# CS5800: Algorithms — Virgil Pavlu

Homework 1
Due : 05/23/2022

Name: Xuran Feng

Instructions:

- Make sure to put your name on the first page. If you are using the LaTeX template we provided, then you can make sure it appears by filling in the `yourname` command.
- Please review the grading policy outlined in the course information page.
- You must also write down with whom you worked on the assignment. If this changes from problem to problem, then you should write down this information separately with each problem.
- Problem numbers (like Exercise 3.1-1) are corresponding to CLRS $3^{rd}$ edition. While the $2^{nd}$ edition has similar problems with similar numbers, the actual exercises and their solutions are different, so make sure you are using the $3^{rd}$ edition.

**1.** *(20 points)*

Two linked lists (simple link, not double link) heads are given: headA, and headB; it is also given that the two lists intersect, thus after the intersection they have the same elements to the end. Find the first common element, without modifying the lists elements or using additional data structures.

(a) A linear algorithm is discussed in the lecture: count the lists first, then use the count difference as an offset in the longer list, before traversing the lists together. Write a formal pseudocode (the pseudocode in the lecture is vague), using "next" as a method/pointer to advance to the next element in a list.

**Solution:**
ListNode getIntersection(ListNode headA, ListNode headB):
  lengthA=list length of headA;
  lengthB=list length of headB;
  offset=lengthA-lengthB;
  if(offset<0):
    offset=-offset;
    switch headA and headB;
  iterA=headA; iterB=headB;
  **move** iterA ahead until eliminating offset;
  **move** iterA and iterB at same pace until they reach same node;
  **r**eturn iterA or iterB;

(b) Write the actual code in a programming language (C/C++, Java, Python etc) of your choice and run it on a made-up test pair of two lists. A good idea is to use pointers to represent the list linkage.

**Solution:**The complete code demo is attached at end of homework file, functions containing algorithms are pasted here.

---

```
//'offset' algorithms
static ListNode getIntersectionNode(ListNode headA, ListNode headB) {
       int lengthA=getLength(headA);
       int lengthB=getLength(headB);
       int offset=lengthA-lengthB;
       if(offset<=0){
            offset=lengthB-lengthA;
            ListNode tmp=headA;
            headA=headB;
            headB=tmp;
        }
       int i=0;
       ListNode iterA=headA; ListNode iterB=headB;
       while(i<offset){iterA=iterA.next;i++;}
       while(iterA!=iterB){iterA=iterA.next;iterB=iterB.next;}
       return iterB;
 }
```

```java
//helper fucnction to get list length
static int getLength(ListNode head){
       ListNode iter=head;
       int i=0;
       while(iter!=null){iter=iter.next;i++;}
       return i;
 }
```

---

**2.** ***(10 points)*** *Exercise 3.1-1*
*Because f(n) and g(n) are asymptotically nonnegative functions, $\exists n_1$ that satisfies f(n)≥0 for $n \geq n_1$ and $\exists n_2$ that satisfies g(n)≥0 for $n \geq n_2$, we choose $n_0$=max($n_1$,$n_2$).*

*(1) for any $n \geq n_0$,*
*max(f(n),g(n)) ≥ f(n),*
*max(f(n),g(n)) ≥ g(n),*
*2∗max(f(n),g(n)) ≥ f(n)+g(n),*
*max(f(n),g(n)) ≥ (f(n)+g(n))/2 ≥ 0*

*(2) for any $n \geq n_0$,*
*max(f(n),g(n)) ≤ f(n)+g(n) based on the definition of max function*

*We are asked to prove max(f(n),g(n))=θ(f(n)+g(n)), according to definition, we need to prove there exists $c_2$>$c_1$>0, 0≤$c_1$∗(f(n)+g(n))≤max(f(n),g(n)) ≤$c_2$∗(f(n)+g(n)) after n reaching some $n_0$, from (1) and (2) we can find $c_1$=$\frac{1}{2}$ and $c_2$=1.*

**3.** ***(5 points)*** *Exercise 3.1-4*

*1) Yes, $2^{n+1}$=O($2^n$), by definition of Big O, there exists a positive constant c and positive constant $n_0$ such that 0≤$2^{n+1}$≤c∗$2^n$ for $n \geq n_0$. $2^{n+1}$=2∗$2^n$, we can let c=2 and thus $n_0$ equals any positive constant.*

*2) No, $2^{2n}$=O($2^n$) is false, by definition of Big O, there exists a positive constant c and positive constant $n_0$ such that 0≤$2^{2n}$≤c∗$2^n$ for $n \geq n_0$. If we divide $2^n$ on both sides, then we need to find 0≤$2^n$≤c for $n \geq n_0$, however $2^n$ is asymptotically reaching positive infinity when n is large enough, we cannot find a constant c to satisfy this equation.*

**4.** ***(15 points)***

Rank the following functions in terms of asymptotic growth. In other words, find an arrangement of the functions $f_1, f_2, ...$ such that for all i, $f_i = \Omega(f_{i+1})$.

$$\sqrt{n}\ln n \quad \ln\ln n^2 \quad 2^{\ln^2 n} \quad n! \quad n^{0.001} \quad 2^{2\ln n} \quad (\ln n)!$$

The rank will be $n!$, $2^{\ln^2 n}$, $(\ln n)!$, $2^{2\ln n}$, $\sqrt{n}\ln n$, $n^{0.001}$, $\ln\ln n^2$ following $f_i = \Omega(f_{i+1})$ requirement.

Based on what professor taught in lecture, $n! \geq 2^{\ln^2 n}$ by converting equation into $n! \geq (\frac{n}{2})^{\frac{n}{2}}$; next we need to prove $2^{\ln^2 n} \geq (\ln n)!$, we know $(\ln n)! \leq (\ln n)^{\ln n}$ and $2^{\ln^2 n} \simeq n^{\ln n}$, and $n^{\ln n} \geq (\ln n)^{\ln n}$, so $2^{\ln^2 n} \geq (\ln n)!$ can be proved based on the transitive property; next we need to prove $(\ln n)! \geq 2^{2\ln n}$, we know $(\ln n)! \geq (\frac{\ln n}{2})^{\frac{\ln n}{2}}$, so if we can prove $(\frac{\ln n}{2})^{\frac{\ln n}{2}} \geq 2^{2\ln n}$, we are done. By taking natural logarithm of both sides, we get $\frac{\ln n}{2}*\ln(\frac{\ln n}{2}) \geq 2*\ln n$ and we eliminate $\ln n$ on both sides to get $\ln(\frac{\ln n}{2}) \geq 4$, there should exist a constant $n_0$ to satisfy this; next we prove $2^{2\ln n} \geq \sqrt{n}\ln n$, left side $2^{2\ln n} = 2^{\ln n^2} \simeq n^2$, $n^2 \geq \sqrt{n}\ln n = n^{1.5} \geq \ln n$ which is valid because polynomial is always larger than logarithm; next we need to prove $\sqrt{n}\ln n \geq n^{0.001}$, since $\sqrt{n}$=$n^{0.5} \geq n^{0.001}$, this is also true; last we need to prove $n^{0.001} \geq \ln\ln n^2$, $\ln\ln n^2$=$\ln(2\ln n)$=$\ln 2$+$\ln\ln n$, since polynomial and linear are always larger than logarithm and constant number $\ln 2$ can be safely ignored here, we get $n^{0.001} \geq \ln n \geq \ln\ln n$, so the whole rank is proved.

**5.** ***(40 points)*** *Problem 4-1 (page 107)*

Give asymptotic upper and lower bounds for T(n) in each of the following recurrences. Assume that T(n) is constant for $n \leq 2$. Make your bounds as tight as possible, and justify your answers.

(a) $T(n) = 2T(n/2) + n^4$
Base version of Master Theorem can be applied, a=2, b=2, c=4 and $\frac{a}{b^c}$<1, T(n)=$\theta(n^c)$=$\theta(n^4)$.

(b) $T(n) = T(7n/10) + n$
Base version of Master Theorem can be applied, a=1, b=$\frac{10}{7}$, c=1 and $\frac{a}{b^c}$<1, T(n)=$\theta(n^c)$=$\theta(n)$.

(c) $T(n) = 16T(n/4) + n^2$
Base version of Master Theorem can be applied, a=16, b=4, c=2 and $\frac{a}{b^c}$=1, T(n)=$\theta(n^{log_b a}\log n)$=$\theta(n^2\log n)$.

(d) $T(n) = 7T(n/3) + n^2$
Base version of Master Theorem can be applied, a=7, b=3, c=2 and $\frac{a}{b^c}$<1, T(n)=$\theta(n^c)$=$\theta(n^2)$.

(e) $T(n) = 7T(n/2) + n^2$
Base version of Master Theorem can be applied, a=7, b=2, c=2 and $\frac{a}{b^c}$>1, T(n)=$\theta(n^{log_b a})$=$\theta(n^{log_2 7})$.

(f) $T(n) = 2T(n/4) + \sqrt{n}$
Base version of Master Theorem can be applied, a=2, b=4, c=$\frac{1}{2}$ and $\frac{a}{b^c}$=1, T(n)=$\theta(n^{log_b a}\log n)$=$\theta(n^{0.5}\log n)$.

(g) $T(n) = T(n-2) + n^2$
$T(n) = T(n-2) + n^2 = T(n-4) + (n-2)^2 + n^2 = T(n-6) + (n-4)^2 + (n-2)^2 + n^2$, the general pattern is $T(n) = T(n-k) + (n-(k-2))^2 + (n-(k-4))^2 + ... + n^2$, the base case is n=k, and the series will be $2^2 + 4^2 + 6^2 + ... + n^2$ =$(2\cdot1)^2$+$(2\cdot2)^2$+$(2\cdot3)^2$ + ...+$(2\cdot\frac{n}{2})^2$ = $2^2\cdot(1^2 + 2^2 + 3^2 + ... + (\frac{n}{2})^2)$ = $4\cdot\frac{\frac{n}{2}(\frac{n}{2}+1)(n+1)}{6} = \theta(n^3)$.

**6.** ***(30 points)*** *Problem 4-3 from (a) to (f) (page 108)*

Give asymptotic upper and lower bounds for T(n) in each of the following recurrences. Assume that T(n) is constant for sufficiently small n. Make your bounds as tight as possible, and justify your answers.

(a) $T(n) = 4T(n/3) + n\lg n$
$T(n) = \theta(n^{log_3 4})$, can use iteration method to solve this. $T(n) = 4T(\frac{n}{3}) + n\lg n = 4[4T(\frac{n}{9}) + \frac{n}{3}\lg\frac{n}{3}] + n\lg n = 4^2[4T(\frac{n}{27}) + \frac{n}{9}\lg\frac{n}{9}] + \frac{4n}{3}\lg\frac{4n}{3} + n\lg n = ...$, the general pattern is $T(n) = 4^k T(\frac{n}{3^k}) + (\frac{4}{3})^{k-1} n\lg\frac{n}{3^{k-1}} + (\frac{4}{3})^{k-2} n\lg\frac{n}{3^{k-2}} + ... + n\lg n$. The base case happens when $n = 3^k, k = \log_3 n$, thus

T(n) can be written $T(n) = 4^{\log_3 n} T(1) + \sum_{j=0}^{\log_3 n-1} (\frac{4}{3})^j (\log_3 n - j) = 4^{\log_3 n} T(1) + \sum_{j=1}^{\log_3 n} (\frac{4}{3})^{\log_3 n - j} \cdot j = 4^{\log_3 n} T(1) + (\frac{4}{3})^{\log_3 n} \sum_{j=1}^{\log_3 n} j \cdot (\frac{3}{4})^j$, we then need to compute the result of $\sum_{j=1}^{\log_3 n} j \cdot (\frac{3}{4})^j$, we let $S = \sum_{j=1}^{\log_3 n} j \cdot (\frac{3}{4})^j$, we multiply $\frac{3}{4}$ on both sides to get $\frac{3}{4} S = \sum_{j=1}^{\log_3 n} j \cdot (\frac{3}{4})^{j+1}$, we then substract the second equation from first equation to get $\frac{S}{4} = \frac{3}{4} + (\frac{3}{4})^2 + (\frac{3}{4})^3 + ... + (\frac{3}{4})^{\log_3 n} - \log_3 n \cdot (\frac{3}{4})^{log_3 n+1}$ we can rearrange this equation using geometric series sum and get $S = 12 - \frac{12n}{n^{\log_3 4}} (1 + \frac{\log_3 n}{4})$, finally we can plug this S into $T(n) = 4^{\log_3 n} T(1) + (\frac{4}{3})^{\log_3 n} \sum_{j=1}^{\log_3 n} j \cdot (\frac{3}{4})^j = 4^{\log_3 n} T(1) + (\frac{4}{3})^{\log_3 n} \cdot S = 4^{\log_3 n} T(1) + \frac{12 n^{\log_3 4}}{n} - 12(1 + \frac{\log_3 n}{4}) = n^{\log_3 4} T(1) + \frac{12 n^{\log_3 4}}{n} - 12(1 + \frac{\log_3 n}{4}) = n^{\log_3 4} T(1) + 12 n^{\log_3 4 - 1} - 12(1 + \frac{\log_3 n}{4})$, we can see that $n^{\log_3 4}$ is the dominant complexity in this final equation and thus $T(n) = \theta(n^{log_3 4})$ is proved.

(b) $T(n) = 3T(n/3) + n/\lg n$
$T(n) = \theta(n \log_3 \log_3 n)$, can use iteration method to solve this. $T(n) = 3T(\frac{n}{3}) + \frac{n}{\lg n} = 3[3T(\frac{n}{9}) + \frac{\frac{n}{3}}{\lg \frac{n}{3}}] + \frac{n}{\lg n} = 3^2[3T(\frac{n}{27}) + \frac{\frac{n}{9}}{\lg \frac{n}{9}}] + \frac{n}{\lg \frac{n}{3}} + \frac{n}{\lg n}$, the general pattern is $T(n) = 3^k T(\frac{n}{3^k}) + n(\frac{1}{\lg n} + \frac{1}{\lg \frac{n}{3}} + ... + \frac{1}{\lg \frac{n}{3^{k-1}}})$, the base case happens when $n = 3^k, k = \log_3 n$, and now we rearrange $T(n) = nT(1) + \sum_{j=0}^{k-1} n \cdot \frac{1}{\lg \frac{n}{3^j}} \simeq nT(1) + n \sum_{j=0}^{\log_3 n-1} \frac{1}{\lg n - j} = nT(1) + n \sum_{j=1}^{\log_3 n} \frac{1}{j}$, according to harmonic series, $T(n) \simeq nT(1) + n\theta(\log_3 \log_3 n)$, then we can prove that $T(n) = \theta(n \log_3 \log_3 n)$.

(c) $T(n) = 4T(n/2) + n^2\sqrt{n}$,
$T(n) = \theta(n^2\sqrt{n})$, can use iteration method to prove this. $T(n) = 4T(n/2) + n^2\sqrt{n} = 4[4T(n/4) + (n/2)^2 \cdot \sqrt{\frac{n}{2}}] + n^2\sqrt{n} = 4^2[4T(n/8) + (n/4)^2 \cdot \sqrt{\frac{n}{4}}] + n^2\sqrt{\frac{n}{2}} + n^2\sqrt{n}$, the general pattern is $T(n) = 4^k T(\frac{n}{2^k}) + n^2\sqrt{n}(1 + \sqrt{\frac{1}{2}} + \sqrt{\frac{1}{4}} + ... + \sqrt{\frac{1}{2^{k-1}}})$, the base case is $n = 2^k, k = \log_2 n$, so T(n) can be rearranged as $T(n) = n^{\log_2 4} T(1) + n^2\sqrt{n} \sum_{j=0}^{\log_2 n-1} \sqrt{\frac{1}{2^j}}$, however, the geometric series has a ratio $\sqrt{\frac{1}{2}}$ which is smaller than 1, so the geometric series will converge to a constant, so $T(n) \simeq n^2 T(1) + n^2\sqrt{n} \cdot c$, and the dominant complexity is $\theta(n^2\sqrt{n})$.

(d) $T(n) = 3T(n/3 - 2) + n/2$
$T(n) = \theta(n \log_3 n)$, we first use iteration method to make a guess, then we prove our guess. $T(n) = 3T(\frac{n-6}{3}) + \frac{n}{2} = 3[3T(\frac{\frac{n-6}{3}-6}{3}) + \frac{\frac{n-6}{3}}{2}] + \frac{n}{2} = 3^2[3T(\frac{\frac{n-24}{9}-6}{3}) + \frac{\frac{n-24}{9}}{2}] + \frac{n-6}{2} + \frac{n}{2} = 3^3 T((\frac{\frac{n-24}{9}-6}{3}) + \frac{n-24}{2} + \frac{n-6}{2} + \frac{n}{2}$, although there is a constant part in recursive part, we can male a guess that base case is $n = 3^k, k = \log_3 n$ and non-recursive time complexity is $\theta(\frac{n}{2} \cdot k) = \theta(n \log_3 n)$ while recursive part time complexity is $3^k T(1) = \theta(n)$; we now prove $\theta(n \log_3 n)$ is the time complexity of T(n) by using induction.
Our induction hypothesis for lower bound is $T(\frac{n-6}{3}) = \theta(\frac{n-6}{3} \cdot \log_3 \frac{n-6}{3})$ and since constant 6 can be asymptotically ignored here, $\frac{n-6}{3} \cdot \log_3 \frac{n-6}{3} \leq \frac{n}{3} \cdot \log_3 \frac{n}{3}$, we can write $T(\frac{n-6}{3}) = \theta(\frac{n}{3} \cdot \log_3 \frac{n}{3})$; induction step: $T(n) \geq 3c_1 \frac{n}{3} \log_3 \frac{n}{3} + \frac{n}{2}$, to make our induction conclusion true, we need to find $c_1$ to satisfy $3c_1 \frac{n}{3} \log_3 \frac{n}{3} + \frac{n}{2} \geq c_1 n \log_3 n$, after rearranging, we find $c_1 \leq \frac{1}{2}$; for upper bound, we apply same process and rearrange $3c_2 \frac{n}{3} \log_3 \frac{n}{3} + \frac{n}{2} \leq c_2 n \log_3 n$ and find $c_2 \geq \frac{1}{2}$. Since $c_1$ and $c_2$ can be found, we finish our proof.

(e) $T(n) = 2T(n/2) + n/\lg n$
$T(n) = \theta(n \log_2 \log_2 n)$, the whole process is almost same as part(b) with only different param-

eters. $T(n) = 2T(\frac{n}{2}) + \frac{n}{\lg n} = 2[2T(\frac{n}{4}) + \frac{\frac{n}{2}}{\lg \frac{n}{2}}] + \frac{n}{\lg n} = 2^2[2T(\frac{n}{8}) + \frac{\frac{n}{4}}{\lg \frac{n}{4}}] + \frac{n}{\lg \frac{n}{2}} + \frac{n}{\lg n}$, the general pattern is $T(n) = 2^k T(\frac{n}{2^k}) + n(\frac{1}{\lg n} + \frac{1}{\lg \frac{n}{2}} + ... + \frac{1}{\lg \frac{n}{2^{k-1}}})$, the base case happens when $n = 2^k, k = \log_2 n$, and now we rearrange $T(n) = nT(1) + \sum_{j=0}^{k-1} n \cdot \frac{1}{\lg \frac{n}{2^j}} \simeq nT(1) + n\sum_{j=0}^{\log_2 n-1} \frac{1}{\lg n - j} = nT(1) + n\sum_{j=1}^{\log_2 n} \frac{1}{j}$, according to harmonic series, $T(n) \simeq nT(1) + n\theta(\log_2 \log_2 n)$, then we can prove that $T(n) = \theta(n \log_2 \log_2 n)$.

(f) $T(n) = T(n/2) + T(n/4) + T(n/8) + n$
$T(n) = \theta(n)$, we can prove this by first drawing the recursion tree, make a guess and prove the guess by definition. The recursion tree is unbalanced, its longest branch depth is $\log_2 n$ and shortest branch depth is $\log_8 n$, the number of all leaf nodes of this recursion tree falls into $3^{\log_2 n}$ to $3^{\log_8 n}$ becuase each layer has three children branches, which is $n^{\log_2 3}$ to $n^{\log_8 3}$(borders are exclusive); each layer also has non-recursive part, first layer is $n$, next layer is $\frac{n}{2} + \frac{n}{4} + \frac{n}{8} = \frac{7}{8}n$, third layer is $\frac{n}{4} + \frac{n}{8} + \frac{n}{16} + \frac{n}{8} + \frac{n}{16} + \frac{n}{32} + \frac{n}{16} + \frac{n}{32} + \frac{n}{64} = (\frac{7}{8})^2 n$, the sum of non-recursive part is a geometric series times $n$ and the geometric series has a ratio less than 1, so the non-recursive part has time complexity $= \theta(n)$; now recursive part has time complexity from $n^{\log_2 3}$ to $n^{\log_8 3}$(borders exclusive), while non-recursive part has $\theta(n)$, we can guess $T(n) = \theta(n)$ and prove it below.
Induction hypothesis for upper bound: $T(\frac{n}{2}) \leq c_2 \cdot \frac{n}{2}$, $T(\frac{n}{4}) \leq c_2 \cdot \frac{n}{4}$, $T(\frac{n}{8}) \leq c_2 \cdot \frac{n}{8}$, induction step: $T(n) = T(\frac{n}{2}) + T(\frac{n}{4}) + T(\frac{n}{8}) + n \leq c_2 \cdot \frac{n}{2} + c_2 \cdot \frac{n}{4} + c_2 \cdot \frac{n}{8} + n = n(\frac{7c_2}{8} + 1)$, to make induction conclusion $T(n) \leq c_2 \cdot n$ true, we need to find a $c_2$ satisfying $n(\frac{7c_2}{8} + 1) \leq c_2 \cdot n$, solving the equation we get $c_2 \geq 8$; for lower bound, we apply same process and solving $n(\frac{7c_1}{8} + 1) \geq c_1 \cdot n$, solving the equation we get $c_1 \leq 8$.
We now prove $T(n) = \theta(n)$.

```java
class findIntersection {
    static class ListNode {
        int val;
        ListNode next;
        ListNode(int x) {
            val = x;
            next = null;
        }
    }

    static ListNode getIntersectionNode(ListNode headA, ListNode headB) {
        int lengthA=getLength(headA);
        int lengthB=getLength(headB);
        int offset=lengthA-lengthB;
        if(offset<=0){offset=lengthB-lengthA;ListNode tmp=headA;headA=headB;headB=tmp;}
        int i=0;
        ListNode iterA=headA; ListNode iterB=headB;
        while(i<offset){iterA=iterA.next;i++;}
        while(iterA!=iterB){iterA=iterA.next;iterB=iterB.next;}
        return iterB;
    }

    static int getLength(ListNode head){
        ListNode iter=head;
        int i=0;
        while(iter!=null){iter=iter.next;i++;}
        return i;
    }

    public static void main(String[] args) {
        ListNode node1=new ListNode(1);ListNode node2=new ListNode(2);
        ListNode node3=new ListNode(3);ListNode node4=new ListNode(4);
        ListNode node5=new ListNode(5);ListNode node6=new ListNode(8);

        //headA: 1->2->3->4->5
        //headB:    8->3->4->5
        node1.next=node2;node2.next=node3;node3.next=node4;node4.next=node5;
        node6.next=node3;

        ListNode headA1,headB1;
        headA1=node1;headB1=node6;

       ListNode intersectNode1=getIntersectionNode(headA1,headB1);
       System.out.println("Two lists intersect at "+intersectNode1+" with node value "+intersectNode1.val);

        //headA: 1->2->3->4->5
        //headB:          4->5
       ListNode headA2,headB2;
       headA2=node1;headB2=node4;

       ListNode intersectNode2=getIntersectionNode(headA2,headB2);
       System.out.println("Two lists intersect at "+intersectNode2+" with node value "+intersectNode2.val);

    }
}
```