1. (25 pts)

Express Insertion Sort as a Recursive Procedure.

(Hint: In order to Sort A[1....n], we recursively sort A[1...n-1] and insert A(n) into the sorted array A[1.... n-1].

```
Recursive-Insertion-Sort(A, M)
If M < 2
        error "Please start from right position!"
If M > A.length
        print "Sorting procedure finished!"
key = A[M]
k = M - 1
while k > 0 and A[k] > key
        A[k+1] = A[k]
        k = k - 1
A[k+1] = key
Recursive-Insertion-Sort(A, M+1)
```

2. (30 pts)

(a) Use iteration method or recursion tree method to determine the total running time $T(n)$ of the recurrence (formula): $T(n) = 3T(n/4) + n$

$T(\frac{n}{4}) = 3T(\frac{n}{4^2}) + n/4$

$T(\frac{n}{4^2}) = 3T(\frac{n}{4^3}) + n/4^2$

$T(\frac{n}{4^3}) = 3T(\frac{n}{4^4}) + n/4^3$

$T(\frac{n}{4^4}) = 3T(\frac{n}{4^5}) + n/4^4$

$$T(n) = 3\left(3T(\tfrac{n}{4^2}) + n/4\right)$$

$$= 3^2 T(\tfrac{n}{4^2}) + \tfrac{3}{4}n$$

$$= 3^2\left(3T(\tfrac{n}{4^3}) + \tfrac{n}{4^2}\right) + \tfrac{3}{4}n$$

$$= 3^3 T(\tfrac{n}{4^3}) + \tfrac{3^2}{4^2}n + \tfrac{3}{4}n$$

$$= 3^3\left(3T(\tfrac{n}{4^4}) + n/4^3\right) + (\tfrac{3}{4})^2 n + (\tfrac{3}{4})n$$

$$= 3^4 T(\tfrac{n}{4^4}) + (\tfrac{3}{4})^3 n + (\tfrac{3}{4})^2 n + (\tfrac{3}{4})n$$

$$\cdots$$

$$= 3^k T\left(\frac{n}{4^k}\right) + \left(\frac{3}{4}\right)^{k-1} n + \left(\frac{3}{4}\right)^{k-2} n + \cdots \left(\frac{3}{4}\right)^2 n + \left(\frac{3}{4}\right)n$$

$$= 3^k T\left(\frac{n}{4^k}\right) + n\underbrace{\left(\left(\frac{3}{4}\right)^{k-1} + \left(\frac{3}{4}\right)^{k-2} + \cdots + \left(\frac{3}{4}\right)^2 + \left(\frac{3}{4}\right)\right)}_{k-1 = \frac{1}{2}\lg n - 1 < \lg n}$$

$$\frac{n}{4^k} = 1 \Rightarrow 4^k = n$$

$$k = \log_4 n$$

$$k = \log_{2^2} n$$

$$k = \frac{1}{2}\log_2 n$$

Thus, $T(n) = 3^{\frac{1}{2}\lg n} T(1) + n\left(\left(\frac{3}{4}\right)^{k-1} + \left(\frac{3}{4}\right)^{k-2} + \cdots + \left(\frac{3}{4}\right)^2 + \left(\frac{3}{4}\right)\right)$

...

$$\leq 3^{\frac{1}{2}\lg n} T(1) + n\left(\tfrac{1}{2}\lg n - 1\right)\tfrac{3}{4}$$

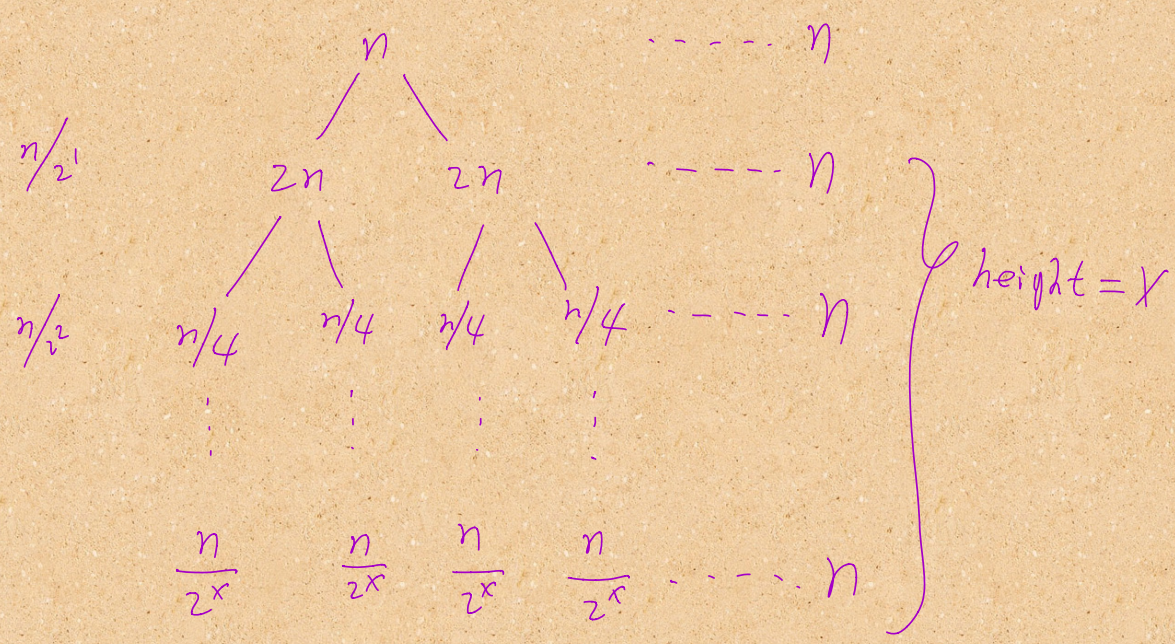
$$= C_1 T(1) + \tfrac{3}{8} n \lg n - \tfrac{3}{4} n$$

$$= C_1 T(1) + C_2 n \lg n - C_3$$

$$\longrightarrow O(n \lg n)$$

(b) Use iteration method or recursion tree method to determine the total running time of the formula:

$T(n) = 2T(n/2) + n$



$\frac{n}{2^x} = 1 \Rightarrow 2^x = n$ each level consume n

$\Rightarrow x = \lg n$

Total running time: $n \lg n$

$T(n) = O(n \lg n)$

3. (15 pts)

(a) Write the definition of $o(g(n))$

$o(g(n))$ is the asymptotic notation used to describe the growth of functions.

for a given function $g(n)$, we use below symbols:

1) $\Theta(g(n)) \Rightarrow$ we say $g(n)$ is an asymptotically tight bound for $f(n)$.
(Θ-notation)
when $f(n)$ is a member of $\Theta(g(n))$, we write $f(n)=\Theta(g(n))$

2) O-notation $\Rightarrow$ asymptotic upper bound

3) Ω-notation $\Rightarrow$ asymptotic lower bound

4) o-notation $\Rightarrow$ upper bound that is not asymptotically tight

5) ω-notation $\Rightarrow$ lower bound that is not asymptotically tight

those are different notations.

$o(g(n)) \Rightarrow$ refers to upper bound that is not asymptotic tight

we formally define $o(g(n))$ "little-oh of get n" as the set

$o(g(n)) = \{ f(n)$: for any positive constant $c>0$, there exists a constant $n_0>0$ such that $0 \le f(n) < c\,g(n)$ for all $n \geq n_0 \}$

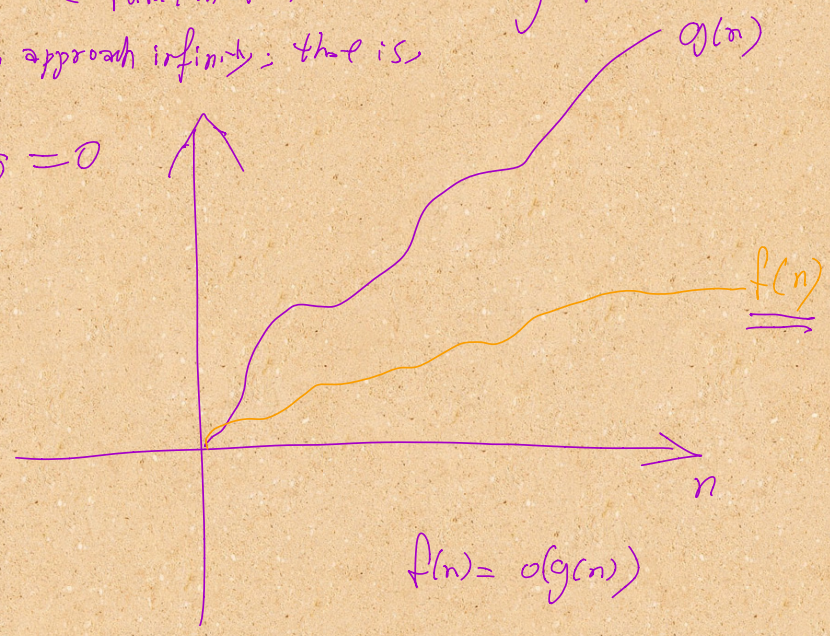
(b) if $f(n)=o(g(n))$, prove and conclude the limit of $f(n)/g(n)$ when n approach to infinity.

for $O(g(n))$, the bound $0 \le f(n) \le c\,g(n)$ holds for some constant $c>0$;

However, for $o(g(n))$, the bound $0 \le f(n) < c\,g(n)$ holds for all constant $c>0$.

in o-notation, the function $f(n)$ become insignificant relative to $g(n)$ as n approach infinity; that is,

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$$



as I included in the chart, $g(n)$ grow much much faster than $f(n)$, so when n is infinite large, $f(n)$ is infinitely smaller than $g(n)$.

4.

(a) write the procedure of Build-Max-Heap(A)

```
Build-Max-Heap(A)
A.heap-size = A.length
for i = ⌊A.length/2⌋ downto 1
    Max-Heapify(A, i)
```

```
Max-Heapify(A, i)
l = Left(i)
r = right(i)
if l ≤ A.heap-size and A[l] > A[i]
    largest = l
else largest = i
if r ≤ A.heap-size and A[r] > A[largest]
    largest = r
if largest ≠ i
    exchange A[i] with A[largest]
    Max-Heapify(A, largest)
```

b) Draw a diagram to do operation of

Build-Max-Heap(A) on A = [5, 3, 17, 10, 84, 19

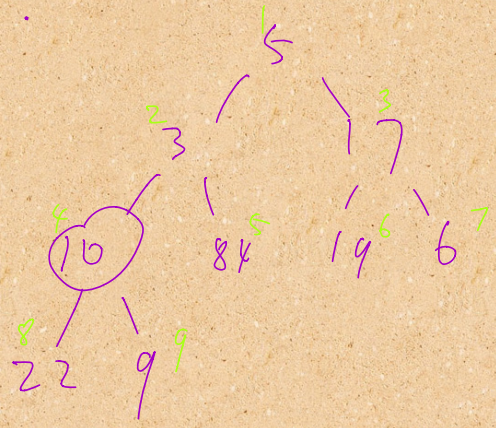
6, 22, 9]

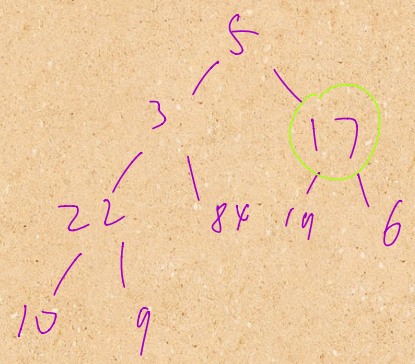
start

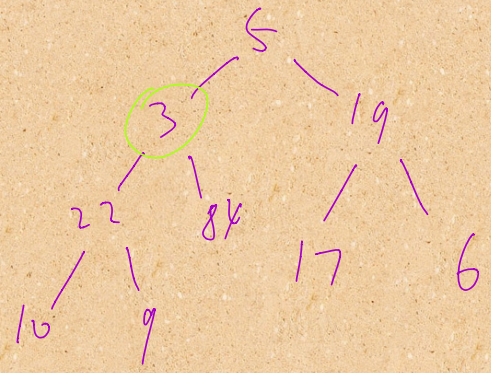
A = 5, 3, 17, 10, 84, 19, 6, 22, 9

1 2 3 4 5 6 7 8 9

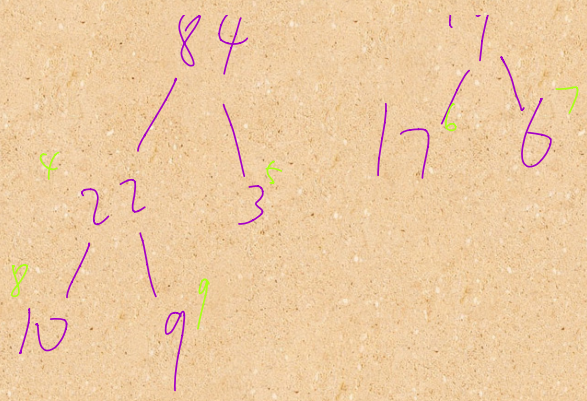
$9/2 = 4,$

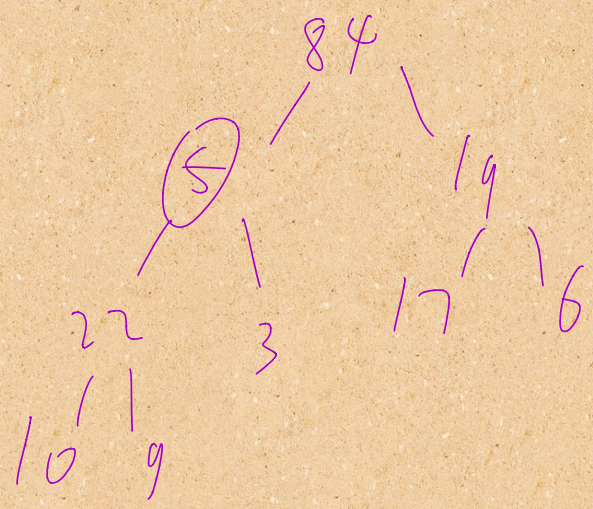


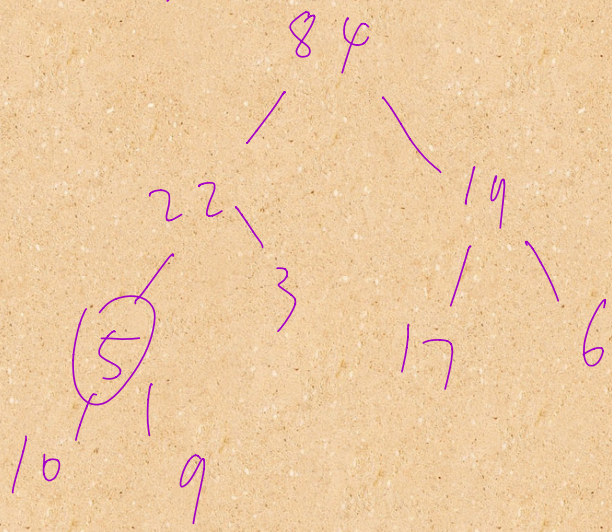




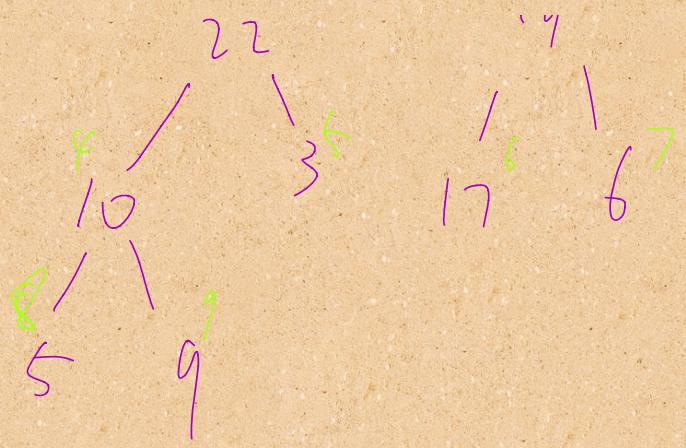
9


84
22
3
10
9
17
6


84
5
19
22
3
17
6
10
9


84
22
19
3
5
17
6
10
9

84
16



5.

(a)

```
QuickSort(A, p, r)
if p < r
    q = partition(A, p, r)
    QuickSort(A, p, q-1)
    QuickSort(A, q+1, r)
```

```
partition(A, p, r)
x = A[r]
i = p-1
for j = p to r-1
    if A[j] ≤ x
```

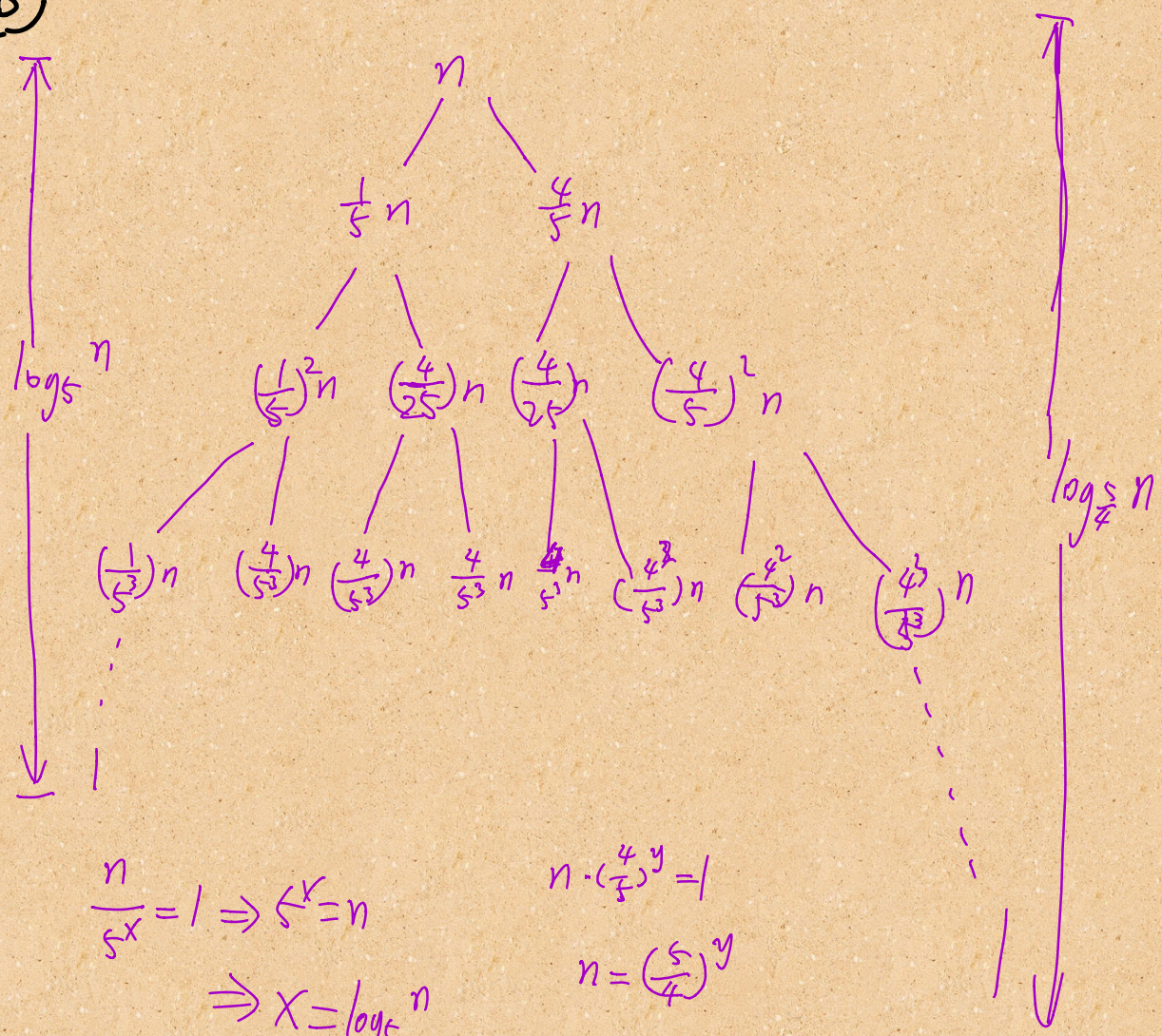
5. $j = i+1$

6. exchange $A[i]$ with $A[j]$

7. exchange $A[i+1]$ with $A[r]$

8. return $i+1$

(b)



$$\frac{n}{5^x} = 1 \Rightarrow 5^x = n$$

$$\Rightarrow x = \log_5 n$$

$$n \cdot \left(\frac{4}{5}\right)^y = 1$$

$$n = \left(\frac{5}{4}\right)^y$$

$$\Rightarrow y = \log_{\frac{5}{4}} n$$

$$= \frac{1}{5} \log_4 n$$

$$= \frac{1}{5} \log_2 n$$

$$= \boxed{\frac{1}{10} \lg n}$$

$$T(n) = T\left(\frac{4}{5}n\right) + T\left(\frac{1}{5}n\right) + cn$$

The per level costs include the constant c implicit the $\Theta(n)$ term

The deepest length is $\frac{1}{10} \lg n$

each level consumption is $< n$

$$T(n) \le n \cdot \frac{1}{10} \lg n \le n \lg n$$

$$T(n) = O(n \lg n)$$