CS515 - Algorithms & Data Structures

Practice Assignment 3

Vy Bui - 934370552

Instructor: Professor Glencora Borradaile

The School of Electrical Engineering and Computer Science
Oregon State University

Problem 1
Job Scheduling

(a) Let $T = \{t_1, t_2, ..., t_n\}$ be the time the jobs $j_1, j_2, ..., j_n$ take.

Algorithm 1 $A(T)$

```
sortedT ← sort(T)
lastJobCompleteTime ← 0
totalTime ← 0
for t in T do
    totalTime ← totalTime + lastJobCompleteTime + t
    lastJobCompleteTime ← lastJobCompleteTime + t
end for
    return sortT, totalTime
```

(b) We have

$$\sum_{i=1}^{n} C_i = t_1 + (t_1 + t_2) + (t_1 + t_2 + t_3) + ... + t_n = nt_1 + (n-1)t_2 + ... + (n+1-i)t_i + (n-i)t_{i+1} + ... + t_n$$

Theorem 1: The total cost is minimum when $t_i \leq t_{i+1}$ for all i
Proof: assume that there exists some optimal job ordering that has $t_i > t_{i+1}$. Observe that there are (n + 1 - i) of t_i terms and (n - i) of t_{i+1} terms in the above summation. If we swap t_i and t_{i+1}, the summation will have one more t_{i+1} term and one less t_i term. Because $t_i > t_{i+1}$, the total cost after the swap will reduce, thus producing a not worse solution. From some optimal solution O, we can swap these inversions ($t_i > t_{i+1}$) until there is no inversions left in the ordering, which is exactly the solution of our greedy algorithm. And each swap guarantees to produce at least equally good result.

(c)
The algorithm takes $O(nlogn)$ to sort the list of jobs by time needed to complete the job. It then takes $O(n)$ time to iterate through the sorted list and accumulate the total time. The ordering is the order of the sorted list. In total, it takes $O(nlogn)$ time.

Problem 2
A wrong greedy algorithm for the Knapsack problem

(a) First, we find the list of bricks C that can be fit in the bags $c1, c2, ..., c_i$, then we iteratively take the heaviest brick in C until the bag is full.

Assume that B is the list of bricks sorted by weights.

Algorithm 2 $A(B, n)$

```
w ← 0
while n > 0 do
    Find the heaviest brick b < n
    w ← w + b
    n ← n − b
end while
    return w
```

(b)
Observe that the brick weights are equivalent to binary representation $2^0, 2^1, 2^2, ...$ Our algorithm is analogous to finding the binary representation of n. It starts with the most significant bit of n, then reduce n to, and repeat that process until n is reduced to 0. Because every number has a binary representation, the algorithm guarantees to fill up any given bag (completely full), hence produce the optimal solution.

(c)
With gold bricks having weights $\{2, 2, 3, 5\}$ and $n = 9$, the algorithm will not produce the optimal solution. In particular, the optimal solution is $\{2, 2, 5\}$, while the algorithm will yield $\{5, 3\}$.

Problem 3
A randomized algorithm for generating biased random bits

(a)
Let $P(1) = p$ and $P(0) = 1 - p$ denote the probability that ONEINTHREE returns 1 and 0, respectively. Let F denote the probability distribution of FAIRCOIN, with $F(1) = F(0) = 0.5$.

We have

$$P(0) = 0.5 + (1 - 0.5)P(1) = 0.5 + 0.5P(1)$$

We also have

$$P(1) = 1 - P(0) = 1 - 0.5 - 0.5P(1) = 0.5 - 0.5P(1)$$

Hence, $P(1) = \frac{1}{3}$

(b)
The function only terminates when FAIRCOIN returns 0. The expected number of times we need to call FAIRCOIN it returns 0 is $\frac{1}{0.5} = 2$.

(c)

Algorithm 3 $FAIRCOIN$

if $ONEINTHREE == 1$ **then return** 1
elsereturn $(1 - FAIRCOIN)(1 - FAIRCOIN)$
end if

We have

$$F(1) = P(1) + P(0)F(0)F(0) \qquad (1)$$

We also have

$$F(0) = 1 - F(1) \qquad (2)$$

Plugging (1) into (2) results in

$$F(0) = \frac{2}{3} - \frac{2}{3}F^2(0) \qquad (3)$$

Solving (3) we get $F(0) = 0.5$.

Problem 4
Tax Screening System

Chernoff inequality:

$$Pr[X > (1+\delta)\mu] < (\frac{e^{\delta}}{(1+\delta)^{1+\delta}})^{\mu}$$

$$Pr[X > (1-\delta)\mu] < e^{-\frac{1}{2}\mu\delta^2}$$