CSE548 Fall 2019 Analysis of Algorithms Homework1

Tianao Wang 112819772

September 17, 2019

1. (a) Correct.
   In order to prove $max(f(n), g(n)) = \Theta(f(n)+g(n))$, we need to prove that there are c1,c2 and n0. When n>n0, c1 and c2 satisfy $0 <= c1(f(n) + g(n)) <= max(f(n) + g(n)) <= c2(f(n) + g(n))$.
   i. Due to $f(n)$ and $g(n)$ are non-negative functions, $c1(f(n) + g(n)) >= 0$.
   ii. When $c1 = \frac{1}{2}$, there are two situations. When $g(n) < f(n)$, so that $f(n) + g(n) < 2f(n)$, we can conclude that $\frac{1}{2}(f(n) + g(n)) <= f(n)$ which also means $\frac{1}{2}(f(n) + g(n)) <= max(f(n), g(n))$. In the same way, we can easily prove that, when $g(n) >= f(n)$, it also satisfy $\frac{1}{2}(f(n) + g(n)) <= max(f(n), g(n))$.
   iii. When $c2 = 1$ and f(n),g(n) are non-negative fuctions, we can prove that $max(f(n), g(n)) < c2(f(n) + g(n)))$

   In conclusion, there exist c1,c2 and n0 satisfying $0 <= c1(f(n) + g(n)) <= max(f(n) + g(n)) <= c2(f(n) + g(n))$. Therefore, $max(f(n), g(n)) = \Theta(f(n) + g(n))$

   (b) Correct.
   According to the definition, $of(n))$ means that there are c and n0, when $n > n0, 0 <= f(n) < cg(n)$. And $\omega(f(n))$ means that there are c and n0, when $n > n0, 0 <= cg(n) < f(n)$. Two sets of collections are not communicative. Therefore $o(f(n)) \cap \omega(f(n)) = \emptyset$.

   (c) Correct.
   According to the binomial formula, $(n + a)^b = \sum_{k=0}^{b} \binom{k}{b} n^k a^{b-k}$. When k = b, the Maximum number of power of n is b. The other power can be ignored. When n is extremely larger than a, we can consider $(n + a)^b$ as n^b. So $(n + a)^b = \Theta(n^b)$.

   (d) Incorrect.
   If $f(n) = n$, the equality becomes $n = O(n^2)$. Definitely, this equality is wrong.

   (e) Incorrect.
   If $f(n) = 2n, g(n) = n$, the conclusion becomes $2^{2n} = O(2^n)$ which means $4^n = O(2^n)$.Definitely, this equality is wrong.

2. $\sqrt{\lg n} < \lg\sqrt{n} < \lg n < 2^{\sqrt{\lg n}} = 2^{\lg\sqrt{n}} = 3^{\sqrt{\lg n}} = 3^{\lg\sqrt{n}} < \sqrt{n} = \sqrt{\lg(2^n)} = \sqrt{\lg(3^n)} < \sqrt{2^{\lg n}} = \sqrt{3^{\lg n}} = n = 2^{\lg n} = 3^{\lg n} = \lg(\sqrt{2^n}) = \lg(\sqrt{3^n}) = \lg(2^n) = \lg(3^n) < 2^{\sqrt{n}} < 3^{\sqrt{n}} < \sqrt{2^n} < \sqrt{3^n} < 2^n < 3^n$

3. (a) $O(n^3)$. The loop for i takes n times. The loop for j takes n times. The add operator takes n times. So the total running time is $O(n^3)$.

   (b) $\Omega(n^3)$. The outer loop takes n times. The inner loop takes $(n-1)+(n-2)+(n-3)+...+2+1 = \frac{1}{2}n^2 - \frac{1}{2}n$. The add action takes $\Omega(n)$ times. So the total running time is $\Omega(n^3)$

   (c) Use dynamic programming. Everytime we need to calculate $B[i][j]$, we can calculate the value of $B[i][j-1]+A[j]$. There is no need to add all the numbers each time. The running time is $O(n^2)$

4. (a) $O(n \lg n)$. There are 2 steps to make heapsort. First, we need to build up a heap which runing time is $O(n \lg n)$. Second, we need to change the position of nodes and rebuild the heap which runing time is $O(n \lg n)$. So the total running time is $O(n \lg n)$

   (b) $\Theta(n \lg n)$. There is no need to change nodes' position when build up a heap. However, we still have to change positions of nodes and rebuild the heap when output a increasing array. The running time is still $\Theta(n \lg n)$

   (c) $\Theta(n \lg n)$. Heapsort always keeps its running time no matter the format of input data.

5. According to the definition, the position on the top-left of the matrix is the minimum element. Remove the top-left element and then restore the matrix. The matrix is $m * n$ matrix so we only need to move m+n times. In conclusion, the running time is $O(m+n)$

Algorithm 1 Extract MIN

```
min = matrix[0][0];
matrix[0][0]= ∞;
i=0,j=0;
while matrix[i+1][j]!= ∞ && matrix[i][j+1]!= ∞ do
    if matrix[i+1][j]<matrix[I][j+1] then
        matrix[i][j] = matrix[i+1][j]
        matrix[++i][j] = ∞;
    else
        matrix[i][j] = matrix[i][j+1]
        matrix[i][++j] = ∞;
    end if
end while
return min
```