Here we introduce on-line algorithms and competitive analysis using a few simple examples and then look at a more complex cache management problem.

**Definitions**

An on-line algorithm $A$ is presented with a request sequence $\sigma = \sigma_1, \sigma_2, \ldots$. For example we could consider an on-line version of the 0-1 knapsack problem. Here $\sigma_i$ would correspond to telling $A$ the value and weight of item $i$. (We’ll assume that $A$ already knows the knapsack capacity $W$). In a cache management problem, $\sigma_i$ will be the $i$th page requested by the user.

The on-line algorithm $A$ must make some decision in regards to $\sigma_i$ without knowledge of $\sigma_j$ for all $j > i$. This is why it is called “on-line”. In the example of an on-line 0-1 knapsack algorithm, the algorithm would need to decide if item $i$ should be taken at that point and cannot change its mind after seeing the later items. In the cache maintenance problem, if there is a page fault (i.e. the requested page is not in cache) a decision must be made as to which page to remove from cache to make room for the requested page. Again, this must be done without any knowledge of the future requests that are to be made. In contrast, an off-line algorithm will receive all of $\sigma$ before needing to make any decision about $\sigma_1$.

- Let $C_A(\sigma)$ be the cost incurred by $A$ on input sequence $\sigma$.
- Let $C_{OPT}(\sigma)$ be the cost incurred by the optimal off-line algorithm on input $\sigma$. Note, that the optimal off-line algorithm not only sees all of $\sigma$ before making any decisions, but also has unbounded computational resources.

We say that $A$ is $c$-competitive if there exists a constant $a$ such that

$$\forall \sigma \quad C_A(\sigma) \leq c \cdot C_{OPT}(\sigma) + a$$

As with approximation algorithms, $c$ can be a function of the input size. We say that $A$ is strongly $c$-competitive if

$$\forall \sigma \quad C_A(\sigma) \leq c \cdot C_{OPT}(\sigma)$$

We will also consider randomized on-line algorithms. In this case we consider what is known as the oblivious adversary. The oblivious adversary knows the on-line algorithm $A$ but must generate $\sigma$ before $A$ begins (and so the outcome of the random decisions made are not known when $\sigma$ is made). For a randomized on-line algorithm $A$ we say that $A$ is $c$-competitive if for all input sequences $\sigma$ generated by an oblivious adversary

$$E[C_A(\sigma)] \leq c \cdot C_{OPT}(\sigma) + a$$

where the expectation in the expected performance of $C_A$ on $\sigma$ ($E[C_A(\sigma)]$) is taken over the random choices made by $A$. Observe, that it is still a worst-case bound over all possible inputs sequences $\sigma$. 

1
Rent-to-buy Problem

Suppose that you are going skiing for a season yet you do not know how many good skiing days there will be. Suppose it costs $18 to rent skis for a day and it cost $b8 to buy skis where $b$ is an arbitrary real that is at least 1. (So buying skis is $b$ times more expensive than renting. One could have the rental cost be $r$ and the cost to buy be $br$ and nothing would change. Thus for simplicity we let the cost to buy be 1.) The $i$th element of the input sequence $\sigma_i$ indicates whether or not the ski slopes will be open on the $i$th day.

Let $k$ be the number of days when the ski slope is open during the season. The off-line algorithm has knowledge of $k$, but the on-line algorithm does not know $k$ (until the last day of the season has passed).

We now give a strongly 2-competitive algorithm. Let $t$ be the total rental costs that has been paid. So initially $t = 0$. Until the point at which the skier chooses to buy the skis, on each good skiing day a decision must be made as to whether to rent skis or buy skis. If $t + 1 \geq b$ (i.e. renting one more day would cause the total rental cost to be at least the cost of buying skis) then the skier will buy skis on that day and otherwise (when $t + 1 < b$) the skier will rent (and increment $t$ since another dollar has been spent for rental).

We now prove this is strongly 2-competitive using a proof by cases:

**Case 1:** $k > b$. In this case the optimal solution is to buy on day one for a cost of $b$. The on-line algorithm will have cost at most 2$b$ since $b$ is spent for buying the skis and by construction the total rental cost is at most $b$ (since the algorithm would buy if one more day of rental would cause the rental cost to exceed $b$). Thus $C_A \leq 2C_{OPT}$ as desired.

**Case 2:** $k \leq b$. In this case the optimal solution is to rent for $k$ days for a total cost of $k$.

The on-line algorithm also has cost $k$ and thus in this case $C_A \leq 2C_{OPT}$ as desired.

(In fact, in this case $C_A = C_{OPT}$.)

All $\sigma$ fall into case 1 or case 2 and thus we have shown that for all $\sigma$, $C_A(\sigma) \leq 2C_{OPT}(\sigma)$ proving that $A$ is strongly 2-competitive.

Hole in the Fence Problem

You have an fence that is infinite in both directions. You begin at position 0. Each step will move you exactly one unit (in whichever direction you choose to go). There is a hole a position $h$ (where $h$ is an arbitrary integer). The on-line algorithm does not know the location of the hole and can only determine it by walking to the exact location of the hole. The off-line algorithm is told where the hole is before taking any steps. The goal is to minimize the number of steps required to reach the hole. For this problem the optimal solution will clearly take $|h|$ steps. The goal is to minimize the number of steps required by the on-line algorithm.

We now give a strongly 9-competitive algorithm $A$. Using an adversary lower bound technique, this algorithm can be proven to be optimal for any deterministic algorithm. On-line algorithm $A$ will first go to location 1. If the hole isn’t there it returns to the start and then goes to location -2. If the hole isn’t found on that path, $A$ returns to the start and
then goes to location 4, and so on at each step going twice as far as it just did but in the opposite direction. This is continued until the hole is reached.

For this problem $\sigma$ is just defined by the location of the hole $h$. Let $|h| = 2^x + y$ where $1 \leq y \leq 2^x - 1$. Observe that any integer $h$ can be obtained in this manner by an appropriate choice of $x$ and $y$. Observe that

$$C_A(\sigma) \leq 2 \cdot 1 + 2 \cdot 2 + 2 \cdot 4 + \cdots + 2 \cdot 2^x + 2 \cdot 2^{x+1} + |h|$$

where the multiplicative factor of 2 corresponds to walking to the location and then back to the start. In the worst case, when the algorithm reaches the location $2^x$ steps from the start (in whatever direction that happens to be) and the hole was on that side but still further away from the start. In this case, the on-line algorithm must walk to the location $2^{x+1}$ steps away from the start (and then back) before finally taking the final $|h|$ steps to the hole.

Just rewriting the above and substituting in the equality that $2^x = |h| - y$ gives

$$C_A(\sigma) \leq 2 \left( \sum_{i=0}^{x+1} 2^i \right) + |h|$$

$$= 2(2^{x+2} - 1) + |h| = 8 \cdot 2^x - 2 + |h|$$

$$= 8(|h| - y) - 2 + |h| = 9|h| - 8y - 2 \leq 9|h| - 10$$

for all $y$ since $y \geq 1$. From this one can see that the worst case for the algorithm is when the hole is one position past the position where the on-line algorithm turns around when first at position $2^x$ from the start.

Thus for all $\sigma$ (i.e. hole locations), $C_A(\sigma) \leq 9 \cdot C_{\text{OPT}}$ and thus this algorithm is strongly 9-competitive.