Name: Chaoyu Li

USC ID:6641732094

Date:9/4/2021

CSCI 570 - Fall 2021 - HW 2

Due September 9th

1.1 The worst-case runtime performance is O(nlogn). For the While loop, it is executed logn times at most because this is a binary algorithm. For the For loop, its complexity is a polynomial of n+n/2+n/4…

1.2 We try to simplify all these functions:

* 1. 2logn=2log2n=n;
  2. 23n;
  3. nnlogn;
  4. logn;
  5. nlog(n2)=2nlogn;
  6. nn2;
  7. log(log(nn))=log(nlogn);

It is easy to get:

1. O(nnlogn)⊂O(nn2);
2. O(nlog(n2))=O(nlogn);
3. O(logn)⊂O(log(log(nn)))=O(log(nlogn));
4. O(2logn)=O(n);

We know that it is alway has exponential function> linear function> logarithmic function. So we got O(logn)⊂O(log(log(nn)))⊂O(2logn)⊂O(nlog(n2))⊂O(nnlogn)⊂O(nn2).

Now, the only function we left is 23n. We know O(23n)>O(nlogn), so we want to compare it with nnlogn.

We suppose the function a(n)=3n; b(n)=nlogn. We got a(n)=O(b(n)) because for all n≥1, a(n)≤cb(n), c=3.

We suppose the function c(n)=2nlogn; d(n)=nnlogn. We got c(n)=O(d(n)) because for all n≥2, c(n)<cd(n), c=1.

So we got O(23n )⊂O(nnlogn).

The final answer is:

**O(logn)⊂O(log(log(nn)))⊂O(2logn)⊂O(nlog(n2))⊂O(23n )⊂O(nnlogn)⊂O(nn2).**

1.3 f1(n)=O(g1(n)),f2(n)=O(g2(n))

(a) f1(n) \* f2(n)=O(g1(n) \* g2(n))

**True.** Because f1(n)≤c1g1(n), f2(n)≤c2g2(n);

So f1(n) \* f2(n)≤c1g1(n) \* c2g2(n)

=c1c2(g1(n)\*g2(n))

≤c(g1(n)\*g2(n)) ,c≥c1c2

=O(g1(n) \* g2(n)).

(b) f1(n)+f2(n)=O(max(g1(n),g2(n)))

**True.** Because f1(n)≤c1g1(n), f2(n)≤c2g2(n);

So f1(n) + f2(n)≤c1g1(n) + c2g2(n)

≤c1max(g1(n),g2(n)) + c2max(g1(n),g2(n))

=(c1+c2)max(g1(n),g2(n))

≤c(max(g1(n),g2(n))) ,c≥c1c2

=O(max(g1(n),g2(n))).

(c) f1(n)2=O(g1(n)2)

**True.** Because f1(n)≤c1g1(n);

So f1(n)2≤c12g1(n)2

≤cg1(n)2 ,c≥c12

=O(g1(n)2)

(d) log2f1(n)=O(log2g1(n))

**False.**

Counterexample:

We suppose that f1(n)=2, g1(n)=1. It satisfies that f1(n)=O(g1(n)).

However, log2f1(n)=1, log2g1(n)=0. For any constant c, log2f1(n)>clog2g1(n).

1.4 The algorithm based on DFS.

Algorithm:

Set a flag to 0 representing there is not cycle in the graph

While there is a node u that hasn't been chosen:

Mark u as “Explored”

For each edge (u,v) that incident to u:

If v is not marked “Explored”:

Recursively invoke DFS(v)

Else:

Set flag to 1 representing there is a cycle in the graph

Endif

Endfor

Endwhile

If flag=0:

Return “there is not cycle in the graph”

Else:

Return “there is a cycle in the graph”

Endif

2.1

1. f(n)=n3.
   1. For the outermost loop, it always runs for *n* times.
   2. For the loop of *j*, in the worst case it also runs *n* times.
   3. Then the inner loop is to count the sum from a[i] to a[j]. It runs (j-i) times, but in the worst case j=n, i=1, it also runs *n* times.
   4. For b[i,j]=sum(a[i],a[j]), it only runs 1 time each loop.
   5. Therefore, the bound of the O(f(n))=O(n\*n\*n)=O(n3).
   6. First, for loop i and loop j, they always run n(n-1)/2 times for the whole execution.
   7. For the loop for counting the sum from a[i] to a[j], we know it runs at least 1 time for each time of loop i. So it runs at least (n-1) times for the whole execution.
   8. Therefore, the whole execution runs at least n(n-1)(n-1)/2=(n3-2n2+n)/2 times, Ω(n)=n3.

(c) Consider b[i,j] represents the sum from a[i] to a[j], then we have a transfer equation that b[i,j]=b[i,j-1]+a[j].

So the algorithm can be designed as below:

For i from 1 to n:

Set b[i,i] to a[i]

Endfor

For i from 1 to n-1:

For j from i+1 to n:

Set b[i,j] to b[i,j-1]+a[j]

Endfor

Endfor

This algorithm will only run n(n-1) times at most, so its complexity is O(n2).

2.2 (a) Let us suppose that there is an edge {n,m} in G that does not belong to T.

(b) Because T is a DFS tree, one of the n and m must be the ancestor of the other.

(c) Let us suppose that n is the ancestor of m.

(d) Because edge {n,m} does not belong to T, so there must be a node k between n and m and there is a cycle between these three nodes like “n-k-m-n”, otherwise edge {n,m} must be searched in the process of DFS.

(e) In this case, if we use the BFS algorithm to search G, when it goes to n, k and m are both its direct children. Let us suppose that we got T’ after searching the whole graph.

(f) T’ must contain edge {n,m} because n is the direct parent of m. Then we got T≠T’.

(g) However, in the assumption, the result of DFS and BFS in G are both T. This is a contradiction.