

# DSA Homework #3

## 3.1 Asymptotic Complexity

1. Let  $p(n) = an^2 + bn + c$

Then

$$\begin{aligned}
 \log(p(n)) &= \log(an^2 + bn + c) \\
 &\leq \log(an^2 + bnn + cn^2) \text{ (for } n \geq 1) \\
 &= \log((a + b + c)n^2) \\
 &= \log(a + b + c) + 2 \log n \\
 &\leq \log n + 2 \log n \text{ (for } n \geq a + b + c) \\
 &= 3 \log n
 \end{aligned}$$

That is for  $n \geq a + b + c$

$$\log(p(n)) \leq 3 \log n$$

Thus,  $\log p(n)$  is  $O(\log n)$

- 2.

$$\begin{aligned}
 \lceil f(n) \rceil &< f(n) + 1 \\
 \because f(n) &> 1 \\
 \therefore f(n) + 1 &< f(n) + f(n) \\
 \therefore \lceil f(n) \rceil &< 2f(n)
 \end{aligned}$$

Thus,  $\lceil f(n) \rceil$  is  $O(f(n))$

3. From the definition of limit, for any  $\varepsilon > 0$  there is a  $N$  such that  $\forall n > N$ ,

$$\begin{aligned}
 \left| \frac{f(n)}{g(n)} - A \right| &< \varepsilon \\
 \Rightarrow A - \varepsilon &< \frac{f(n)}{g(n)} < A + \varepsilon \\
 \Rightarrow g(n)(A - \varepsilon) &< f(n) < g(n)(A + \varepsilon)
 \end{aligned}$$

That is, for any constants  $c_1 = A - \varepsilon$  and  $c_2 = A + \varepsilon$  there is a  $N$  such that  
 $c_1 g(n) < f(n) < c_2 g(n)$  ( for  $n > N$  )  
 Thus,  $f(n) = \Theta(g(n))$

4.

$$40n_0^2 \leq 2n_0^3$$

$$n_0 \geq 20$$

5. (a) 

```
sum = 0;
for ( i = 0 ; i < n ; ++i ){
    power = 1;
    for ( t = 0 ; t < i ; ++t ){
        power = power * x;
    }
    sum = sum + a_i ;
}
return sum;
```

(b)

$O(n)$

6. Consider  $f(n) = n^3$  and  $g(n) = n$ ,  
 $\log(f(n)) < 4 \log(g(n))$  for all  $n > 0$   
 $\therefore \log(f(n)) = O(\log g(n))$   
 However,  $f(n) > g(n)$  for all  $n > 1$   
 $\therefore f(n) \neq O(g(n))$

## 3.2 Stack, Queue, Deque

1.
  - **Step1:** Pop an element from the stack and push it back to the queue  $n$  times.
  - **Step2:** Pop an element from the front of the queue and push it back both to the stack
  - **Step3:** Pop an element from the stack and push it back to the queue  $n$  times.
  - **Step4:** Pop an element from the front of the queue and push it back both to the stack and to the queue  $n$  times.
  - **Step5:** Repeat popping an element from the front of the queue until the queue is empty or the popped element is equal to  $x$ . If any element is equal to  $x$ , return true; otherwise, return false.
2.
  - **Step1:** Repeat popping an element from the stack  $S$  and push it to the front of the dequeue until  $S$  is empty.
  - **Step2:** Repeat popping an element from the stack  $T$  and push it to the front of the dequeue until  $T$  is empty.

- **Step3:** Repeat popping an element from the front of the queue and push it back to the stack  $S$  until the queue is empty.
- 3.
- 4.
- **Step1:** Repeat popping an element from stake  $S$  and push it into the third stake until stake  $S$  is empty.
  - **Step2:** Repeat popping an element from stake  $T$  and push in into the third stake until stake  $T$  is empty.
  - **Step3:** Repeat popping an element from the third stake and push it into stake  $S$  until the third stake is empty.

### 3.3 List, Iterator

1. Each time the vector resize, its capacity becomes  $\frac{5}{4}$  of its original capacity.  
Let  $t$  = times a vector of a capacity  $n$  needs to resize.

$$\frac{\left(\frac{5}{4}\right)^t - 1}{\frac{5}{4} - 1} = n$$

Each time the vector resize, it has to copy all the element in the original array.  
So the sum of times it copy is:

$$\begin{aligned} & \sum_{i=1}^t \left(\frac{5}{4}\right)^{n-1} \\ &= \frac{\left(\frac{5}{4}\right)^t - 1}{\frac{5}{4} - 1} \\ &= n \end{aligned}$$

Therefore, the time complexity is  $O(n)$ .

- 2.
- **Algorithm:** Swap the **randomInteger(i)**-th with the  $i$ -th element for  $i = n - 1$  to  $i = 0$ .
  - **Explain:** The concept of that algorithm is to pick up an element from a array randomly and push it into another array for “number of elements” times. Because the total number of the elements is constant, the other array storing the result can simply be kept in the end of the original array. As each time choosing a element in the original array, every elements has equal probability to be chosen. From the senior high school math, we know that each element has the same probability to be the  $i$ -th chosen element, for all  $i$  greater than 0 and not greater than the number of elements. Therefore, every possible ordering of the result array that store the picked elements is equally likely.
  - **Running time:** The algorithm picks and swap elements for “number of elements” times. Thus the running time is  $O(n)$ .