McGill University COMP360 Winter 2011 **Instructor: Phuong Nguyen**

Assignment 9

Due March 23 before the lecture

The work you submit must be your own. You may discuss problems with each others; however, you should prepare written solutions alone. Copying assignments is a serious academic offense, and will be dealt with accordingly.

Question 1 (10pt) Consider the following variant of the Knapsack problem. The input consists of

- a set of items with associated weights and values, just as before:

$$S = \{(w_1, v_1), (w_2, v_2), \ldots, (w_n, v_n)\},$$

- a target value V,
- an upper bound W,
- and a "relax" factor ϵ.

Furthermore, the set S is guaranteed to contain a subset of items whose total weight is $\leq W$ and whose total value is *exactly* V. The problem is to compute a subset of S whose total value is *at least* V, and whose total weight is $\leq (1 + \epsilon)W$ (so it can be a bit more than W).

Give a dynamic programming algorithm for solving this problem. Your algorithm must run in time polynomial in n and $\frac{1}{\epsilon}$. Prove the correctness of your algorithm and analyze its running time.

Question 2 (10pt) Your friends are looking at n consecutive days of a given stock, at some point in the past. The days are numbered $1, 2, \ldots, n$. For each day i they have a price $p(i)$ per share for the stock on that day.

For a certain (possibly large) integer k your friends want to know what is the best return of a so-called *k-shot strategy*. Here a k-shot strategy is a collection of m pairs of days

$$(b_1, s_1), (b_2, s_2), \ldots, (b_m, s_m)$$

for some $m \leq k$ and $b_1 < s_1 < b_2 < s_2 < \ldots < b_m < s_m$. This can be viewed as a set of at most k non-overlapping intervals, during each of which your friends buy 1,000 shares of the stock (on day b_t) and then sell it (on day s_t). The return of such a strategy is simply the profit of the transaction, i.e.,

$$1,000 \sum_{t=1}^{m} (p(s_t) - p(b_t))$$

You are asked to design an efficient algorithm to determine the best k-shot strategy.

Formally, the input to your algorithm consists of

- positive integers $p(1), p(2), \ldots, p(n)$,
- a positive integer $k \leq n/2$

The output is a sequence of m pairs

$$(b_1, s_1), (b_2, s_2), \ldots, (b_m, s_m)$$

as above, for some $m \leq k$, with maximum possible return.

Your algorithm must run in time polynomial in n, k. Analyze its running time.