CS 5329

chapter 2 - all
chapter 3 - 3.1.
chapter 4 - skip 4.2 pg 86 - 87
skip 4.5, 4.6.
chapter 5 - skip
chapter 1 - skip but recommend to read.
chapter 6 - All
chapter 7 - skip pg - 177 to 178
182 - 185.

chapter 2 - insertion sort → Best case, computation times, worst case.
Merge sort → Analysis, Recurrence, iteration Me.

chapter 3:
Maximum subarray program.
(Submission method for solving recurrence) (Iteration Method)
Recursion tree method for solving recurrence.

chapter 6: heap sort.
build heap sort.
Priority queue.

chapter 7: Quick sort.
Randomized version of quick sort.

Fall 14

1> (30pts)

a> Given the following Merge sort function, write Merge function in detail pseudocodes.

b> Indicates the cost of computation complexity line by line or block by block in both function in big O or in Theta Θ Notation

c> Derive the recurrence formula T(n) for merge sort by using recursion tree and log algebra.

d> And use iteration Method & log algebra to compute the total running time T(n) for merge sort.

c>

```
MERGE.SORT(A.P.r)          (Ans a&b)
if p<r
    q = [(p+r)/2]
  MERGE.SORT(A.P.q)
  MERGE.SORT(A.q+1,r)
  MERGE(A.P,q.r)
```

$\overset{\circ}{\circ}$ —

a>

```
MERGE(A,P,q,r)                                  (b)
n1 = q-p+1;
n2 = q-r;                                       Θ(1)
Let L[1...n1+1] & R[1...n2+1] be the new array
for i<-1 to n1
        do L[i] = A[P+i-1]                     Θ(n1+n2)
for j<-1 to n2                                  = Θ(n)
        do R[j] = A[q+j]
```

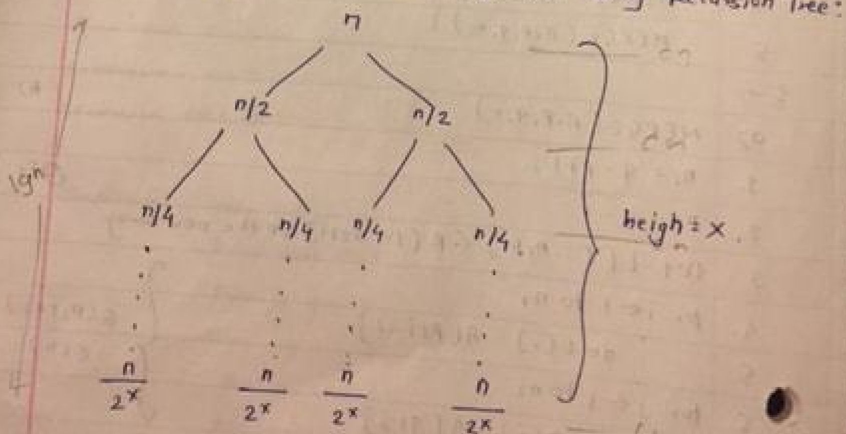
```
L[n1+1] = ∞
R[n2+1] = ∞
i ← 1
j ← 1
for k ← p to r
do if L[i] ≤ R[j]
      A[k] ← L[i]
      i ← i+1
   else
      A[k] ← R[j]
      j ← j+1
```

$\Theta(1)$

$\Theta(n)$

$T(n) = \Theta(n)$.

c) Recurrence Formula $T(n)$ for Merge sort using Recursion Tree:



$$\frac{n}{2^x} \cong 1 \Rightarrow n \cong 2^x$$

Taking log on both sides

$$\log n = \log 2^x$$

$$\log n = x \log 2$$

$$x = \frac{\log n}{\log 2}$$

$$x = \log_2 n$$

Total running time $T(n) = cn(x)$

$$= cn(\log_2 n)$$

$$= c(n \log_2 n)$$

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

Hence, Recurrence equation $\boxed{T(n) = 2T(n/2) + n}$

d) Iteration Method :

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

$n \log n$

$\boxed{\text{put } T(n/2) = 2T(n/4) + n/2}$

$$T(n) = 2\left(2T(n/4) + \frac{n}{2}\right) + \frac{n}{1}$$

$= 4T(n/4) + n + n)$

$= 4T(n/4) + 2n \cancel{+ n/2}$

$= 4\left(2T\left(\frac{n}{8}\right) + \frac{n}{4}\right) + 2n \cancel{+ n/2}$

$= 8T\left(\frac{n}{8}\right) + n + 2n \cancel{+ n/2}$

$= 8T\left(\frac{n}{8}\right) + 3n \cancel{+ n/2}$

$= 8T\left(\frac{n}{2^i}\right) + 3n$

$\therefore \frac{n}{2^i} = 1$

$n = 2^i$

$i = \log_2 n$

Total running time $= cn(i)$

$= cn(\log_2 n)$

$= c\,n \log n$

$T(n) = \Theta(n \log n)$

2> (35 or 25 pts)

a) Write insertion sort algorithm in detail psecdocodes.

b) Indicate the cost of computation time on all state and derive the total running time for best case.

c) Derive the total computation time for worst case.

d) Write insertion sort program.

a:

a> Psesudo Code :

Insertion_Sort(A)	Cost	Time
for $j \leftarrow 2$ to length[A]	c_1	n
do key $\leftarrow A[j]$	c_2	$n-1$
// insert $A[j]$ into the sorted sequence $A[1..j-1]$	0	$n-1$
$i \leftarrow j-1$	c_4	$n-1$
while $i>0$ and $A[i] >$ key	c_5	$\sum_{j=2}^{n} t_j$
do $A[i+1] \leftarrow A[i]$	c_6	$\sum_{j=2}^{n} (t_j-1)$
$i \leftarrow i-1$	c_7	$\sum_{j=2}^{n} (t_j-1)$
$A[i+1] \leftarrow$ key	c_8	$n-1$

b> Total running time for best case : when $t_j = 1$

$\underline{an+b}$

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^{n} t_j + c_6 \sum_{j=2}^{n} (t_j-1)$$

$$+ c_7 \sum_{j=2}^{n} (t_j-1) + c_8(n-1)$$

$$= \underbrace{(c_1+c_2+c_4+c_5+c_8)}_{a}\, n - \underbrace{(c_2+c_4+c_5+c_8)}_{b}$$

∴ Time Complexity = $\Theta(n)$ for best case.

$an^2+bn+c.$

c) Total running time for worst case $(t_j = j)$

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^{n} t_j + c_6 \sum_{j=2}^{n} t_j - 1$$

$$+ c_7 \sum_{j=2}^{n} t_j - 1 + c_8 (n-1)$$

$$= c_1 n + c_2(n-1) + c_4(n-1) + c_5\left(\frac{n(n+1)}{2} - 1\right) + c_6\left(\frac{n(n-1)}{2}\right)$$

$$+ c_7\left(\frac{n(n-1)}{2}\right) + c_8(n-1)$$

$$= \underbrace{\left(\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2}\right)}_{a} n^2 + \overbrace{\left(c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8\right)}^{b} n$$

$$- (c_2 + c_4 + c_5 + c_8)$$

Time complexity = $\Theta(n^2)$ for worst case.

d) Program for insertion sort.

```
#include <iostream>
#include <cstdlib>
using namespace std;
```

m[4,6], s[4,6]

i = 4,
j = 6

k = (4,5)

m[4,6] =

min { (4,4) + (4,6)
      (4,5) + (6,6)

```
int main ()
{
   int data[100];
   int i,j, temp=0;
   int cnr=0, num=0;
  cout<<"enter size of array";
   cin>>num
   for( i=0; i<num; i++ )
     {
      data[i] = rand() % 500 + 1;
     }
 for( i=0; i<num; i++ )
     {
       cout<< data[i];
     }
for( i=0; i<num; i++ )
  {    cnr++;
     for ( j=i; j>-1; j-- )
       {    cnr++;
          if (data[j] > data[j+1] )
             {
                cnr++;
                temp = data[j];
                data[j] = data[j+1];
                data[j+1] = temp;
             }
  }   }
```

P4, P6

P5

P3 P4 P6 k

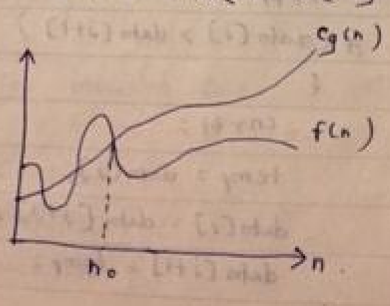
P3 P5 P6 k=5

```
for( i=0; i<num ; i++ )
   {
     cout<< data [i];
   }
cout<<"Execution time for best case"<< cnr ;
cout<<"Execution time for worst case"<< cnr*cnr ;
return 1;
}
```

3) a) Define mathematically the asymptotic upper bound big O notation : $f(n) = O(g(n))$ and draw a diagram to represent it.

: The Θ notation asymptotically bounds a function from above and below. When we have only a asymptotic upper bound, we use O notation. For given function $g(n)$ we denote by $O(g(n))$ (pronounced "big-oh of g of n"

$O(g(n)) = \{ f(n)$: there exist positive constants c & n_0 such that $0 \leq f(n) \leq c g(n)$ for all $n \geq n_0 \}$



The fig gives an intuition behind O-notation. For all values of n at and to the right of n_0, f(n) lies on or below cg(n).

b) Define mathematically the upper bound little-oh notation $f(n) = o(g(n))$ & indicate the limit of $f(n)/g(n)$ when n approaches to infinity.

∴ The asymptotic ~~no~~ upper bound provided by O-notation may or may not be asymptotically tight. The bound $2n^2 = O(n^2)$ is asymptotically tight, but the bound $2n = O(n^2)$ is not.

[No need to write]

We use o-notation to denote an upper bound that is not asymptotically tight.

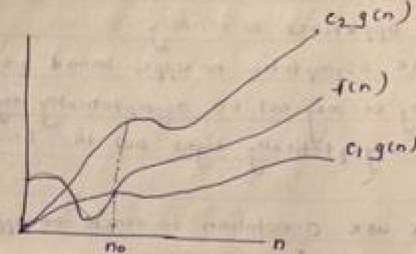
We define o(g(n)) ("little oh of g of n")

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

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

c) Define mathematically the asymptotic tight bound Theta notation : $f(n) = \Theta(g(n))$

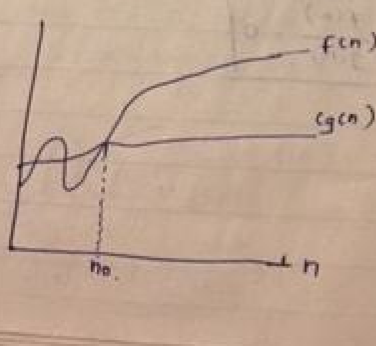
$\Theta(g(n)) = \{ f(n)$: there exist positive constants $c_1, c_2, \& n_0$ such that $0 \le c_1 g(n) \le f(n) \le c_2 g(n)$ for all $n \geq n_0 \}$



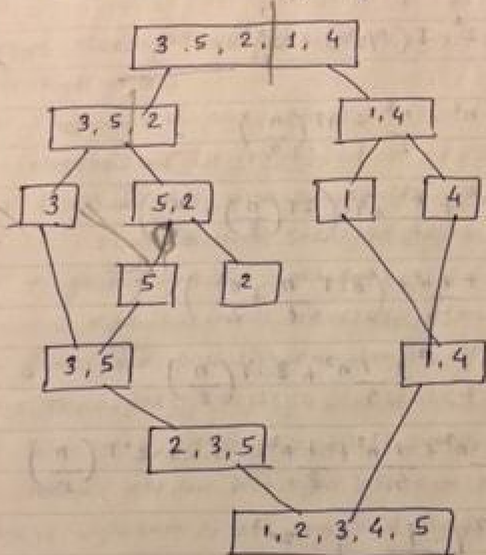
$f(n) = \Theta(g(n))$

d) $f(n) = \Omega(g(n))$

$\Omega(g(n)) = \{ f(n)$: there exist positive constants $c \& n_0$ such that $0 \le cg(n) \le f(n)$ for all $n \geq n_0 \}$



c) Show the steps in sorting array 3, 5, 2, 1, 4 using Merge sort.



d) Use iteration Method and log algebra to derive the total running time $T(n)$ for the recurrence.

$T(n) = 2T(n/2) + n * n$. (i.e. n^2) (Prove this) $\boxed{T(n) \le cn\lg n}$ $c > 0$.

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

$T(n/2) = 2T\left(\frac{n/2}{2}\right) + (n/2)^2$

So $T(n/2) = 2T(n/4) + (n/2)^2$

Total running time $\boxed{T(n) = \Theta(n^2)}$

put n as n/2

$$T(n) = 2\left(2T\left(\frac{n/2}{2}\right) + (n/2)^2\right) + n^2$$

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

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

$$= n^2 + \frac{n^2}{2} + 4\cdot\left(2T\left(\frac{n}{8}\right) + \frac{n^2}{4^2}\right)$$

$$= n^2 + \frac{1}{2}n^2 + \left(8\cdot T\frac{n}{8} + \frac{n^2}{4}\right)$$

$$= n^2 + \frac{1}{2}n^2 + \frac{1}{4}n^2 + 8\cdot T\left(\frac{n}{8}\right)$$

$$= n^2 + \frac{1}{2}n^2 + \frac{1}{4}n^2 + \frac{1}{8}n^2 + \ldots\ldots + 2^i T\left(\frac{n}{2^i}\right)$$

$$\leq n^2\left(\frac{1}{1-1/2}\right) = 2n^2.$$

$$T(n) = n^2 + \frac{1}{2}n^2 + \frac{1}{4}n^2 + \frac{1}{8}n^2 \ldots$$

$$\leq n^2\left(1 + \frac{1}{2} + \frac{1}{4} + \ldots\right)$$

$$= 2n^2$$

$$= \Theta(n^2)$$

$$\frac{1}{1-1/2} = 2.$$

$2T(n/2) + \Theta(n) = \Theta(n \lg n)$

4) Maximum Sub-array Problem:

```
FIND_MAXIMUM_SUBARRAY(A.low, high)                          T(n)
  if high == low
      return (low, high, A[low])
    else mid = ⌊(low+high)/2⌋                                 Θ(1)
     (left_low, left_high, left_sum) =
          FIND_MAXIMUM_SUBARRAY(A.low, high)                  T(n/2)
    (right_low, right_high, right_sum) =
          FIND_MAXIMUM_SUBARRAY(A, mid+1, high)               T(n/2)
    (cross_low, cross_high, cross_sum) =
      FIND_MAX_CROSSING_SUBARRAY(A.low, mid, high)
  if left_sum >= rightsum and left_sum >= cross_sum
      return (left_low, left_high, left_sum)
  else if rightsum >= left_sum and right_sum >= cross_sum
      return (right_low, right_high, right_sum)
  else
    return (cross_low, cross_high, cross_sum)                 Θ(1)
```

```
FIND_MAX_CROSSING_SUBARRAY(A.low, mid, high)
   left_sum = -∞
   sum = 0
  for i = mid downto low
     sum = sum + A[i]
```

```
    if sum > left_sum
        left_sum = sum
            max_left = i
right_sum = -∞
sum = 0
for j = mid+1 to high  -> sum = sum + A[j]
    if sum > right_sum
        right_sum = sum
            max_right = j
return (max_left, max_right, left_sum + right_sum)
```

5> Use iteration Method to compute $T(n) = 3T(n/4) + n$.
Determine the total running time

$$T(n) = 3T(n/4) + n.$$

$$\therefore T(n/4) = 3T(n/16) + \frac{n}{4}$$

$$T(n) = 3\left(3T(n/16) + (n/4)\right) + n$$

$$= 9T(n/16) + 3n/4 + n.$$

$$= 9\left(3T(n/64) + n/16\right) + \frac{3n}{4} + n$$

$$= 27T(n/64^3) + \frac{9n}{16^2} + \frac{3n}{4} + n.$$

$g(n) > f(n) \rightarrow g(n)$

$g(n) < f(n) \rightarrow f(n)$

$g(n) == f(n) \rightarrow g(n) \lg n$

$T(n) = 3T(n/4) + n \quad a = 3, b = 4, \quad g(n) = n^{\log_4 3} = n^{0.7}$

$f(n) = n$

$$T(n) = 27T(n/4^i) + \frac{9n}{4^{i+1}} + \frac{3n}{4} + n$$

$$\therefore \left(\frac{3n}{4}\right)^i n = 1$$

$$\therefore n = \left(\frac{4}{3}\right)^i$$

$$i = \log_{\frac{4}{3}} n = \frac{\lg n}{\lg 4/3}$$

Total running time $= cn(i)$

$$= cn\left(\log_{\frac{4}{3}} n\right) = cn\left(\frac{\lg n}{\lg 4/3}\right)$$

$$= cn \log_{4/3} n = cn \lg n$$

$$= O(n \log_{4/3} n) = O(n \lg n)$$

When should the iteration be stopped.

$$3^i T\left(\frac{n}{4^i}\right) = \quad \therefore \frac{n}{4^i} = 1 \quad \therefore n = 4^i \quad \therefore i = \log_4 n.$$

$$3^i T(1) = 3^{\log_4 n} \Theta(1)$$

$$\Theta(3^{\log_4 n}) = O(n^{\log_4 3})$$

$$T(n) \leq n + \frac{3n}{4} + \frac{9n}{16} + \frac{27n}{64} + \dots + 3^{\log_4 n} \Theta(1)$$

$$\leq n \sum_{i=0}^{\infty} (3/4)^i + \Theta(n^{\log_4 3}) = 4n + O(n)$$

$$= O(n).$$

6) (30pts)

a) Given the following partial heap sort algorithm, write the detail program for the MAX-HEAPIFY(A,i) function.

b) Given the total running time for MAX-HEAPIFY(A,i) is O(lg n), mark running time on HEAP-SORT statements and BUIL-MAX-HEAP statements and to derive the total running time for heap sort function.

c) Given the following partial heap sort algorithm, write the detail pseudocodes for the MAX-HEAPIFY(A,i) function.

d) Indicate the cost of computation complexity line by line or block by block for both function in big O & Theta Θ notation.

e) And then derive the total running time T(n) for heap sort by using Recursion tree.

HEAPSORT(A)	O(n lg n)
BUILD-MAX-HEAP(A)	O(lg n)
for i = A.length downto 2	} O(n)
exchange A[1] with A[i]	
A.heap-size = A.heap-size - 1	Θ(1)
MAX-HEAPIFY(A,1)	O(lg n)

```
BUILD-MAX-HEAP (A)
A.heap.size = A.length          ⟶ O(1)
for i = [A.length/2] down to 1
   MAX-HEAPIFY (A, i)           ⟶ O(lg n)
```

a)
```
void MAX-HEAPIFY ( float array [], int size, int id )
{
      int current = id;
      int max;
   while ( true )
   {
   int left = current * 2 + 1;
   int right = current * 2 + 2;
   if ( left >= size )
        return ;
   else if ( right >= size )
         max = left;
   else if ( array[left] < array[right] )
          max = right;
   else if ( array[left] > array[right] )
          max = left;
     if ( array[max] > array[current] )
      {
        float temp = array[max];
        array[max] = array[current];
```

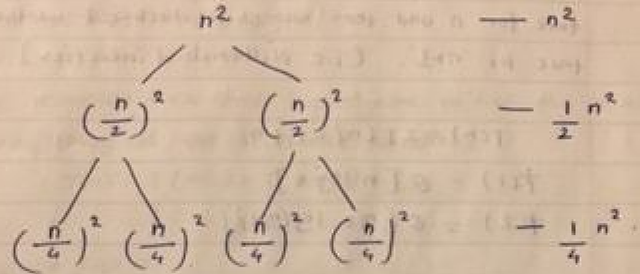
```
        array[current] = temp;
           current = max;
      }
      else
          return ;
    }
 }
```

c)

```
MAX-HEAPIFY (A,i)
l = ~~length~~ Left(i)                          O(lgn)
r = Right(i)
if l ≤ A.heap-size and A[l] > A[i]              Θ(1)
        largest = l;
   else                                         O(n)
        largest = i
if r ≤ A.heap-size and A[r] > A[largest]
        largest = r                             O(n)
if largest ≠ i
     exchange A[i] with A[largest]              O(n)
   MAX-HEAPIFY (A, largest)                     Θ(1)
```

7> Use Recursion tree for find total Running time of
$T(n) = 2T(n/2) + n^2$.

:-



Recurence = $T(n) = 2T\left(n/2\right) + n^2$

$T(n) = n^2 + 1/2\, n^2 + 1/4\, n^2 + 1/8\, n^2 + \cdots$

$\leq n^2\left(1 + \frac{1}{2} + \frac{1}{4} + \cdots\right) \leftarrow \frac{1}{1 - 1/2} = 2$

$= 2n^2 = \Theta(n^2)$

8> Use Substitution Method to prove the solution to the following recurrence equation: $T(n) = \Theta(n \lg n)$

Recurrence equation :

$T(n) = \Theta(1) \quad n = 1$

or

$T(n) = 2T(n/2) + \Theta(n) \quad n > 1$

Initially we shall prove that the given equation is true for values 1, 2, 3 and then we shall assume that it is true for 'n' and then we shall check out whether it is true for 'n+1'. (i.e. Mathematical induction)

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

$$k(1) = \Theta[n \log n]$$

$$k(2) = \Theta[n/2 \log(n/2)]$$

Let us assume the given expression is true for $\boxed{k(i)}$ and we shall prove it for $\boxed{k(i+1)}$

$$k(i) = [n/2] \quad k(i+1) = [n]$$

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

$$= 2\left[c(n/2) \cdot \log(n/2)\right] + cn$$

$$= cn \log[n/2] + cn \qquad \log_2^n \quad \lg$$

$$= cn\left[\log_2^n - \log_2^2\right] + cn$$

$$= cn\left[\log_2^n - 1\right] + cn$$

$$= cn \log_2^n - cn + n$$

$$= cn \log n - n(c-1)$$

$$\leq cn \log n.$$

Our assumption i.e. $k(i)$ true

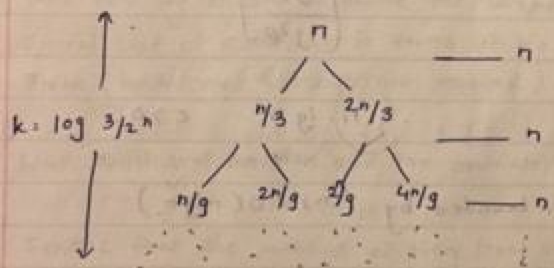
Solution to the given recurrence equation is.

$$T(n) = \Theta(n \lg n).$$

g> Use recursion tree and log algebra to find the total running time of the recurrence equation.

$$T(n) = T(n/3) + T(2n/3) + cn.$$

S:- Recursion tree :



Total: $O(n \lg n)$.

We stop when $\frac{n}{3^i} = 1$

$$n = 3^i$$

$$i = \log_3 n$$

We c

we stop $\left(\frac{2}{3}\right)^k \cdot n = 1$

$n = \left(\frac{3}{2}\right)^k$

$k = \log_{3/2} n = \frac{\lg n}{\lg 3/2}$

Total running time = $n \cdot k$

$= n\left[\frac{\lg n}{\lg 3/2}\right]$

$= n \cdot \lg n \qquad c > 0.$

$\therefore$ upper bounded by $T(n) = O(n \lg n)$

10)

a) Write quick sort algo with partition function.

b) Use recursion tree in best case partition to describe the total Running time of Quick sort $T(n) = \Theta(n \lg n)$

c) Write the program algorithm in some detail for partition function in the following QUICKSORT (A, P, r) function, given A an array with first index P & with

with last index r.

QUICKSORT (A.P.r)

1. if p<r
2. q = PARTITION(A.P.r)
3. QUICKSORT (A.P.q-1)
4. QUICKSORT (A.q+1, r)

d) Assuming the Partition function of QuickSort always splits the sub-array of array A into 40% & 60%, using recursion tree and log algebra to derive the computation time for such special case of Quick sort in terms of big O or in term of Theta (both proof & result are required)

e) Write Quick sort function and the associated Partition algorithms

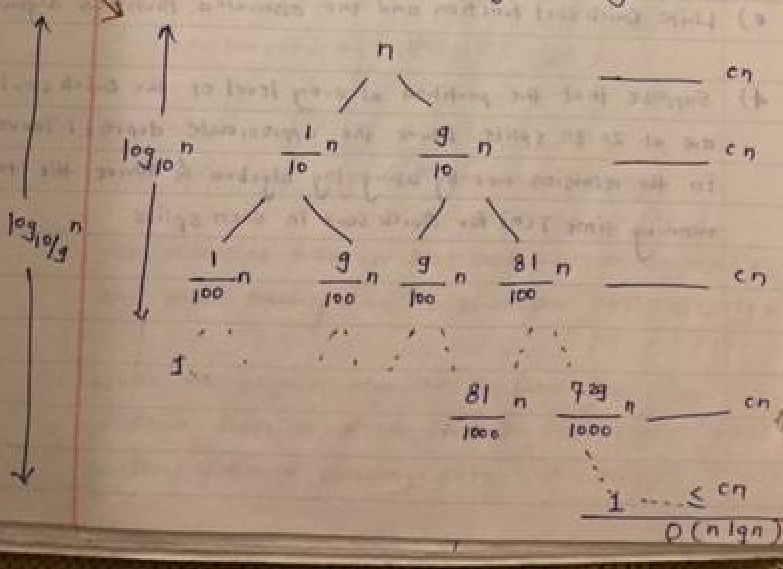
f) Suppose that the partition at every level of the Quick sort are at 20-80 splits. Derive the approximate depth of leaves in the recursion tree by using log algebra & derive the total running time T(n) for Quick sort in such splits

(ake,r)

```
PARTITION(A,p,r)
    x = A[r]
    i = p-1
    for j = p to r-1
        if A[j] ≤ x
            i = i+1
            exchange A[i] with A[j]
    exchange A[i+1] with A[r]
    return i+1
```

Balanced Partitioning

Recursion tree of quicksort in which partition always produces a 9 to 1 split, yielding a running time of $O(n \lg n)$



$$\boxed{T(n) = T(9n/10) + T(n/10) + cn}$$

$$\therefore \left(\frac{9}{10}\right)^x \cdot n \cong 1$$

$$\left(\frac{9}{10}\right)^x = \frac{1}{n}$$

$$\therefore n = \left(\frac{10}{9}\right)^x$$

$$x = \log_{10/9} n = \frac{\log n}{\lg 10/9}$$

Total running time $= n \cdot x$

$$= n \cdot \frac{\lg n}{\lg 10/9}$$

$$T(n) = n \cdot \lg n \qquad c > 0$$

* We can also write:

9:1 split still a partition.

$$\therefore T(n) = T\left(\frac{9}{10}n\right) + T\left(\frac{n}{10}\right) + \Theta(n) \leq \left(\log_{10/9} n\right) \cdot \Theta(n)$$

$$= \Theta(\lg n) \cdot \Theta(n)$$

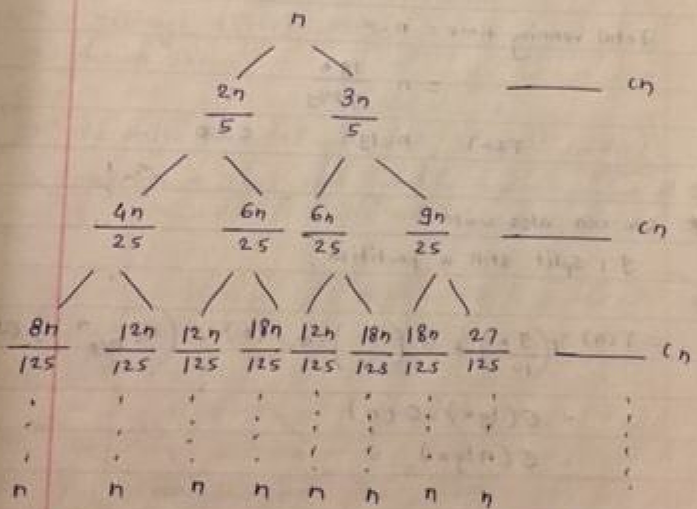
$$= \Theta(n \lg n)$$

$\therefore$ Balanced partition for quick sort $T(n) = \Theta(n \lg n)$.

d) $40\% = \frac{40}{100} = \frac{4}{10} = \frac{2}{5}$

$60\% = \frac{60}{100} = \frac{6}{10} = \frac{3}{5}$

The recursion tree is



Total $\Theta(n \lg n)$

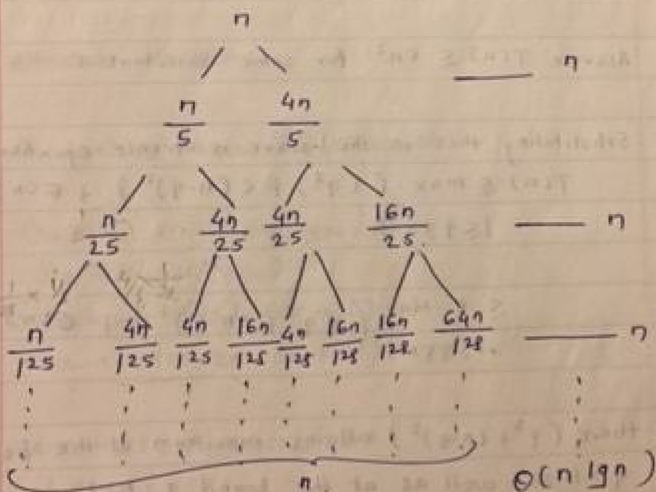
∴ Here the length of the tree is $\log n$.

∴ The total computation time is $O(n \lg n)$

f) $20 = \frac{20}{100} = \frac{2}{10} = \frac{1}{5}$

$80 = \frac{80}{100} = \frac{8}{10} = \frac{4}{5}$

The recursion tree :



11) For the Quicksort use following recurrence formula,

$$T(n) = \underset{1 \le q \le n-1}{Max} (T(q) + T(n-q)) + \Theta(n)$$

prove that quicksort is always upper bounded by n^{**2} i.e n^2. (using substitution Method)

$$T(n) = \underset{1 \le q \le n-1}{Max} (T(q) + T(n-q)) + \Theta(n)$$

Assume $T(n) \le cn^2$ for some constant c.

Substituting this in the above recurrence equation we get

$$T(n) \le \underset{1 \le q \le n-1}{max} ((q^2) + c(n-q)^2) + \Theta(n)$$

$$\le c \cdot \underset{1 \le q \le n-1}{Max} ((q^2 + (n-q)^2) + \Theta(n)$$

there $(q^2 + (n-q)^2)$ attains maximum at the bound $q = 1$ as well as at the bound $q = n-1$, as it can be seen since the second derivation of $(q^2 + (n-q)^2)$ with respect to q is positive.

This observation yields that $(q^2 + (n-q)^2) \le (n-1)^2$

$$\therefore T(n) \le c(n-1)^2 + \Theta(n)$$
$$\le cn^2 - c(2n-1) + \Theta(n)$$
$$\le cn^2$$

Here we can choose sufficient large c to dominate $2n-1$

Thus $T(n) = O(n^2)$

Hence, quicksort is always upper bounded by n^2.

* ~~Alternate Method :~~

12> Randomized version of quicksort :

```
RANDOMIZED_PARTITION (A,P,r)
   i = RANDOM(P,r)
   exchange A[r] with A[i]
   return PARTITION(A,P,r)
```

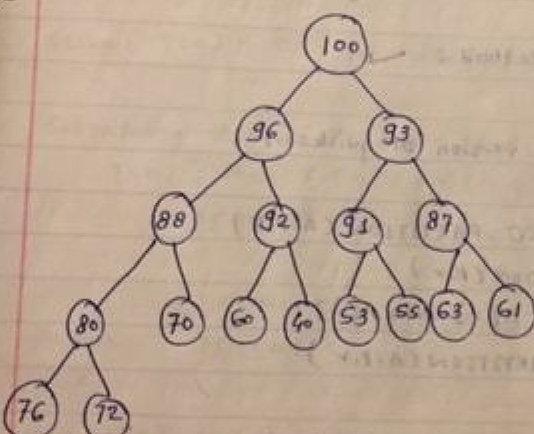
```
RANDOMIZED_QUICKSORT (A,P,r)
   if P<r
        q= RANDOMIZED_PARTITION (A,P,r)
        RANDOMIZED_QUICKSORT (A,P,q-1)
        RANDOMIZED_QUICKSORT (A,q+1,r)
```

13> (35pts)

a) The following array represent a heap. Draw a tree to represent the heap.

index	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
key	100	96	93	88	92	91	87	80	70	60	40	53	55	63	61	76	72

:-



b) Regard the heap as a Priority Queue show the resulting array / heap after an item with key = 98 is inserted into the priority queue

x	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	100	98	93	96	88	92	91	87	80	70	60	40	53	55	63	61	76

18

c) Heap Sort builds a heap & then enters a loop. Regard the original heap (item a) as first step in using Heapsort. Show the contents of array after one execution of the loop in HEAPSORT

index	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
key	72	96	93	83	92	91	87	80	70	60	40	53	55	63	61	76	-

d> Use iteration Method to find the T(n) for

$$T(n) = T(n-2) + 1.$$

8.

here, $T(n) = T(n-2) + 1$

Now $T(n-2) = T(n-2) - 2 + 1$

Then,

$$T(n) = [T(n-2) - 2 + 1] + 1$$

iterating

$T(n-2)$ gives $T(n) = [T(((n-2)-2)-2)+1) + 1] + 1] + 1$

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

or.

$$\sum n + n-1 + n-2 + n-3 + \dots (n-k+1) + T(n-k)$$

$$i = \sum_{i=n-k+1}^{n} + T(n-k) \qquad \text{for } n \geq k.$$

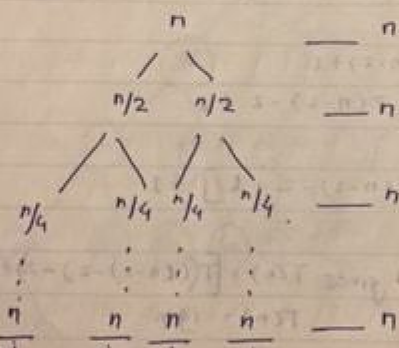
To stop recursion we should have $n-k = 0$ i.e. $k = n$.

$$\sum_{i=1}^{n} i + T(0) = \sum_{i=1}^{n} i + 0 = n\frac{n+1}{2} = O(n^2)$$

14) Suppose that the splits at every level of quicksort are in the proportion $1-\alpha$ to α, where $0<\alpha \le 1/2$ is a constant. Show that the minimum depth of a leaf in the recursion tree is approximate $-\lg n / \lg \alpha$ & the maximum depth is approximately $-\lg n / \lg(1-\alpha)$.

: Given : $0 \le \alpha \le 1/2$

Assume $\alpha = 1/2$ the position will undergo as follows



Total: $n\lg n$.

The tree will stops when $\frac{n}{2^i} = 1$ $n = 2^i$

$i = \log_2 n = \lg n$.

$$\text{height} = \lg n$$

Here the depth of the tree $i = \lg n$.

Total running time $= O(n \lg n)$

15> Prove $2^{n+1} = O(2^n)$

$\underline{\circ}$ $n \to 1$ $\quad 2^{1+1} = 2^2$

$n \to 2$ $\quad 2^{2+1} = 2^3$

$n \to 100$ $\quad 2^{100+1} = 2^{101}$

With the increase in value of n.

The total value is increasing but adding 1 to the total value. only the fractional value is changing 2^{n+1} is only factorial of n

$$2^{n+1} = O(2^n)$$

Let $f(n) = 2^{n+1}$

$g(n) = 2^n$

According to $O f(x)$ is upper bounded to $g(x)$.

2^{n+1} is upper bounded to 2^n.

$\therefore \; 0 \leq 2^{n+1} \leq c \cdot 2^n$

$\Rightarrow c \cdot 2^n = O(2^n)$

8) Solution of substitution Method :

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

Our goal is : $2T(n/2) + \theta(n) = O(n \lg n)$

Thus, we need to show that $T(n) \leq cn \lg n$ with an appropriate value of c or choice of c.

Assume

$T(n/2) \leq c(n/2) \cdot \lg(n/2)$

- Substitute back into recurrence to show that $T(n) \leq cn \lg n$ follows $c \geq 1$.

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

$\leq 2[c(n/2) \cdot \lg(n/2)] + n$

$= cn \cdot \lg (n/2) + n$

$= cn \lg n - cn \lg 2 + n$

$= cn \lg n - cn + n$

$\leq cn \lg n$ for $c \geq 1$

$= O(n \lg n)$ for $c \geq 1$

16> Show that an n-element heap has height of $[\lg n]$

: To prove, n-element heap has $h = [\lg n]$

Now,

Suppose the height of tree is h.

Then,

$$2h \leq n \leq 2h+1$$

i.e. the number of element is more than minimum height and less than maximum height (depth).

Taking log on both sides,

$$\log 2h \leq \log n \leq \log (2h+1)$$

i.e. $\log 2h \leq \log n \leq \log 2h + \log 1$

i.e. $\log 2h \leq \log n \leq \log 2h$.

i.e. $h = \log n$ — proved.

(Exam 1)
spring 17> Write a recurrence equation for the total running time $T(n)$ of FIND-MAXIMUM-SUBARRAY algorithm & calculate the $T(n)$.

18> Proof that the function $f(n) = 5n^2+10$ is $O(n^2)$ by computing the constant c & n_0.

:-

$$f(n) \le c(g(n)$$
$$5n^2+10 \le c \cdot n^2$$
$$5n^2 - cn^2 \le -10$$
$$n^2(c-5) \geq 10$$
$$n^2 \geq \frac{10}{c-5}$$

for $c = 6$, $n = 4$, we have

$$16 \geq 10$$

Thus, the complexity is $O(n^2)$

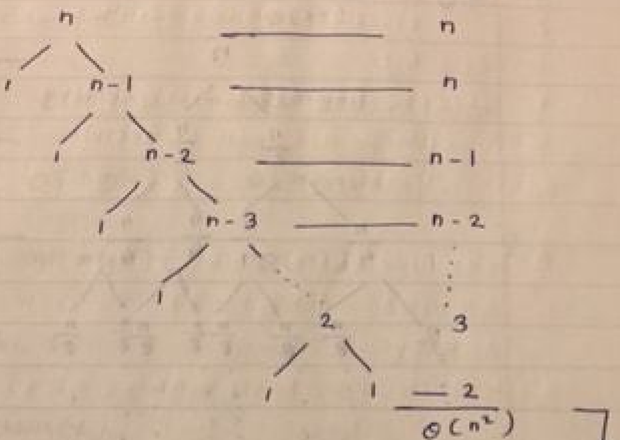
19> Derive that in the worst case scenario the performance of the quick sort will be $O(n^2)$ (draw comparison tree)

:- In the quick sort, we divide the list by implementing the following 4 steps :

i> Find the pivot element x in the list.

ii> Insert all the element less than x in a sublist L.

when input array is already completely sorted.



$T(n) = T(n-1) + \Theta(n) \longrightarrow n + n + (n-1) + (n-2) + \dots 2$

$= \frac{n \cdot n(n+1)}{2} - 1$

$= n^2$.

$= \sum_{k=1}^{n} \Theta(k)$

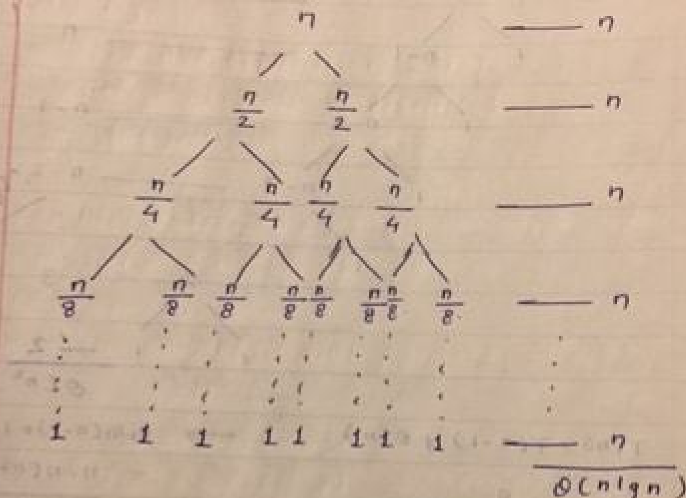
$= \Theta\left[\sum_{k=1}^{n} \cdot k\right]$

$= \Theta\left(\frac{n(n+1)}{2}\right)$

$= \Theta(n^2)$

Quick sort doesn't require additional memory storage. sorts the array on the same spot. The worst case doesn't happen very often

20) Quick sort best-case partitioning i.e. $\Theta(n \lg n)$ using recursion tree.



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

Partitioning time.

$\frac{n}{2^i} = 1$

$n = 2^i$

$i = \lg n$.

Total running time $= cn \cdot i$

$= cn \cdot \lg n$

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

* Time complexity

	Best	Worst
Insertion	$\Theta(n)$	$\Theta(n^2)$
Merge	$\Theta(n\lg n)$	$\Theta(n\lg n)$
Heap	$O(n\lg n)$	$O(n\lg n)$
Quick	$\Theta(n\lg n)$	$\Theta(n^2)$

21) Priority Queue :

```
HEAP_MAXIMUM(A)
return A[1]
```

```
HEAP_EXTRCT_MAX(A)
if A.heap-size < 1
    error "heap underflow"
max = A[1]
A[1] = A[A.heap-size]
A.heapsize = A.heapsize - 1
MAX_HEAPIFY(A, 1)
return max.
```

```
HEAP-INCREASE-KEY(A, i, key)
if key < A[i]
    error new key is ~~similar~~ ~~to~~ smaller than current key.
A[i] = key
while i > 1 and A[PARENT(i)] < A[i]
    exchange A[i] with A[PARENT(i)]
    i = PARENT(i)
```

```
MAX-HEAP-INSERT(A, key)
A.heapsize = A.heapsize + 1
A[A.heapsize] = -∞
HEAP-INCREASE-KEY(A, A.heapsize, key)
```

$$T(n) = T(n-2)+1$$
$$= T[(n-4)+1]+1$$
$$= T(n-4)+2$$
$$= T(n-6)+3$$
$$= T(n-8)+4$$
$$\vdots \qquad \vdots$$
$$= T(0) \qquad n$$

$1+n$
$= O(n)$

$T(n-i) \cong 1$

$i \cong n$