

HW1

- For each function $f(n)$ and time t in the following table, determine the largest size n of a problem that can be solved in time t , assuming that the algorithm to solve the problem takes $f(n)$ microseconds.

	1 seconds	1 minute	1 hour	1 day	1 month	1 year	1 century
$\log\log n$	$2^{2^{1000000}}$	$2^{2^{(6 \times 10^7)}}$	$2^{2^{(3.6 \times 10^9)}}$	$2^{2^{(8.64 \times 10^{10})}}$	$2^{2^{(2.6 \times 10^{12})}}$	$2^{2^{(3.15 \times 10^{13})}}$	$2^{2^{(3.15 \times 10^{15})}}$
$2\log n$	2^{500000}	$2^{(3 \times 10^7)}$	$2^{(1.8 \times 10^9)}$	$2^{(4.32 \times 10^{10})}$	$2^{(1.3 \times 10^{12})}$	$2^{(1.58 \times 10^{13})}$	$2^{(1.58 \times 10^{15})}$
N	1000000	6×10^7	3.6×10^9	8.64×10^{10}	2.6×10^{12}	3.15×10^{13}	3.15×10^{15}
$n^2 \log n$	344	2316	16052	73124	374175	1247898	11592507
$n^{5/2}$	251	1291	6645	23692	92353	250912	1583152
3^n	12	16	20	22	26	28	32

Solution:

1 second=1000000 microseconds; 1 minute=60 s; 1 hour=60 minutes; 1 day=24 hours; 1 month=30 days; 1 year=365 days; 1 century=100 years

A: $\log\log n=1000000 \rightarrow \log(\log n)=1000000 \rightarrow \log n=2^{1000000} \rightarrow n=2^{2^{1000000}}$

B: $2 \log n=1000000 \rightarrow \log n=500000 \rightarrow n=2^{500000}$

C: $n=1000000$;

others: write a program to solve these question: e.g for $n^2 \log n=1000000$, if $n=344$, $n^2 \log n < 1000000$; if $n=345$, $n^2 \log n > 1000000$; so the max number is 344. This is my Python program below.

```
import math
a=[]
a.append(1000000)    # 1 second
a.append(a[0]*60)    # 1 minutes
a.append(a[1]*60)    # 1 hour
a.append(a[2]*24)    # 1 day
a.append(a[3]*30)    # 1 month
a.append(a[3]*365)   # 1 year
a.append(a[5]*100)   # 1 century
print(a) #these values is the answer for n
i=0
for i in range(0,7):
    for n in range(1,100000000):
        # you can choose different formula to get other answer
        if n*n*math.log(n,2) > a[i]:
            #if math.sqrt(math.pow(n,5)) > a[i]:
            #if math.pow(3,n) > a[i]:
                print(i+1,n-1)
                break
```

2. Illustrate the operation of the Insertion-Sort algorithm from book on the array $A=\langle 5,4,3,2,1 \rangle$ and $A=\langle 42,13,54,19,21 \rangle$.

The red number indicates $A[j]$, the green number indicates that the number need to shift 1 position to right.

Solution:

- a) $A=\langle 5,4,3,2,1 \rangle$

5	4	3	2	1
---	---	---	---	---

At first, $j=2$, so $\text{key}=a[j]=4$. compare with $a[i=j-1] \rightarrow a[1]$, which is 5. Since $\text{key}<a[1]$, so shift $a[1]$ to $a[2]$ and replace $a[1]$ with key.

4	5	3	2	1
---	---	---	---	---

$j=3$, $\text{key}=a[3]=3$. Key compare with $a[2]=5$. Since $\text{key}<a[2]$, shift $a[2]$ to $a[3]$. Then key compare $a[1]=4$. Since $\text{key}<a[1]$, shift $a[1]$ to $a[2]$. Finally, replace $a[1]$ with key.

3	4	5	2	1
---	---	---	---	---

$j=4$, $\text{key}=a[4]=2$. Key compare with $a[3]=5$. Since $\text{key}<a[3]$, shift $a[3]$ to $a[4]$. Then key compare with $a[2]=4$. Since $\text{key}<a[2]$, shift $a[2]$ to $a[3]$. Then key compare $a[1]=3$. Since $\text{key}<a[1]$, shift $a[1]$ to $a[2]$. Finally, replace $a[1]$ with key.

2	3	4	5	1
---	---	---	---	---

$j=5$, $\text{key}=a[5]=1$. Key compare with $a[4]=5$. Since $\text{key}<a[4]$, shift $a[4]$ to $a[5]$. Then key compare with $a[3]=4$. Since $\text{key}<a[3]$, shift $a[3]$ to $a[4]$. Then key compare with $a[2]=3$. Since $\text{key}<a[2]$, shift $a[2]$ to $a[3]$. Then key compare $a[1]=2$. Since $\text{key}<a[1]$, shift $a[1]$ to $a[2]$. Finally, replace $a[1]$ with key.

1	2	3	4	5
---	---	---	---	---

$j>a.\text{length}$, the program end.

- b) $A=\langle 42,13,54,19,21 \rangle$

42	13	54	19	21
----	----	----	----	----

At first, $j=2$, so $\text{key}=13$, compare with $a[i=j-1] \rightarrow a[1]$, which is 42. Since $\text{key}<a[1]$, so shift $a[1]$ to $a[2]$ and replace $a[1]$ with key.

13	42	54	19	21
----	----	----	----	----

$j=3$, $\text{key}=a[3]=54$. Key compare with $a[2]=42$. Since $\text{key}>a[2]$, exit while loop. Replace $a[2+1]$ with key.

13	42	54	19	21
----	----	----	----	----

$j=4$, $\text{key}=a[4]=19$. Key compare with $a[3]=54$. Since $\text{key}<a[3]$, shift $a[3]$ to $a[4]$. Then key compare with $a[2]=42$. Since $\text{key}<a[2]$, shift $a[2]$ to $a[3]$. Then key compare $a[1]=13$. Since $\text{key}<a[1]$, exit while loop. Finally, replace $a[1+1]$ with key.

13	19	42	54	21
----	----	----	----	----

j=5, key=a[5]=21. Key compare with a[4]=54. Since key<a[4], shift a[4] to a[5]. Then key compare with a[3]=42. Since key<a[3], shift a[3] to a[4]. Then key compare with a[2]=19. Since key>a[2], exit while loop. Finally, replace a[2+1] with key.

13	19	21	42	54
----	----	----	----	----

j>a.length, the program end.

3. Use mathematical induction to show that when n is an exact power of 4, the solution of the recurrence relation

$$T(n) = \begin{cases} 8 & \text{if } n = 4 \\ 4T\left(\frac{n}{4}\right) + 2n & \text{if } n = 4^k, \text{ for } k > 1 \end{cases}$$

is $T(n) = n \log_2 n$.

Solution:

Base case: $k=1, n=4$

LHS: $T(4)=8$

RHS: $T(4)=4\log_2 4=4 \times 2=8$

So, LHS=RHS, so the base case is true.

Inductive case:

Assume the statement hold up to k ,

So, $n=4^k$, $T(n)=4T\left(\frac{n}{4}\right) + 2n = n \log_2 n$, so LHS(n)=RHS(n)

And try to prove it is true for $k+1$:

when $4^{k+1}=4 \times 4^k=4n$,

so, we need to prove : LHS(4n)=RHS(4n)

LHS(4n)= $4T\left(\frac{4n}{4}\right) + 2(4n)$

= $4T(n) + 8n$, since $T(n)=n \log_2 n$

= $4n \log_2 n + 8n$

RHS(4n)= $4n \log_2 4n$

= $4n(\log_2 4 + \log_2 n)$

= $4n(2 + \log_2 n)$

= $4n \log_2 n + 8n$

So, LHS(4n)=RHS(4n), the statement hold for $k+1$.

Conclusion:

Since the statement hold in base case and inductive case, the statement is true.

4. Problem 2.3 out of the book (make sure this is in your own words and you don't look on the internet!).

Correctness of Horner's rule:

```
Y=0
For i= n downto 0
    Y=ai+x.y
```

Solution:

- a) In terms of O-notation, what is the running time of this code fragment for Horner's rule?
From the codes, we can see there is a for loop, which i form n to 0, so the running time should be O(n).
- b) Write pseudocode to implement the naïve polynomial-evaluation algorithm that computes each term of the polynomial from scratch. What is the running time of this algorithm? How does it compare to Horner's rule?

```
y=0
for i=0 to n
    z=1 // X0=1
    for j=1 to i //z=Xi
        z*=x
    y+=a[i]*z
```

The running time of this codes is O(n²) , because it has two for loop. It is much slow than Horner's rule.

- c) Consider the following loop invariant:

$$y = \sum_{k=0}^{n-(i+1)} a_{k+i+1} x^k$$

Initialization:

Before the for loop: y=0.

At the beginning of the loop, i=n. From the formula, we get

$y = \sum_{k=0}^{-1} a_{k+n+1} x^k$, we also get y=0.

It shows that the loop invariant holds prior to the first iteration of the loop.

Maintenance:

Assume the loop invariant hold at i=j:

$$y = \sum_{k=0}^{n-(j+1)} a_{k+j+1} x^k$$

Then, at i=j-1

$$\begin{aligned} y(j-1) &= a_j + x \sum_{k=0}^{n-(j+1)} a_{k+j+1} x^k \\ &= a_j + \sum_{k=0}^{n-(j+1)} a_{k+j+1} x^{k+1} \\ &= a_j x^0 + a_{j+1} x^1 + a_{j+2} x^2 + \dots + a_n x^{n-j} \\ &= \sum_{k=0}^{n-(j)} a_{k+j} x^k \end{aligned}$$

So, the loop invariant also holds in induction case.

Termination:

When program exits the for loop, $i=-1$, then

$$\begin{aligned} y &= \sum_{k=0}^{n-(i+1)} a_{k+i+1} x^k \\ &= \sum_{k=0}^{n-(-1+1)} a_{k-1+1} x^k \\ &= \sum_{k=0}^n a_k x^k \end{aligned}$$

So, the loop invariant also holds.

- d) Conclude by arguing that the given code fragment correctly evaluates a polynomial characterized by coefficients a_0, a_1, \dots, a_n .

According to the c) part, we have proved that from the beginning to the end of the program, the codes are compute the value of y is the same as the polynomial function. So, it correctly evaluates a polynomial characterized by coefficients a_0, a_1, \dots, a_n .