CSOR W4246–Fall, 2020
Name: Rebecca He
UNI: zh2449

Homework 1 Theoretical Solutions (150 points)

1. **Solution to Problem 1**

   Algorithm: we first find the value of n in O(logn) and use Binary search to find the index in O(logn) time. To find the value in O(logn) time, we first set left=0 and right=1, compare from A[0] to A[right] and the input integer, if it is greater than right index element then copy right index in left index and double the right index ; if it is smaller, then apply binary search on left and right indices found.

   The worst run time will exam k right indices, where A[2^k]==∞, to leave the comparison, thus $2^{k-1} \leq n \leq 2^k$, thus run time is O(log2^k), the binary search will search k times and thus run time is O(log2^k), thus total run time is O(log2^k). Since $2^k \leq 2n$, thus O$(log_2(2n)) = O(logn)$

   correctness is the same as binary search; $O(logn)$ run time

---

Algorithm 1 Algorithm binary search

Input 1. a sorted array A of n integers. 2. integer x
Output The index of x if exist; ∞ if not.

  procedure Search(A,x)
    left=0, right=1,value=A[0]
    if $x > value$ **then**
      left = right, right= 2*right, value=A[right]
    end if
    return BinarySearch(A,x,left,right)
  end procedure

---

2. **Solution to Problem 2**

   (a) a time: $O(n) * O(n-1) = O(n^2)$
   correctness: iterativly go through all $n > j > i$ for each i and compare,if A[i] is greater, count pair, to get total number of pairs of disorder

---

Algorithm 2 Algorithm brute force

Input a sequence of n distinct numbers

Output number of pairs of disorder

```
procedure Bruteforce(A)
    pair=0,n=len(A)
    for i ∈ [1, n] do
        while j ∈ [i+1,n] do
            if A[i]>A[j] then
                pair+=1
            end if
        end while
    end for
    return pair
end procedure
```

(b) Algorithm: Mergesort(A,l,r) recursively divide the array into sub-problems and count the pairs of disorder. Then use Merge to combine the sub-arrays and count the total number of pairs.
run time of mergesort: O(nlogn); Merge is O(n) time
correctness of Mergesort and Merge is proven in class

Algorithm 3 Algorithm MergeSort Count

---

input: a sequence of n distinct numbers
output: number of pairs of disorder

```
procedure MergeSort(A,l,r)
    pair=0
    if l = r then return 0
    else if l < r then
        mid=(l+r)//2
        pair += MERGESORT(A,l,mid)
        pair += MERGESORT(A,mid+1,r)
        pair += MERGE(A,l,mid,r)
    end if
    return pair
end procedure

input: List A is comprised of two sorted lists
output: number of pairs disorder
procedure Merge(A,l,mid,r)
    pair=0, i=l, j=mid+1,k=l,output=list for output with length of A
    while i≤ mid & j≤ r do
        if A[i]≤ A[j] then
            output[k]=A[i],i+=1,k+=1
        else
            output[k] =A[j],pair+=(mid-i+1),k+=1,j+=1
                                          ▷ A[j] is less than every number from index i to mid
        end if
    end while
    while i≤mid do copy remaining of left to the end of output list
    end while
    while j≤r do copy remaining of right to the end of output list
    end while
    return pair
end procedure
```

---

3. (25 points) In the table below, indicate the relationship between functions f and g for each pair (f, g) by writing "yes" or "no" in each box. For example, if $f = O(g)$ then write "yes" in the first box. Here $\log^b x = (\log_2 x)^b$.

f	g	O	o	Ω	ω	Θ
$6n \log^2 n$	$n^2 \log n$	yes	yes	no	no	no
$\sqrt{\log n}$	$(\log \log n)^3$	yes	yes	no	no	no
$10n \log n$	$n \log(10n^2)$	yes	no	yes	yes	yes
$n^{4/5}$	$\sqrt{n} \log n$	no	no	yes	yes	no
$5\sqrt{n} + \log^2 n$	$3\sqrt{n}$	yes	no	yes	no	yes
$\frac{3^n}{n^3}$	$n^3 2^n$	no	no	yes	yes	no
$\sqrt{n}2^n$	$2^{n/2+\log n}$	no	no	yes	yes	no
$n \log(2n)$	$\frac{n^2}{\log n}$	yes	yes	no	no	no
$n!$	n^n	yes	yes	no	no	no
$\log n!$	$\log(n^{n/2})$	yes	no	yes	no	yes

4. (a) **Proof** :

   i. Base case:
   $F_6 = 8 \geq 2^3$
   $F_7 = F_6 + F_5 = 13 \geq 2^{7/2}$

   ii. Inductive hypothesis:
   suppose for all $n \geq 6, F_n \geq 2^{n/2}$holds.

   iii. inductive step:
   For $n > 6$, the following holds
   $F_{n+1} = F_n + F_{n-1} \geq 2^{n/2} + 2^{(n-1)/2} = 2^{n/2} * (1 + \frac{1}{\sqrt{2}}) > 2^{n/2}\sqrt{2} = 2^{(n+1)/2}$
   Thus, proved.

   (b) i) Give an algorithm that computes Fn based on the recursive definition above. Develop a recurrence for the running time of your algorithm and give an asymptotic lower bound for it.
   Run time:

   T(n)= T(n-1)+T(n-2)+c
   = 2T(n-2)+T(n-3)+2c

= 3T(n-3)+2T(n-4)+4c

$\geq 2T(n-3) + 2T(n-4) + 4c$

$= 4T(n-4) + 2T(n-5) + 6c$

...

$\geq 2^k T(n-2k) + (2^{k+1} - 2)c$

let $n - 2k = 0$, thus T(n)=$\Omega(2^{n/2})$

Correctness:

Base: if n=0 n=1, Recurfib(n)=F_n=n is correct

Hypothesis: for $n > 1$, $Recurfib(n)$ correctly computes F_n

Inductive Step: By definition of the algorithm

$Recurfib(n+1) = Recurfib(n) + Recurfib(n-1)$

$= F_n + F_{n-1} = F_{n+1}$

Thus the algorithm is correct.

Algorithm 4 Algorithm Fib Recursion

Input An integer n

Output n^{th} Fibonacci number

```
procedure Recurfib(n)
    if n≤ 1 then return n
    else
        return Recurfib(n-1)+Recurfib(n-2)
    end if
end procedure
```

ii) 8 points) Give a non-recursive algorithm that asymptotically performs fewer additions than the recursive algorithm. Discuss the running time of the new algorithm.

Description: First set the base cases, to find the nth fib number, iterate n-2 times: next fib is sum of the last two which a+b, move a=b, b=c to start the next iteration until done.

Run time: since there is only one for loop: O(n)

Correctness:

Base: fib(n) is correct for n=0 and n=1

Hypothesis: fib(n) return Fibonacci number correctly for $n > 1$

Inductive step:

Since fib(n) and fib(n-1) return correct F_n and F_{n-1}

We will show fib(n+1) returns F_{n+1}

from the algorithm, at the end of n-3 iteration, b=fib(n-1)=F_{n-1};at the end of $n-2$ iteration, a=fib(n-1),b=fib(n)=F_n; at the end of $n-1$ iteration c=fib(n)+fib(n-1)=$F_n + F_{n-1} = F_{n+1}, return\ b = c = F_{n+1}$ correctly

---

Algorithm 5 Algorithm Fib Non-Recursion

Input An integer n
Output n^{th} Fibonacci number

```
procedure fib(n)
    a=0,b=1
    if n≤ 1 then return n
    else
        for i∈[1,n-2] do
            c=a+b,a=b,b=c
        end for
    end if
    return b
end procedure
```

---

iii) Description:

To get the nth fib number, we do n-1 power of matrix M and multiply by matrix [1 0] and get the first element in the matrix. Improve (n-1) power by using (n-1)//2 power of matrix multiply by itself; If n is even, we get (n-1) power we need; If n is odd, multiply by M and get n-1 power we need.

Run time of Power(M,n): T(n)=T(n/2)+c=O(logn)

Run time of fib(n): T(n) = O(Logn)+O(1)=O(logn)

Correctness of Power(M,n):

We know Multiply(A,B) is correct because it is just multiplication and addition. We need to prove the algorithm returns correct Matrix power of n

Base: n=1, Power(M,n) is correct

Hypo: assume for any $n > 1$,Power(M,n) gives correct matrix of n power to M.

Inductive step:

if n is even, since Power(M,n/2) is correct, then multiply by itself gives correct n power.

if n is odd, since Power(M,n//2) is correct, then multiply by itself and K gives correct power 2*(n//2)+1 = n. proved.

Correctness of fib(n):

We know Power(M,n) is correct.

Base: fib(0) and fib(1)=$M^0 * F_{base}[0][0]$ is correct;

Inductive hypothesis: assume fib(n) correctly return F_n

Inductive Step:

For n+1, $Power(M, n) * F_{base} = M * Power(M, n-1) * F_{10} = M * [F_n \ F_{n-1}]^T = [F_n + F_{n-1} \ F_n]^T$ which will return the first element of the matrix: $F_n + F_{n-1} = F_{n+1}$, proved.

Algorithm is located the next page

Algorithm 6 Algorithm Fib Matrix

```
    Input An integer n
    Output n^th Fibonacci number
procedure fib(n)
    M= [1 1
        1 0]
    F_base=[1 0]
    if n= 0 then return 0
    else
        return Power(M,n-1)*F_base[0][0]          ▷ Return the first element of the matrix
    end if
end procedure

procedure Power(M,n)
    K= [1 1
        1 0]
    if n ≤ 1 then return M
    else
        M=Power(M,n//2),M=multiply(M,M)
        if n is odd then
            M=Multiply(K,M)
        end if
    end if
end procedure

procedure Multiply(A,B)
    x = (A[0][0] * B[0][0] + A[0][1] * B[1][0])
    y = (A[0][0] * B[0][1] + A[0][1] * B[1][1])
    z = (A[1][0] * B[0][0] + A[1][1] * B[1][0])
    w = (A[1][0] * B[0][1] + A[1][1] * B[1][1])
    A[0][0] = x
    A[0][1] = y
    A[1][0] = z
    A[1][1] = w
    return A
end procedure
```

iv) This algorithm is the same as the last part except the matrix M and the F_base are different. Run time O(logn) and correctness can both be proved similarly to the previous question.

Algorithm 7 Algorithm Fib Plus Number

Input An integer n

Output n^{th} Fibonacci Plus number

procedure p(n)

M= $\begin{bmatrix} 1 & 3 & 2 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \end{bmatrix}$

F_{base}=[p_1 p_0 2 1]

if n= 0 **then return** a

else

return Power(M,n-1)*F_{base}^T[0][0]

end if

end procedure

procedure Power(M,n)

K= $\begin{bmatrix} 1 & 3 & 2 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \end{bmatrix}$

if $n \leq 1$ **then** return M

else

M=Power(M,n//2),M=Multiply(M,M)

if n is odd **then**

M=Multiply(K,M)

end if

end if

end procedure

procedure Multiply(A,B)

A[i][j]=$\sum_{k=1}^{4}(A[i][k] * B[k][j])$ **return**A

5. (a) Run time is O(1)+O(n-1)*O(n-2)*O(1)=O(n^2) since there are two loops

   Description and Correctness: Initialize min to be very large, then we iteratively calculate the euclidean distance of p_i and all p_j for j∈[i+1,n], for every i∈[1,n-1] and obtain the min by comparing with the previous min, the pair with smallest distance is the closest.

Algorithm 8 BruteForce Closest Pair

Input a set of n points,(n is a power of 2) in the plane P={$p1 = (x1, y1), p2 = (x2, y2), ..., pn = (xn, yn)$}.

Output the pair (pi,pj) with pi≠pj for which the euclidean distance between pi and pj is minimized.

```
procedure BF(P)
    min=∞
    for i in [1,n-1] do
        for j in [i+1,n] do
            if euclidean distance of p[i] and p[j]<min then
                min=the euclidean distance, pair=(p[i],p[j])
            end if
        end for
    end for
    return pair
end procedure
```

(b) i. Let T(n) be the run time of the algorithm.

1. Find a value x for which exactly half the points have $xi < x$ and half have $xi > x$.On this basis, split the points in two groups,L and R. We need to sort by x coordinates first, which is O(nlogn) using mergesort

2. Recursively find the closest pair, which takes 2*T(n/2)

3. Discard all points with $xi < xd$ or $xi > x + d$ and sort the remaining points by y coordinate. Finding the points to discard and Mergesort takes O(n)+O(nlogn)

4. Now go through the sorted list and for each point compute its distance to the seven subsequent points in the list. O(n)*7=O(n)

The answer is whichever has the smallest euclidean distance.

Thus, Recurrence run time= T(n)=2*T(n/2)+cnlogn+cn=2T(n/2)+cnlogn. Solving this in the next part.

Algorithm 9 Divide and Conquer Cloest Pair

Input two set of n points $\{xp\}, \{yp\}$.

Output the distance and the pair (pi,pj) with pi≠pj for which the euclidean distance between pi and pj is minimized.

```
procedure Cloest(xp, yp)
    if less than 3 points then use brute force
    else
        xp=Mergesort(xp)
        xm=xp[ n//2]
        xl=poins that has x coordinate less then xm,yl is corresponding y coordinate
        xr=poins that has x coordinate greater then xm,yr is corresponding y coordinate
        dl,(pl,ql) = Cloest(xl,yl)
        dr,(pr,qr) = Cloest(xr,yr)
    end if
    dmin=min(dl,dr), minpair is pair with min distance
    remove(x,y) such that |xm − x| > dmin
    p_sorted=Mergesort the remaining points by y coordinate
    for i in length of p_sorted do
        k=i+1
        while k≤ len(7) do
            if euclidean distance (p_sorted[i],p_sorted[k]) <dmin then
                dmin= euclidan distance of p_sorted[i] and p_sorted[k]
                pair=(p_sorted[i],p_sorted[k])
            end if
            k+=1
        end while
    end for
    return pair, dmin
end procedure
```

---

ii. Let T(n) be the run time of the algorithm.

1. Find a value x for which exactly half the points have $xi < x$ and half have

$xi > x$.On this basis, split the points in two groups,L and R. We need to sort by x coordinates first, which is O(nlogn) using mergesort
2. Recursively find the closest pair, which takes 2*T(n/2)
3. Discard all points with $xi < xd$ or $xi > x + d$ and sort the remaining points by y coordinate. Finding the points to discard and Mergesort takes O(n)+O(nlogn)
4. Now go through the sorted list and for each point compute its distance to the seven subsequent points in the list. O(n)*7=O(n)
The answer is whichever has the smallest euclidean distance.
Thus, Recurrence run time= T(n)=2*T(n/2)+cnlogn+cn=2T(n/2)+cnlogn. Solving this in the next part.

T(n)=2*T(n/2)+cnlogn+cn=2T(n/2)+cnlogn
=2(2T(n/4)+(n/2)log(n/2))+nlogn
=4T(n/4)+nlog(n/2)+nlogn
...
=$2^k T(n/2^k) + n(logn + ... + log(n/2^k))$
let $n = 2^k$, then k=logn,T(1)=1 thus:

$$T(n) = 2^k T(n/2^k)+n(logn+...+log(n/2^k)) < n+nklog(n) = n+nlog^2(n) = O(nlog^2(n))$$