0 |
| Pear, Inc. | 0 |

Your supervisor asks you to write an algorithm for computing the best way to purchase stocks, given the initial money $total$, the number $count$ of companies with stock available, an array $start$ containing the prices of each stock in 1991, and an array $end$ containing the prices of each stock in 2011. All prices are assumed to be positive integers.

There is a strong relationship between this problem and the knapsack problem. The knapsack problem takes four inputs: the number of different items $items$, the item sizes $size$ (all of which are integers), the item values $value$ (which may not be integers), and the size $capacity$ of the knapsack. The goal is to pick a subset of the items that fits inside the knapsack and maximizes the total value.

**(d)** [1 point] Which input to the knapsack problem corresponds to the input $total$ in the stock purchasing problem?

1. $items$  2. $size$  3. $value$  4. $capacity$

**Answer:** 4

**(e)** [1 point] Which input to the knapsack problem corresponds to the input $count$ in the stock purchasing problem?

1. $items$  2. $size$  3. $value$  4. $capacity$

**Answer:** 1

**(f)** [1 point] Which input to the knapsack problem corresponds to the input $start$ in the stock purchasing problem?

1. $items$  2. $size$  3. $value$  4. $capacity$

**Answer:** 2

**(g)** [1 point] Which input to the knapsack problem corresponds to the input $end$ in the stock purchasing problem?

1. $items$  2. $size$  3. $value$  4. $capacity$

**Answer:** 3

**(h)** [6 points] Unfortunately, the algorithm for the knapsack problem cannot be directly applied to the stock purchasing problem. For each of the following potential reasons, state whether it's a valid reason not to use the knapsack algorithm. (In other words, if the difference mentioned were the only difference between the problems, would you still be able to use the knapsack algorithm to solve the stock purchasing problem?)

1. In the stock purchasing problem, there is a time delay between the selection and the reward.
2. All of the numbers in the stock purchasing problem are integers. The $value$ array in the knapsack problem is not.
3. In the stock purchasing problem, the money left over from your purchases is kept as cash, which contributes to your ultimate profit. The knapsack problem has no equivalent concept.
4. In the knapsack problem, there are some variables representing sizes of objects. There are no such variables in the stock purchasing problem.
5. In the stock purchasing problem, you can buy more than one share in each stock.
6. In the stock purchasing problem, you sell all the items at the end. In the knapsack problem, you don't do anything with the items.

**Answer:** 3 5

Despite these differences, you decide that the knapsack algorithm is a good starting point for the problem you are trying to solve. So you dig up some pseudocode for the knapsack problem, relabel the variables to suit the stock purchasing problem, and then start modifying things. After a long night of work, you end up with a couple of feasible solutions. Unfortunately, there is a bit of a hard-drive error the next morning, and the files are all mixed up. You have recovered six different functions, from various states in your development process. The first function is the following:

```
STOCK(total, count, start, end)
    purchase = STOCK-TABLE(total, count, start, end)
    return STOCK-RESULT(total, count, start, end, purchase)
```

This is the function that you ran to get your results. The STOCK-TABLE function generates the table of subproblem solutions. The STOCK-RESULT function uses that to figure out which stocks to purchase, and in what quantities. Unfortunately, you have two copies of the STOCK-TABLE function and three copies of the STOCK-RESULT function. You know that there's a way to take one of each function to get the pseudocode for the original knapsack problem (with the names changed). You also know that there's a way to take one of each function to get the pseudocode for the stock purchases problem. You just don't know which functions do what.

Analyze each of the other five procedures, and select the correct running time. Recall that $total$ and $count$ are positive integers, as are each of the values $start[stock]$ and $end[stock]$. To make the code simpler, the arrays $start$, $end$, and $result$ are assumed to be indexed starting at $1$, while the tables $profit$ and $purchase$ are assumed to be indexed starting at $(0, 0)$. You may assume that entries in a table can be accessed and modified in $\Theta(1)$ time.

**(i)** [1 point] What is the worst-case asymptotic running time of STOCK-TABLE-A (from Figure 1) in terms of $count$ and $total$?

```
STOCK-TABLE-A(total, count, start, end)
create a table profit
create a table purchase
for cash = 0 to total
    profit[cash, 0] = cash
    purchase[cash, 0] = FALSE
    for stock = 1 to count
        profit[cash, stock] = profit[cash, stock − 1]
        purchase[cash, stock] = FALSE
        if start[stock] ≤ cash
            leftover = cash − start[stock]
            current = end[stock] + profit[leftover, stock]
            if profit[cash, stock] < current
                profit[cash, stock] = current
                purchase[cash, stock] = TRUE
return purchase
```

**Figure 1**: The pseudocode for STOCK-TABLE-A.

```
STOCK-TABLE-B(total, count, start, end)
create a table profit
create a table purchase
for cash = 0 to total
    profit[cash, 0] = 0
    purchase[cash, 0] = FALSE
    for stock = 1 to count
        profit[cash, stock] = profit[cash, stock − 1]
        purchase[cash, stock] = FALSE
        if start[stock] ≤ cash
            leftover = cash − start[stock]
            current = end[stock] + profit[leftover, stock − 1]
            if profit[cash, stock] < current
                profit[cash, stock] = current
                purchase[cash, stock] = TRUE
return purchase
```

**Figure 2**: The pseudocode for STOCK-TABLE-B.

1. $\Theta(count)$
2. $\Theta(count^2)$
3. $\Theta(count^3)$
4. $\Theta(total)$
5. $\Theta(total^2)$
6. $\Theta(total^3)$
7. $\Theta(count + total)$
8. $\Theta(count^2 + total)$
9. $\Theta(count + total^2)$
10. $\Theta(count \cdot total)$
11. $\Theta(count^2 \cdot total)$
12. $\Theta(count \cdot total^2)$

**Answer:** 10

**(j)** [1 point] What is the worst-case asymptotic running time of STOCK-TABLE-B (from Figure 2) in terms of $count$ and $total$?

1. $\Theta(count)$
2. $\Theta(count^2)$
3. $\Theta(count^3)$
4. $\Theta(total)$
5. $\Theta(total^2)$
6. $\Theta(total^3)$
7. $\Theta(count + total)$
8. $\Theta(count^2 + total)$
9. $\Theta(count + total^2)$
10. $\Theta(count \cdot total)$
11. $\Theta(count^2 \cdot total)$
12. $\Theta(count \cdot total^2)$

**Answer:** 10

**(k)** [1 point] What is the worst-case asymptotic running time of STOCK-RESULT-A (from Figure 3) in terms of $count$ and $total$?

1. $\Theta(count)$
2. $\Theta(count^2)$
3. $\Theta(count^3)$
4. $\Theta(total)$
5. $\Theta(total^2)$
6. $\Theta(total^3)$
7. $\Theta(count + total)$
8. $\Theta(count^2 + total)$
9. $\Theta(count + total^2)$
10. $\Theta(count \cdot total)$
11. $\Theta(count^2 \cdot total)$
12. $\Theta(count \cdot total^2)$

**Answer:** 1

**(l)** [1 point] What is the worst-case asymptotic running time of STOCK-RESULT-B (from Figure 4) in terms of $count$ and $total$?

1. $\Theta(count)$
2. $\Theta(count^2)$
3. $\Theta(count^3)$
4. $\Theta(total)$
5. $\Theta(total^2)$
6. $\Theta(total^3)$
7. $\Theta(count + total)$
8. $\Theta(count^2 + total)$
9. $\Theta(count + total^2)$
10. $\Theta(count \cdot total)$
11. $\Theta(count^2 \cdot total)$
12. $\Theta(count \cdot total^2)$

```
STOCK-RESULT-A(total, count, start, end, purchase)
create a table result
for stock = 1 to count
    result[stock] = 0
cash = total
stock = count
while stock > 0
    quantity = purchase[cash, stock]
    result[stock] = quantity
    cash = cash − quantity · start[stock]
    stock = stock − 1
return result
```

**Figure 3**: The pseudocode for STOCK-RESULT-A.

```
STOCK-RESULT-B(total, count, start, end, purchase)
create a table result
for stock = 1 to count
    result[stock] = FALSE
cash = total
stock = count
while stock > 0
    if purchase[cash, stock]
        result[stock] = TRUE
        cash = cash − start[stock]
    stock = stock − 1
return result
```

**Figure 4**: The pseudocode for STOCK-RESULT-B.

```
STOCK-RESULT-C(total, count, start, end, purchase)
create a table result
for stock = 1 to count
    result[stock] = 0

cash = total
stock = count
while stock > 0
    if purchase[cash, stock]
        result[stock] = result[stock] + 1
        cash = cash − start[stock]
    else
        stock = stock − 1

return result
```

**Figure 5**: The pseudocode for STOCK-RESULT-C.

**Answer:** 1

**(m)** [1 point] What is the worst-case asymptotic running time of STOCK-RESULT-C (from Figure 5) in terms of *count* and *total*?

1. $\Theta(\mathit{count})$
2. $\Theta(\mathit{count}^2)$
3. $\Theta(\mathit{count}^3)$
4. $\Theta(\mathit{total})$
5. $\Theta(\mathit{total}^2)$
6. $\Theta(\mathit{total}^3)$
7. $\Theta(\mathit{count} + \mathit{total})$
8. $\Theta(\mathit{count}^2 + \mathit{total})$
9. $\Theta(\mathit{count} + \mathit{total}^2)$
10. $\Theta(\mathit{count} \cdot \mathit{total})$
11. $\Theta(\mathit{count}^2 \cdot \mathit{total})$
12. $\Theta(\mathit{count} \cdot \mathit{total}^2)$

**Answer:** 7

**(n)** [2 points] The recurrence relation computed by the STOCK-TABLE-A function is:

1. $\mathit{profit}[c, s] = \max\{\mathit{profit}[c, s-1], \mathit{profit}[c - \mathit{start}[s], s-1]\}$
2. $\mathit{profit}[c, s] = \max\{\mathit{profit}[c, s-1], \mathit{profit}[c - \mathit{start}[s], s-1] + \mathit{end}[s]\}$
3. $\mathit{profit}[c, s] = \max_q\{\mathit{profit}[c - q \cdot \mathit{start}[s], s-1] + q \cdot \mathit{end}[s]\}$
4. $\mathit{profit}[c, s] = \max\{\mathit{profit}[c, s-1], \mathit{profit}[c - \mathit{start}[s], s]\}$
5. $\mathit{profit}[c, s] = \max\{\mathit{profit}[c, s-1], \mathit{profit}[c - \mathit{start}[s], s] + \mathit{end}[s]\}$
6. $\mathit{profit}[c, s] = \max_q\{\mathit{profit}[c - q \cdot \mathit{start}[s], s] + q \cdot \mathit{end}[s]\}$

**Answer:** 5

**(o)** [2 points] The recurrence relation computed by the STOCK-TABLE-B function is:

1. $\mathit{profit}[c, s] = \max\{\mathit{profit}[c, s-1], \mathit{profit}[c - \mathit{start}[s], s-1]\}$
2. $\mathit{profit}[c, s] = \max\{\mathit{profit}[c, s-1], \mathit{profit}[c - \mathit{start}[s], s-1] + \mathit{end}[s]\}$
3. $\mathit{profit}[c, s] = \max_q\{\mathit{profit}[c - q \cdot \mathit{start}[s], s-1] + q \cdot \mathit{end}[s]\}$
4. $\mathit{profit}[c, s] = \max\{\mathit{profit}[c, s-1], \mathit{profit}[c - \mathit{start}[s], s]\}$
5. $\mathit{profit}[c, s] = \max\{\mathit{profit}[c, s-1], \mathit{profit}[c - \mathit{start}[s], s] + \mathit{end}[s]\}$
6. $\mathit{profit}[c, s] = \max_q\{\mathit{profit}[c - q \cdot \mathit{start}[s], s] + q \cdot \mathit{end}[s]\}$

**Answer:** 2

With this information, you should be able to figure out whether STOCK-TABLE-A or STOCK-TABLE-B is useful for the knapsack problem, and similarly for the stock purchasing problem. From there, you can figure out which of STOCK-RESULT-A, STOCK-RESULT-B, and STOCK-RESULT-C is best for piecing together the optimal distribution of stocks and/or items.

**(p)** [3 points] Which two methods, when combined, let you compute the answer to the knapsack problem?

1. STOCK-TABLE-A and STOCK-RESULT-A
2. STOCK-TABLE-A and STOCK-RESULT-B
3. STOCK-TABLE-A and STOCK-RESULT-C
4. STOCK-TABLE-B and STOCK-RESULT-A
5. STOCK-TABLE-B and STOCK-RESULT-B
6. STOCK-TABLE-B and STOCK-RESULT-C

**Answer:** 5

**(q)** [3 points] Which two methods, when combined, let you compute the answer to the stock purchases problem?

1. STOCK-TABLE-A and STOCK-RESULT-A
2. STOCK-TABLE-A and STOCK-RESULT-B
3. STOCK-TABLE-A and STOCK-RESULT-C
4. STOCK-TABLE-B and STOCK-RESULT-A
5. STOCK-TABLE-B and STOCK-RESULT-B
6. STOCK-TABLE-B and STOCK-RESULT-C

**Answer:** 3

With all that sorted out, you submit the code to your supervisor and pat yourself on the back for a job well done. Unfortunately, your supervisor comes back a few days later with a complaint from the higher-ups. They've been playing with your program, and were very upset to discover that when they ask what to do with $1,000,000,000 in the year 1991, it tells them to buy tens of millions of shares in Dale, Inc. According to them, there weren't that many shares of Dale available to purchase. They want a new feature: the ability to pass in limits on the number of stocks purchaseable.

You choose to begin, as always, with a small example:

| **Company** | **Price in 1991** | **Price in 2011** | **Limit** |
|---|---|---|---|
| Dale, Inc. | $12 | $39 | 3 |
| JCN Corp. | $10 | $13 | $\infty$ |
| Macroware, Inc. | $18 | $47 | 2 |
| Pear, Inc. | $15 | $45 | 1 |

**(r)** [5 points] If you had $30 available to purchase stocks in 1991, how much of each stock should you have bought, given the limits imposed above?

**Answer:**

| **Company** | **Number of Shares** |
|---|---|
| Dale, Inc. | 1 |
| JCN Corp. | 0 |
| Macroware, Inc. | 0 |
| Pear, Inc. | 1 |

**(s)** [5 points] If you had $120 available to purchase stocks in 1991, how much of each stock should you have bought, given the limits imposed above?

**Answer:**

| **Company** | **Number of Shares** |
|---|---|
| Dale, Inc. | 3 |
| JCN Corp. | 3 |
| Macroware, Inc. | 2 |
| Pear, Inc. | 1 |

**(t)** [20 points] Give pseudocode for an algorithm STOCKLIMITED that computes the maximum profit achievable given a starting amount $total$, a number $count$ of companies with stock available, an array of initial prices $start$, an array of final prices $end$, and an array of quantities $limit$. The value stored at $limit[stock]$ will be equal to $\infty$ in cases where there is no known limit on the number of stocks. The algorithm need only output the resulting quantity of money, not the purchases necessary to get that quantity.

Remember to analyze the runtime of your pseudocode, and provide a brief justification for its correctness. It is sufficient to give the recurrence relation that your algorithm implements, and talk about why the recurrence relation solves the problem at hand.

**Answer:** I will implement the following recurrence, where $c$ is the current cash, $s$ is the current stock, and $x$ is the current number of stocks bought.

$$profit(c, s, x) = \max \begin{cases} \max_{i \in \{0,\ldots,limit[s-1]\}} \{profit(c, s-1, i)\} \\ profit(c - start[s], s, x-1) + end[s] \end{cases} \tag{1}$$

Here, $profit(c, s, x)$ corresponds to the highest possible profit one can achieve given a certain amount of cash, a decision to begin purchasing stock $s$, and a maximum amount $x$ of stocks of $s$ one can buy.

```
STOCKLIMITED(total, count, start, end, limit)
create a table profit
create a table bestprofit
for stock = 1 to count
    if limit[stock] = ∞
        limit[stock] = total/start[stock]
for cash = 0 to total
    for stock = 1 to count
        profit[cash, stock, 0] = cash
        leftover = cash − start[stock]
        currentMax = profit[cash, stock − 1, 0]
        for x = 1 to limit[stock]
            if profit[leftover, stock, x − 1] > bestprofit[cash, stock − 1]
                if start[stock] ≤ cash
                    profit[cash, stock, i] = profit[leftover, stock, i − 1] + end[stock]
                    if profit[cash, stock, i] > currentMax
                        currentMax = profit[cash, stock, i]
            else
                profit[cash, stock, i] = bestprofit[cash, stock − 1]
                if bestprofit[cash, stock − 1] > currentMax
                    currentMax = bestprofit[cash, stock − 1]
        bestprofit[cash, stock] = currentMax
return bestprofit[total, count]
```

The psuedocode implements the recurrence relation in equation (1) by creating a new table $bestprofit$. Here, $bestprofit[c, s] = \max_{i \in \{0,\ldots,limit[s]\}} \{profit(c, s, i)\}$. Thus, the recurrence relation becomes $profit(c, s, x) = \max\{bestprofit[c, s-1], profit(c - start[s], s, x-1) + end[s]\}$. Since an elementary lookup can be performed in amortized $\Theta(1)$ time for both $bestprofit[c, s-1]$ and $profit(c - start[s], s, x-1) + end[s]$, we see that each subproblem takes $\Theta(1)$ time. We can find the number of subproblems:

$$\#subproblems = cash \left( \sum_{s=1}^{count} limit[s] \right) \tag{2}$$

In the worst case, each $limit[s] = \infty$. However, since we can only buy stock $s$ a maximum of $n = total/start[s]$ times, my psuedocode sets $limit[s] = total/start[s]$ when $limit[s] = \infty$ occurs. Thus, the worst case for our recurrence is when $start[s] = 1$ for all $s$ and $limit[s] = total/start[s] = total$. Thus, the worst case running time is $\Theta(cash * count * total)$ based on equation (2).

The recurrence is correct because each $profit(c, s, x)$ represents the possible choices one can take. The first choice is that one decides to purchase stock $s$. In this case, the maximum profit is the maximum profit of what you can do with the rest of the cash after buying the stock plus the stock price. Note that since you have just bought stock $s$, you can buy one less stock than before, so you must decrement the maximum number of stocks one can still buy. This represents the case of $profit(c - start[s], s, x - 1) + end[s]$.

The other case is where you decide not to buy a stock. In this case, you will go on to the next stock, but you must take the best possible choice with that next stock. Thus, you must check all possible amounts of the next stock you could possibly buy. Therefore, if $s$ is the next stock, you have to take the maximum $profit(c, s, i)$ for all $i$ below $limit[s]$. This represents the case $\max_{i \in \{0,\ldots,limit[s-1]\}}\{profit(c, s-1, i)\}$ in the recurrence.

Since these are the only possible decisions on can make, we have finished our recurrence. To verify the base case, we should verify $profit(c, s, 0)$. If we have not bought any stocks of $s$, then we still have a profit of the amount of cash we have, $profit(c, s, 0) = c$.