29-05-23

Design & Analysis of Algorithms

Brute force
Div & Conq.
Decrease & conq.
Dynamic Prog.
Greedy
Backtracking
etc.

Analysis → Algo → Program

→ GCD of two no.'s

(1) Euclid's

m	n
74	60
60	14
14	4
4	2
2	0

```
Algorithm Euclid(m,n)
// I.P : two non-negative, non-zero integers m,n
// OP : GCD of (m,n)

begin
  while(n ≠ 0):
    begin
      r ← m mod n
      m = n
      n = r
    end
  return m
end
```

(ii) Successive integer check

```
Algorithm IntCheck(m,n)
// I/P : two non-negative integers m,n
// O/P : GCD of m/n
begin
    t = min(m,n)
    while (m mod t != 0 OR n mod t != 0)
        t = t-1
    return t
end
```

(iii) Middle school method

$m = 72$, $n = 60$

$m = 72 = 2 \times 2 \times 2 \times 3 \times 3$

$n = 60 = 2 \times 2 \times 3 \times 5$

$GCD = 2 \times 2 \times 3 = 12$

(iv) Subtraction

m	n
72	60
12	60
12	48
12	36
12	24
12	12

Prime numbers

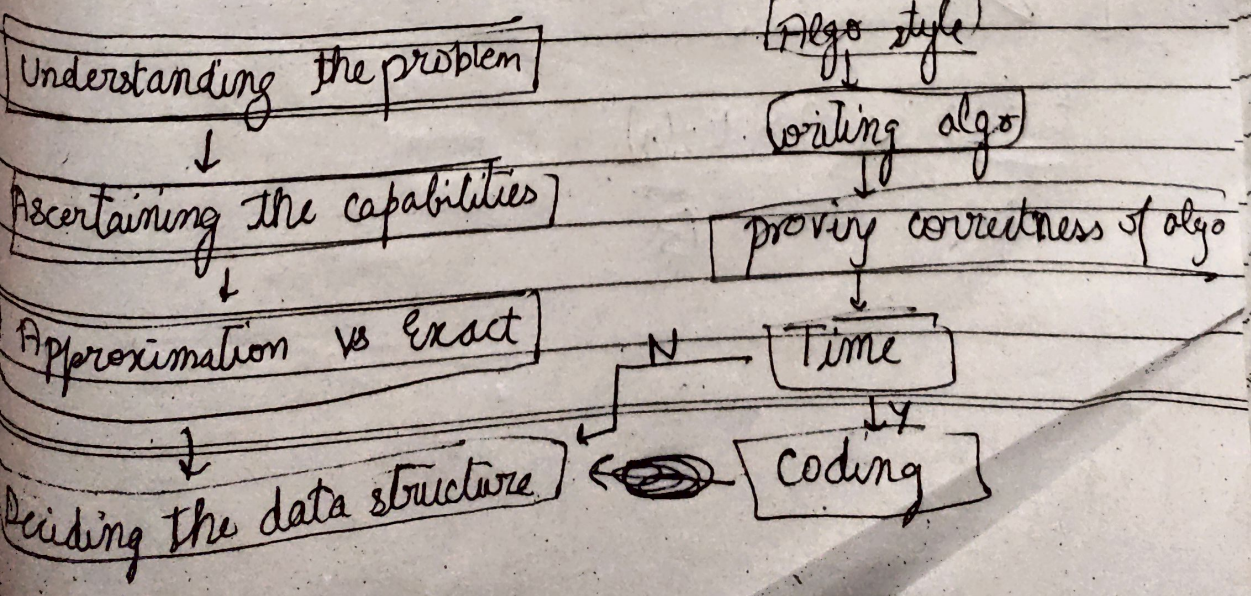
(i) Sieve of Erastosthenes

```
for p<-2 to n
    a[p] = p;

for p<-2 to √n
begin
    if (a[p] ≠0)
        j = p*p
        while (j<=n)
        begin
            a[j] <-0
            j = j+p
        end
end

for i<-2 To n
    if (a[i] !=0)
        print(a[i])
```

DESIGN & ANALYSIS PROCESS

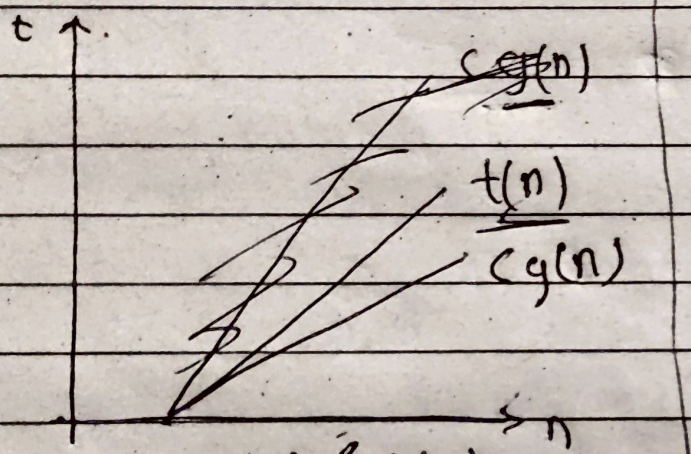


Asymptotic Notation

Ω - notation → Best
O - notation → Worst
Θ - notation → Average

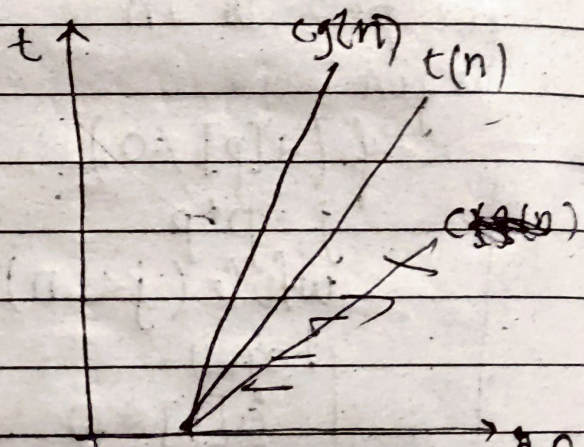
Problem
↓
Mathematical soln
↓
Algorithm

Ω - notation



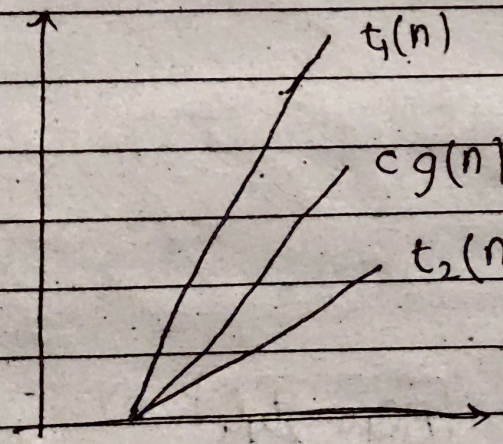
$c \cdot g(n) \leq t(n)$

O - notation



$c \cdot g(n) \geq t(n)$

Θ - notation



$t_1(n) \geq cg(n) \geq t_2(n)$

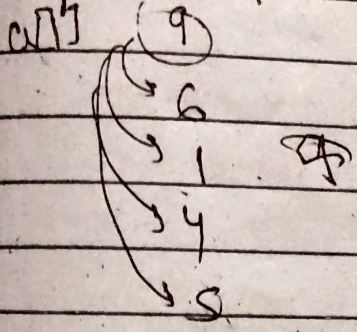
W A B

Analysis of Algorithms

Non-recursive

1. Identify the I/P size 'n'
2. Identify the key operation
3. Find how many times key operation is executed
4. Form proper mathematical series
5. Represent the series by formula

Uniqueness of elements



Algorithm unique (a, n)
// I/P array 'a' with 'n' elements
// O/P return 1 if unique else -1

```
begin
  for i ← 1 to n-1 do
  begin
    for j ← i+1 to n do
    begin
      if (a[i] == a[j])
        return -1
    end
  end
  return 1
end
```

Time Let $T(n)$ be the time taken to check the uniqueness

$$T(n) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n-i} (1)$$

If not an exp.
UB − LB + 1

$$= \sum_{i=1}^{n-1} (n-i) - (i+1) + 1$$

$$= \sum_{i=1}^{n-1} n - i$$

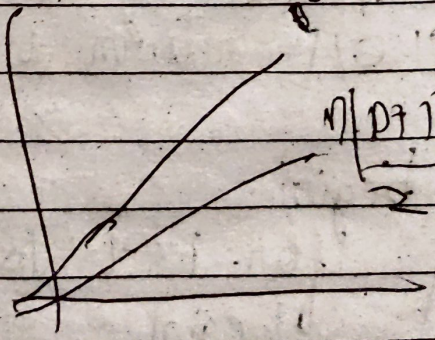
$$= (n-1) + (n-2) + \cdots + 1$$

$$(n-2) + (n-4) + \cdots + (n-2(n-1))$$

$$= \frac{n(n-1)}{2} \quad \left| \; n(n-1) + (2(2^{n-1}-1)) \right| \; n^2$$

$$= n(n-1)$$

$$= O(n^2)$$



$n(n-1)$

2. Write the algorithm to find biggest of n elements and find its time complexity.

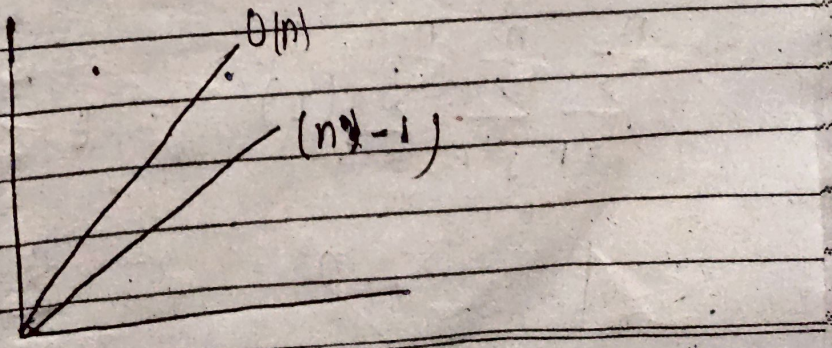
Algorithm big_n (a, n)
I/P - array a with n elements
O/P - biggest of n elements
return

```
begin
  var maxi = INT_MIN
  for i<-1 to n do
  begin
    if (a[i] > maxi)
      maxi = a[i]
  end
  return maxi
end
```

```
begin
  big = a[1]
  for i<-2 to n do
  begin
    if (a[i] > big)
      big = a[i]
  end
  return big
end
```

Let T(n) be the time taken to ~~check the max~~ find the biggest of n

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



Q. Multiplication of two $n \times n$ matrices, find time comp

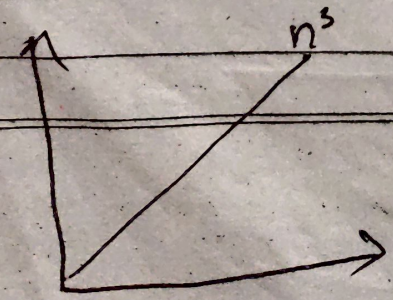
Algorithm (a[n][n], b[n][n], n)

Algorithm matmul (a, b, n)
// I/P : Two $n \times n$ matrices
// O/P : Return the multiplication of a & b

```
begin
  for i ← 1 to n do
    begin
      for j ← 1 to n do
        begin
          c[i,j] ← 0
          for k ← 1 to n do
        begin
          c[i,j] ← c[i,j] + a[i,k] * b[k,j]
        end
        end
    end
    return c
end
```

Let $T(n)$ be the time taken

$$\sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{k=j}^{n} (1) = \sum_{i=1}^{n} \sum_{j=1}^{n} 1 \; (n-j+1)$$



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

$$= n^3$$

$$= O(n^3)$$

Brute Force method

- Linear search
- Bubble sort
- Selection sort
- String matching

Linear search

Algorithm linear(a, n, key)
I/P : Array 'a' with n elments and key
O/P : returns 1 if key is found else 0

```
begin
    for i ← 1 to n
    begin
        if (key == a[i])
        Return 1
    end
    return 0
end
```

Time analysis : Let T(n) be the time taken to search for the key in the array

Best case : key found at first position

$$T(n) = \Omega(1)$$

Worst case : key not found or found at last position

$T(n) = O(n)$

Average case : key found b/w 1 to n

$$T(n) = \frac{1+2+\cdots+n}{n} = \frac{n(n+1)}{n \times 2}$$

$$= \frac{n+1}{2}$$

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

Bubble Sort

Algorithm bubblesort (a, n)
I/P : Array 'a' with 'n' elements
O/P : Array 'a' with 'n' sorted elements

```
begin
    for i<-1 to n-1 do
    for j<-1 to n-i do
        if ( a[j+1] < a[j] )
            swap ( a[j], a[j+1] )
end
```

Let T(n) be time taken

$$T(n) = \sum_{i=1}^{n-1} \sum_{j=1}^{n-i} (1)$$

$$T(n) = \sum_{i=1}^{n-1} (n-i-1)+1$$

$$= \sum_{i=1}^{n-1} n-i \quad = (n-1)+(n-2)+ \cdots 1$$

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

$$= n^2 - \frac{n}{2} \quad = n^2 - n$$

$$= O(n^2)$$

Algorithm selection_sort(a, n)
I/P : Array 'a' with 'n' elements
O/P : Array 'a' with 'n' sorted elements

```
begin
    for i<-1 to n-1 do
    begin
      mpos<-i
      for j<-i+1 to n do
      begin
        if (a[j] < a[mpos])
          mpos = j
      end
    end
end
```

$$T(n) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} (1)$$

$$= \sum_{i=1}^{n-1} (n-i-1+1)$$

$$= \sum_{i=1}^{n-1} n-i$$

$$= (n-1) + (n-2) + (n-3) + \cdots\cdots 1$$

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

$$= O(n^2)$$

~~for (int i=0; i<strlen(t); i++)~~

~~{~~

~~if~~

Brute force string matching

Algorithm: BruteForceStringMatching (T, P, m, n)
//I/P : Text & pattern string, m is length(P),
n is length(T)
//O/P : return 0 if not found else 1

```
begin
  i = 0
  while (i<n)
  begin
    j←0
    while (j<m && T[i+j] = P[j])
      j←j+1
    if (j==m) return 1
    else i++
  end
  return 0
end
```

Best case : $\Omega(m)$
Worst case : $O(mn)$

Analysis of Recursive algorithms

1) Algorithm factorial (int n)
// I/P :
// O/P :

```
begin
    if (n=0)
      return 1
    return (n* fact(n-1))
end
```

let T(n) be the time taken to find factorial

$$T(n) = n * fact(n-1)$$
$$= n * T(n-1)$$

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

$$= 3 + T(n-3)$$
$$= n + T(n-n)$$
$$= n + T(0)$$
$$= 0$$

$$T(n) = O(n)$$

Tower of Hanoi

$2^n - 1$

Algorithm TOH (n, S, E, D)

// I/P:

// O/P:

```
begin
    if (n>0)
    begin
        TOH(n-1, S, D, E)
        print (S to D)
        TOH (n-1, E, S, D)
    end
end
```

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

Fibonacci

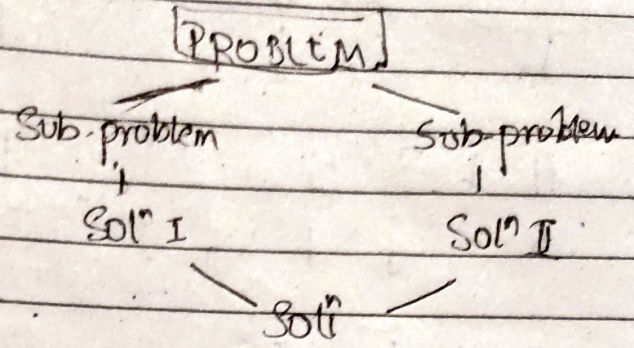
Algorithm fib(n)

```
begin
    if (n==1) return 0
    if (n==2) return 1
    return ( fib(n-1) + fib(n-2) )
end
```

```
for i=1 to n do
begin
    fib(i)
    print (fib(i))
end
```

DIVIDE & CONQUER

- Quick sort
- Merge sort
- Binary search
- S. no. mul.
- S. matrix mul.



→ Quick sort

a = {6, 9, 5, 15, 4, 10, 3, 8, 1, 7}
key i j

if (key > i) i++; else break;
if (key < j) j--; else break;

⇒ 6 9 5 15 4 10 3 8 1 7
k i j

if (i < j) swap(i, j);
else swap(key, j);

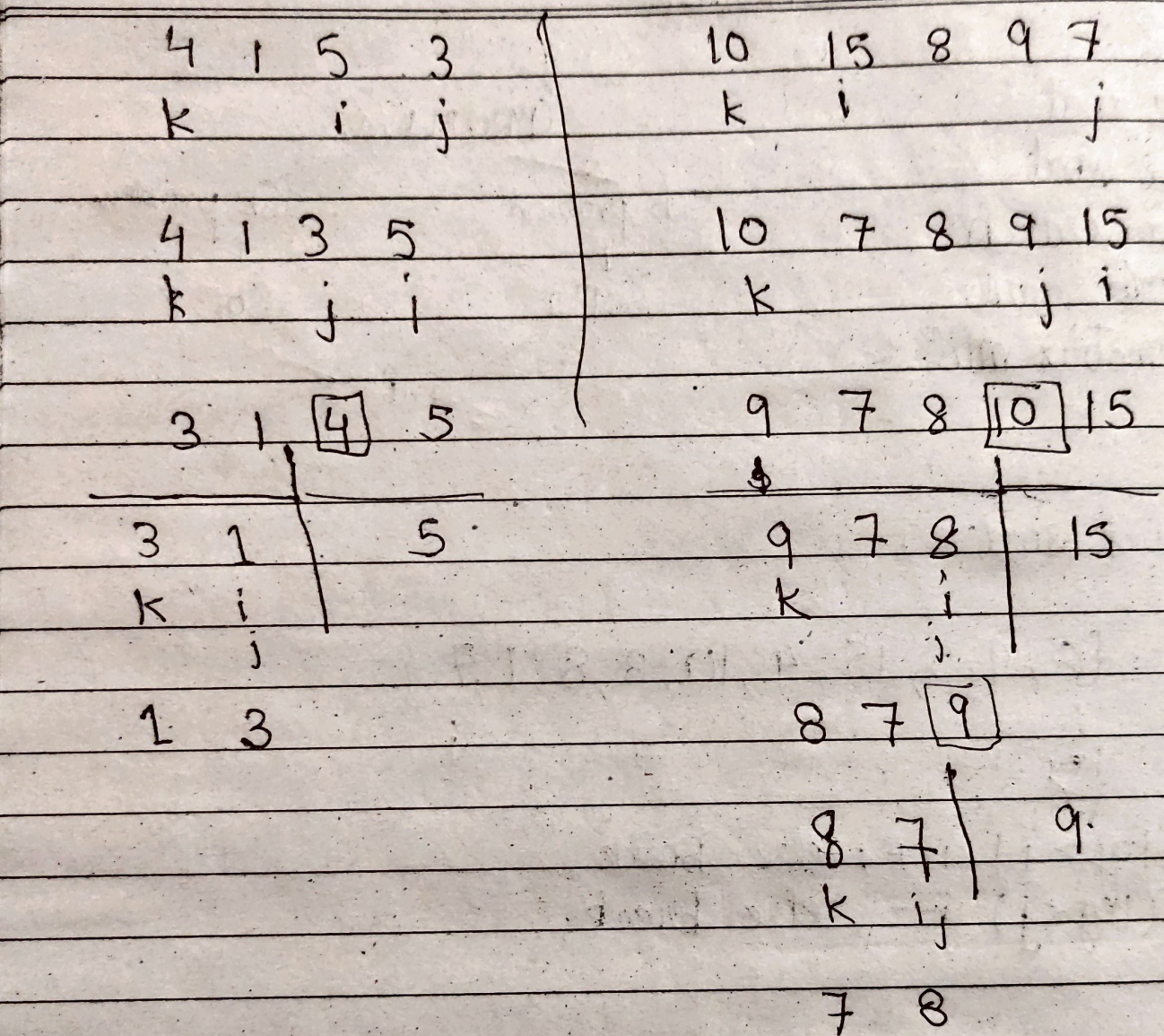
⇒ 6 1 5 15 4 10 3 8 9 7
k i j

⇒ 6 1 5 3 4 10 15 8 9 7
k j i

6

4 6 1 5 3 10 15 8 9 7
k i

Stable algo- when the elements are already sorted their pos^n must not change.



Quick Sort

```
Algorithm partition (a, low, high)
begin
  key = a[low], i = low+1, j = high
  While (1)
  begin
    while (key >= a[i])  i<-i+1
    while (key < a[j])  j<-j-1
    if (i < j)
    swap (a[i], a[j])
    else
    begin
      swap(a[low], a[j])
      return j
    end
  end
end
```

```
Quicksort (a, low, high)
begin
  if (low < high)
  begin
  j = partition (a, low, high)
  Quicksort ( a, low, j-1)
  Quicksort( a, j+1, high)
  end
end
```

```
void main()
{
int n,a[100],i;
printf("Enter the no of elements\n");
scanf("%d",&n);
printf("Enter n elements to array\n");
for(i=0;i<n;i++)
scanf("%d",&a[i]);

Quicksort(a,0,n-1);          //t=clock()      randomize
printf("sorted\n");
for(int i=0;i<n;i++)
printf("%d\n",a[i]);

int partition(int a[100], int low, int high)
{
  int key=a[low],i=low+1,j=high;
  while(1)
  {
    while(key>=a[i]) i++;
    while(key<a[j]) j--;
    if(i<j)
    {
      temp=a[i];
      a[i]=a[j];
      a[j]=temp;
    }
    else
    {
      temp=a[j];
      a[j]=a[low];
      a[low]=temp;
      return j;
}}  }
```

```
void Quicksort (int a[], int low, int high)
{
  int j;
  if (low < high)
  {
    j = partition (a, low, high);
    Quicksort (a, low, j-1);
    Quicksort (a, j+1, high);
  }
}
```

Let $T(n)$ be the time taken

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

Let $n = 2^k$

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

$$T(2^k) = 2T(2^{k-1}) + 2^k$$

Best case when equal elements on either side of pivot

$$T(2^k) = 2\left[2\left(T(2^{k-2}) + 2^{k-1}\right)\right] + 2^k$$

$$= 2^2 T(2^{k-2}) + 2 \cdot 2^{k-1} + 2^k$$
$$= 2^2 T(2^{k-2}) + 2 \cdot 2^k$$
$$= 2^3 T(2^{k-3}) + 3 \cdot 2^k$$

$$\vdots$$

$$= 2^k T(2^{k-k}) + k \cdot 2^k$$
$$= 2^k T(1) + k \cdot 2^k = k \cdot 2^k$$

$$\Rightarrow n = 2^k \Rightarrow (\log_2 n)\, n$$
$$\Rightarrow \log_2 n = k = \Omega(n \log_2 n)$$

Worst case: $T(n) = T(0) + T(n-1) + n$

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

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

$= T(n-3) + 3n$

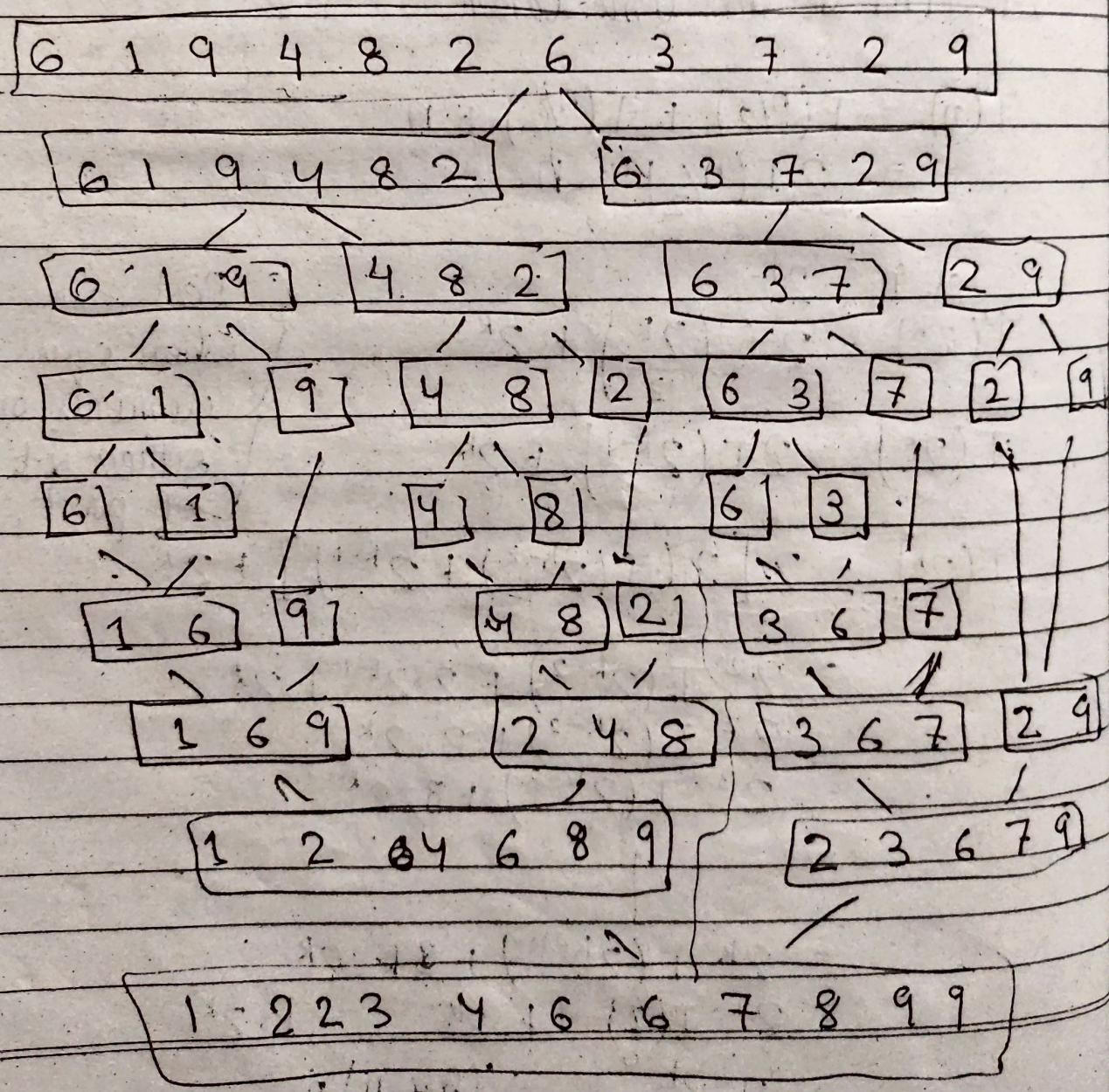
$= T(n-n) + n \cdot n$

$= T(0) + n \cdot n$

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

Merge sort

Recursion Tree



```
Algorithm mergesort (a, low, high)
begin
    if (low < high)
    begin
        mid = (low + high)/2
        mergesort (a, low, mid)
        mergesort (a, mid+1, high)
    end
end

merge (a, low, mid, high)
begin
    i = low, j = mid+1, b[100] =, k = 1,
    while (i <= mid && j <= high)
    begin
        if (a[i] < a[j])
        begin
            b[k] ← a[i]
            k ← k+1
            i ← i+1
        end
        else
        begin
            b[k] ← a[j]
            k ← k+1
            j ← j+1
        end
    end

    while (i <= mid)
    begin
        b[k] ← a[i]
        k ← k+1
        i ← i+1
    end

    while (j ← high)
    begin
        b[k] ← a[j]
        k ← k+1
        j ← j+1
    end

    for (i ← low to high
        a[i] ← b[i]
    end
```

PROGRAM

```
int main
{
  int n, a[100];;
printf (" Enter the no of elements\n"),
  scanf ("%d", &n);
printf (" Enter elements to array\n");
  for (i=0; i<n; i++)
   scanf ("%d", &a[i]);

mergesort (a, 0, n-1);

printf (" The elements after sorting\n");
  for (i=0; i<n; i++)
   printf ("%d\n", a[i]);
}

void mergesort (int a[], int low, int high)
{
  int mid;
  if (low < high)

     mid = (low+high)/2;
     mergesort (a, low, mid);
     mergesort (a, mid+1, high);
     merge(a, low, mid, high);
   }
}
```

```
void merge (int a[], int low, int mid, int high)
{
int b[100], i = low, j = mid, k = 0;
while (i <= mid && j <= high)
{
  if (a[i] < a[j])
    b[k++] = a[i++];
  else
    b[k++] = a[j++];
}

while (i <= mid)
  b[k++] = a[i++];
k = 0;
while (j <= high)
  b[k++] = a[j++];
k = 0;
for (i = low; i <= high; i++)
  a[i] = b[k++];
}
```

Binary Search

$T(n) = 1 + T(n/2)$

Let $n = 2^k \Rightarrow T(2^k) = 1 + T\left(\frac{2^k}{2}\right)$

$T(2^k) = 1 + T(2^{k-1})$

$= 1 + [1 + T(2^{k-2})]$

$= 2 + T(2^{k-2})$

$= 3 + T(2^{k-3})$

$= k + T(2^{k-k})$

$= k + T(1)$

$= k + 1$

$= k$

$n = 2^k$

$T(n) = \log_2 n$

WC	6
Ac	12
	14
BC	18
AC	20
	45
WC	57

$m_1 = (3+7) \times (4+9) = 130$

$m_2 = 13 \times 4 = 52$

$m_3 = ~~-49~~ -21$

$m_4 = 7$

$m_5 = 72$

$m_6 = (-3) \times (6) = -18$

$m_7 = (-2) \times 14 = -28$

$$\begin{matrix} 37 & 51 \\ 59 & 75 \end{matrix}$$

$$\begin{matrix} 137 \\ 72 \\ \hline 65 \\ -28 \\ \hline 37 \end{matrix}$$

Strassan's Matrix Multiplication

$$\begin{bmatrix} C_{00} & C_{01} \\ C_{10} & C_{11} \end{bmatrix} = \begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{bmatrix} * \begin{bmatrix} b_{00} & b_{01} \\ b_{10} & b_{11} \end{bmatrix}$$

$$= \begin{bmatrix} m_1 + m_4 - m_5 + m_7 & m_3 + m_5 \\ m_2 + m_4 & m_1 + m_3 - m_2 + m_6 \end{bmatrix}$$

$m_1 = (a_{00} + a_{11}) * (b_{00} + b_{11})$

$m_2 = (a_{10} + a_{11}) * b_{00}$

$m_3 = a_{00} * (b_{01} - b_{11})$

$m_4 = a_{11} * (b_{10} - b_{00})$

$m_5 = (a_{00} + a_{01}) * b_{11}$

$m_6 = (a_{10} - a_{00}) * (b_{00} + b_{01})$

$m_7 = (a_{01} - a_{11}) * (b_{10} + b_{11})$

$T(n) = 7T(n/2)$

$n = 2^k$

$T(2^k) = 7T(2^{k-1})$

$= 7[7T(2^{k-2})]$

$= 7^2 T(2^{k-2})$

$T(1) = 7^k T(1)$

$= 7^k$

$T(2^k) = 7^k$

$k = \log_2 n \Rightarrow T(n) = 7^{\log_2 n}$

$\Rightarrow T(n) = n^{\log_2 7}$

$\boxed{T(n) = n^{2.807}}$

Master Theorem

If

$T(n) = aT(n/b) + f(n^d)$

$$T(n) = \begin{cases} \theta(n^d) & , a < b^d \\ \theta(n^d \log n) & , a = b^d \\ \theta(n^{\log_b a}) & , a > b^d \end{cases}$$

Time : Recursive Algorithms

- Substitution
- Master Theorem
- Recursion Tree method

e.g. $T(n) = 3(T(n/4)) + n^3$

$\Rightarrow a = 3, b = 4, d = 3$

$3 < 4^3 \Rightarrow \theta(n^3)$

Strassan's number multiplication

* digits req in no.

$a_1 a_0$
$b_1 b_0$

$\Rightarrow$ $c_2 = a_1 * b_1$

$c_0 = a_0 * b_0$

$c_1 = (a_1 + a_0) * (b_1 + b_0) - (c_2 + c_0)$

n = no. of digits

$$c = c_2 10^n + c_1 10^{n/2} + c_0$$

$a_1\ a_0$

24

57

$c_2 = 28$

$c_0 = 10$

$c_1 = (6) \times (12) - (38)$

$= 34$

10
28
34

$c = 28(10^2) + (34)10^1 + 10$

$= 2800 + 340 + 10$

$= 3150$

~~1000 + 280 + 34~~
~~28 340 10~~

1000 + 340 + 28

1368