CS5050 Advanced Algorithms

Spring Semester, 2018

Homework Solution 1

Haitao Wang

1. (a) For $n = 100$, Sunway TaihuLight needs $\frac{2^{100}}{1.25 \cdot 10^{17}} \approx 1.014 \times 10^{13}$ seconds, which is about $\frac{2^{100}}{1.25 \cdot 10^{17} \cdot 3600 \cdot 24 \cdot 365 \cdot 100} \approx 3216$ centuries, to finish the algorithm.

   (b) For $n = 1000$, Sunway TaihuLight needs $\frac{2^{1000}}{1.25 \cdot 10^{17}} \approx 8.572 \times 10^{283}$ seconds, which is about $\frac{2^{1000}}{1.25 \cdot 10^{17} \cdot 3600 \cdot 24 \cdot 365 \cdot 100} \approx 2.7 \times 10^{274}$ centuries, to finish the algorithm.

2. Solution: 2^{500}, $\log(\log n)^2$, $\log n$, $\log_4 n$ $\log^3 n$, $\sqrt{n}$, $2^{\log n}$, $n \log n$, $n^2 \log^5 n$, n^3, 2^n, $n!$. Note that $\log n = \Theta(\log_4 n)$, so you may put $\log_4 n$ in front of $\log n$ as well.

3. For each of the following pairs of functions, indicate whether it is one of the three cases: $f(n) = O(g(n))$, $f(n) = \Omega(g(n))$, or $f(n) = \Theta(g(n))$.

   (a) $f(n) = 7 \log n$ and $g(n) = \log n^3 + 56$.
   Solution: $f(n) = \Theta(g(n))$.
   Note: If your answer is $f(n) = O(g(n))$ or $f(n) = \Omega(g(n))$, then you will get three partial points.

   (b) $f(n) = n^2 + n \log^3 n$ and $g(n) = 6n^3 + \log^2 n$.
   Solution: $f(n) = O(g(n))$.

   (c) $f(n) = 5^n$ and $g(n) = n^2 2^n$.
   Solution: $f(n) = \Omega(g(n))$. An easy way to see it is that $(\frac{5}{2})^n = \Omega(n^2)$.

   (d) $f(n) = n \log^2 n$ and $g(n) = \frac{n^2}{\log^3 n}$.
   Solution: $f(n) = O(g(n))$.

   (e) $f(n) = \sqrt{n} \log n$ and $g(n) = \log^8 n + 25$.
   Solution: $f(n) = \Omega(g(n))$.

   (f) $f(n) = n \log n + 6n$ and $g(n) = n \log_5 n - 8n$
   Solution: $f(n) = \Theta(g(n))$ because $\log_5 n = \frac{\log n}{\log_2 5}$.
   Note: If your answer is $f(n) = O(g(n))$ or $f(n) = \Omega(g(n))$, then you will get three partial points.

4. For each i with $1 \leq i \leq n$, denote by x_i the i-th item. So the size of the item x_i is k_i.

   Denote by S the knapsack that we are going to pack and let X be the total sum of the sizes of the items in S. In the beginning of the algorithm, we have $S = \emptyset$ and $X = 0$.

   We scan the items from x_1 to x_n one by one. Consider the i-th item x_i with $1 \leq i \leq n$.

- If $X+k_i \leq K$, then we set $S = S \cup \{x_i\}$, i.e., we pack the item x_i into S, and then update the value X by setting $X = X + k_i$. We check whether $X \geq K/2$. If yes, the current knapsack S is a factor of 2 approximation solution and we terminate the algorithm (this also means that if the algorithm is not terminated yet, then $X < K/2$ always holds); otherwise, we proceed on the next item x_{i+1}.
- If $X + k_i > K$, then the value k_i must be larger than $K/2$ since $X < K/2$. We check whether $k_i \leq K$. If yes, we make our knapsack S empty and let S pack the only item x_i. Since $k_i > K/2$, the current knapsack S is now a factor of 2 approximation solution and we terminate the algorithm. Otherwise, we proceed on the next item x_{i+1}.

When the algorithm stops, the knapsack S is a factor of 2 approximation solution.

Since we consider each item at most once and we spend constant time on considering each item, the running time of the algorithm is $O(n)$. The following is the pseudocode.

Algorithm 1: Finding a knapsack of factor 2 approximation solution

Input: The items $x_1, x_2, \ldots, x_n$ whose sizes are $k_1, k_2, \ldots, k_n$
Output: A knapsack S of factor 2 approximation solution

```
S ← ∅, X ← 0;
for i ← 1 to n do
    if X + k_i ≤ K then
        S ← S ∪ {x_i};
        X ← X + k_i;
        if X ≥ K/2 then
            break; /* terminate the algorithm                          */
        end
    else/* X + k_i > K                                                 */
        if k_i ≤ K then
            S ← ∅ ; /* empty the knapsack                              */
            S ← {x_i};
            break; /* terminate the algorithm                          */
        end
    end
end
```