1) I solve the recurrence relation using iteration method

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

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

Substitute the value of $T(n/2)$ in $T(n)$

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

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

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

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

$$= 2T\left[\frac{n}{2^2} + \frac{n}{2}\right]$$

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

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

Substitute the value of $T(n/2^2)$ in $T(n)$

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

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

$$= 2^3 T\left(n/2^3\right) + 3n$$

continue this step in i terms, then in general we can say

$$T(n) = 2^i T\left(\frac{n}{2^i}\right) + in \longrightarrow \text{equ①}$$

continues until the value of $\frac{n}{2^i}$ become 1

$$T(n) = 2^i T(1) + in \longrightarrow \text{equ②}$$

compare equ① and ②, $\therefore \frac{n}{2^i} = 1 \Rightarrow n = 2^i$

Apply log on both sides then the equation will be

$$\log_2 n = \log_2 2^i$$

$$\log_2 n = i \log_2 2 \quad (\log_a a^p = p \log_a a)$$

$$i = \frac{\log_2 n}{\log_2 2} \quad \left(\log_a a = \frac{\log_b a}{\log_b a}\right)$$

$$i = \log_2 n$$

Substitute i in equ ②

$T(n) = n\,T(1) + (\log_2 n) * n$

$= n(1) + n \log n$

$= O(n \log n)$

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

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

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

Substitute the value $T(n-1)$ in $T(n)$

$T(n) = c + c + T(n-2) = 2c + T(n-2)$

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

Substitute the value of $T(n-2)$ in $T(n)$

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

⋮

To assume k as constant

$T(n) = kc + T(n-k)$

Assume $n-k=0$ then $k=n$

$T(n) = nc + T(0)$

$= nc \Rightarrow O(n)$ (Big-Oh)

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

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

$T(n/2) = T(n/2/2) + 1 = T(n/4) + 1$

Substitute the value of $T(n/2)$ in $T(n)$

$T(n) = T(n/4) + 1 + 1 = T(n/4) + 2$

$T(n/4) = T(n/4/2) + 1 = T(n/8) + 1$

Substitute the value of $T(n/4)$ in $T(n)$

$T(n) = T(n/8) + 1 + 2 = T(n/8) + 3$

⋮

~~To assume k as~~

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

$\frac{n}{2^k} = 1 \Rightarrow 2^k = n$

Taking log on both sides

$k = \log_2 n$

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

$= 1 + \log_2 n$

$= \underline{O(\log n)}$

d) $T(n) = 8T(n/2) + n^2 \quad (T(1) = 1)$

$T(n) = 8T(n/2) + n^2$

Substitute $T(n/2)$ in $T(n)$

$T(n) = 8(8T(n/4) + (n/2)^2) + n^2$

$= 8^2 T(n/4) + 8(n^2/4) + n^2$

$= 8^2 T(n/4) + 2n^2$

$T(n/2) = 8T(n/4) + (n/2)^2$

$T(n/4) = 8T(n/8) + (n/4)^2$

Substitute the value of $T(n/4)$ in $T(n)$

$T(n) = 8^2(8T(n/8) + (n/4)^2) + 2n^2$

$= 8^3 T(n/8) + 8^2(n^2/16) + 2n^2$

$= 8^3 T(n/8) + 3n^2$

In general

$T(n) = 8^k T(n/2^k) + kn^2$

To stop the iteration, assume $n/2^k = 1 \Rightarrow n = 2^k$

Taking log on both sides

$k = \log_2 n$

$T(n) = 8^{\log_2 n} T(1) + (\log_2 n) n^2 \quad (8^{\log_2 n} = n^{\log_2 8} = n^3)$

$T(n) = n^3 + n^2 \log_2 n$

$= \underline{O(n^3)}$

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

Substitute the value of $T(n-1)$ in $T(n)$

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

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

Substitute the value of $T(n-2)$ in $T(n)$

$T(n) = [T(n-3) + (n-2)] + (n-1) + n$

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

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

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

$= T(n-3) + (n-2)$

$T(n)$

In general

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

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

Assume $n-k=0$ and $n=k$

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

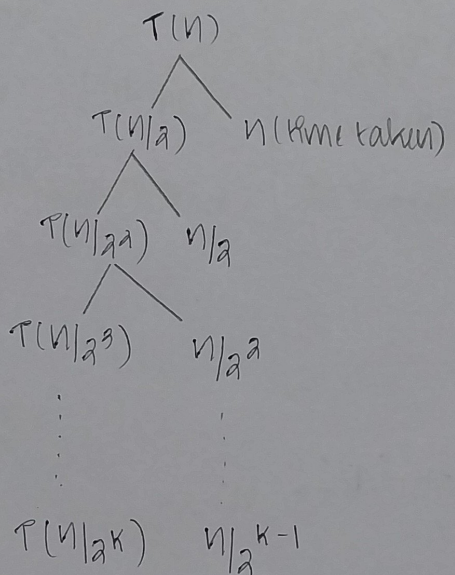
$T(n) = T(0) + 1 + 2 + \cdots + (n-1) + n$

$T(n) = T(0) + \frac{n(n+1)}{2} \Rightarrow 1 + \frac{n(n+1)}{2}$

$T(n) = \underline{On^2}$

II) Solve the recurrence relation using Recurrence Tree Method

a) $T(n) = \begin{cases} T(n/2) + n & n = 1 \\ & n > 1 \end{cases}$



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

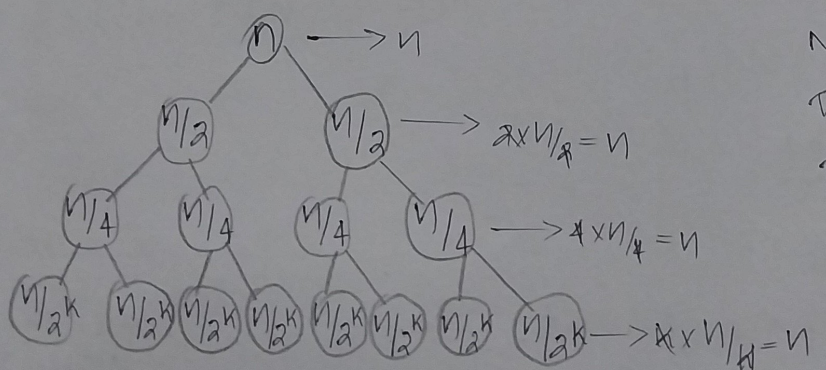
$= n\left[1 + \frac{1}{2} + \frac{1}{2^2} + \cdots + \frac{1}{2^{k-1}}\right]$

$= n \sum_{i=0}^{k-1} \frac{1}{2^i}$

Assume that $\frac{1}{2^i} = 1$ and ignore -1

$= n \times 1 \Rightarrow \underline{On}$

b) $T(n) = \begin{cases} 2T(n/2) + n & n = 1 \\ & n > 1 \end{cases}$



No. of steps $= k$

Total time for each step $= n$

$T(n) = n \times k$, assume that $\frac{n}{2^k} = 1$

$\Rightarrow n = 2^k$

Apply log on both sides

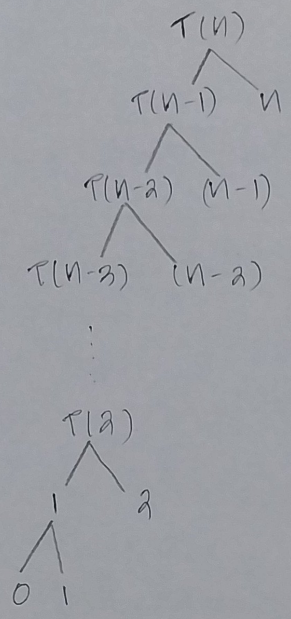
$\log n = \log_2 k$

$\log n = k \log 2$

$k = \log n$

$T(n) = n \times \log n \Rightarrow \underline{On\log n}$

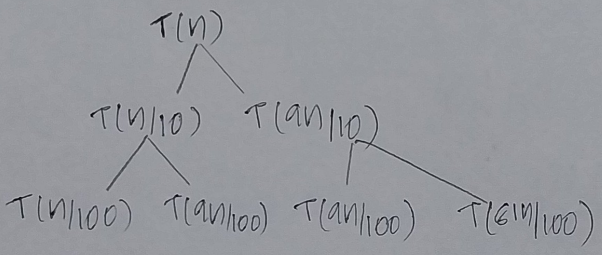
c) $T(n) = T(n-1) + n$



$T(n) = 1 + 2 + \cdots + (n-2) + (n-1) + n$ (sum of n natural numbers)

$T(n) = \frac{n(n+1)}{2} \Rightarrow \underline{O(n^2)}$

d) $T(n) = T(n/10) + T(9n/10) + n$



$T(n) = T(n/10) + T(9n/10) + n$

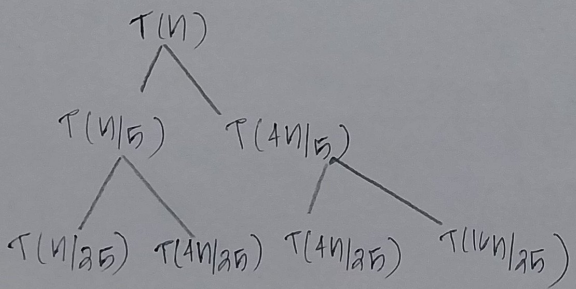
using masters theorem

$T(n) = T(an) + T(bn) + O(n)$

$a = 1, b = 9/10$

$T(n) = \underline{O(n \log n)}$

e) $T(n) = T(n/5) + T(4n/5) + n$



$T(n) = T(n/5) + T(4n/5) + n$

using masters theorem

$a = 1, b = 4/5$

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

II Apply the Masters theorem to determine the time complexity of the recurrence.

a) $T(n) = 3T(n/2) + n^2$

$a = 3, b = 2, k = 2, p = 0$

$b^k = 2^2 \Rightarrow 4$

If $3 < 4$

$\therefore T(n) = O(n^k \log^p n)$

target value 23. Implement a binary search algorithm...

$= \theta(n^2 \log^0 n)$

$T(n) = \theta n^2$

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

$a=4, b=2, k=2, p=0, b^k = 2^k => 4$

if $4=4$ then $T(n) = \theta(n^{\log_b^a} \log^{p+1} n)$

$= \theta(n^{\log_2^4} \log^1 n) = \underline{\theta(n^2 \log n)}$

c) $T(n) = 9T(n/3) + n$

$a=9, b=3, k=2, p=0$

$b^k = 3^2 = 9$

if $9=9$ then

$T(n) = \theta(n \log_b^a \log^{p+1} n)$

$= \theta(n \log_3^9 \log^1 n)$

$= \underline{\theta(n^2 \log n)}$

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

$a=2, b=2, k=1, p=0$

$b^k = 2^1 = 2$

if $2=2$ then

$T(n) = \theta(n \log_b^a \log^{p+1} n)$

$= \theta(n \log_2^2 \log^1 n)$

$= \underline{\theta(n \log n)}$

e) $T(n) = 16T(n/4) + n$

$a=16, b=4, k=1, p=0, b^k = 4^1 = 4$

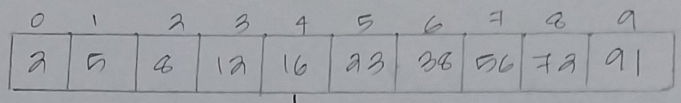
if $16 > 4$

$T(n) = \theta(n \log_b^a)$

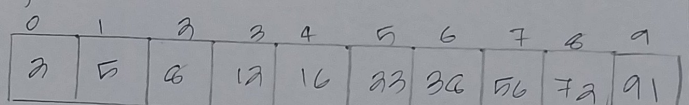
$= \theta(n \log_4^{16})$

$= \underline{\theta(n^2)}$

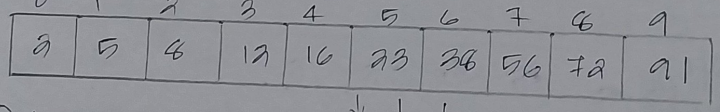
2) i) Given a sorted array arr[] = {2, 5, 6, 12, 16, 23, 36, 56, 72, 91} and a target value 23, implement a Binary search algorithm to determine whether the target exists in the array



1) low = 0
high = 9
while (0 ≤ 9) do
{
mid = (0+9)/2 = 4
if (23 < 16) x
else if (23 > 16) then low = 4+1 = 5
}



2) low = 5
high = 9
while (5 ≤ 9) do
{
mid = (5+9)/2 = 7
if (23 < 56) then high = 7-1 = 6
}

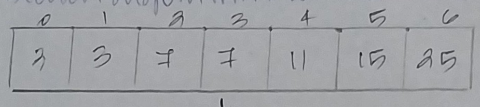


3) low = 5
high = 6
while (5 ≤ 6) do
{
mid = (5+6)/2 = 5
if (23 < 23) x
else if (23 > 23) x
else return 5;
}
return 0;
}

```
Algorithm BINSEARCH (a, n, x)
{
low = 1;
high = n;
while (low ≤ high) do
{
mid = (low + high)/2
if (x < a[mid]) then high = mid - 1;
else if (x > a[mid]) then low = mid + 1;
else return mid;
}
return 0;
}
```

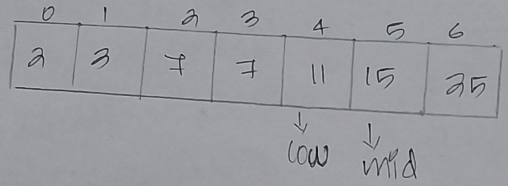
ii) Given a sorted array a[] = {2, 3, 4, 7, 11, 15, 25}, implement a Binary

search algorithm to find the target element 11



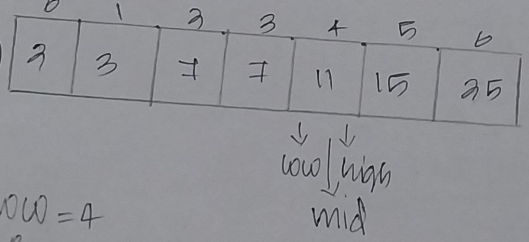
1) low = 0
high = 6
while (0 ≤ 6) do
{
mid = (0+6)/2 = 3
if (11 < 7) x
elseif (11 > 7) then low = 3+1 = 4
}

2)



low = 4
high = 6
while (4 ≤ 6) do
{
mid = (4+6)/2 = 5
if (11 < 15) then high = 5 - 1 = 4
}

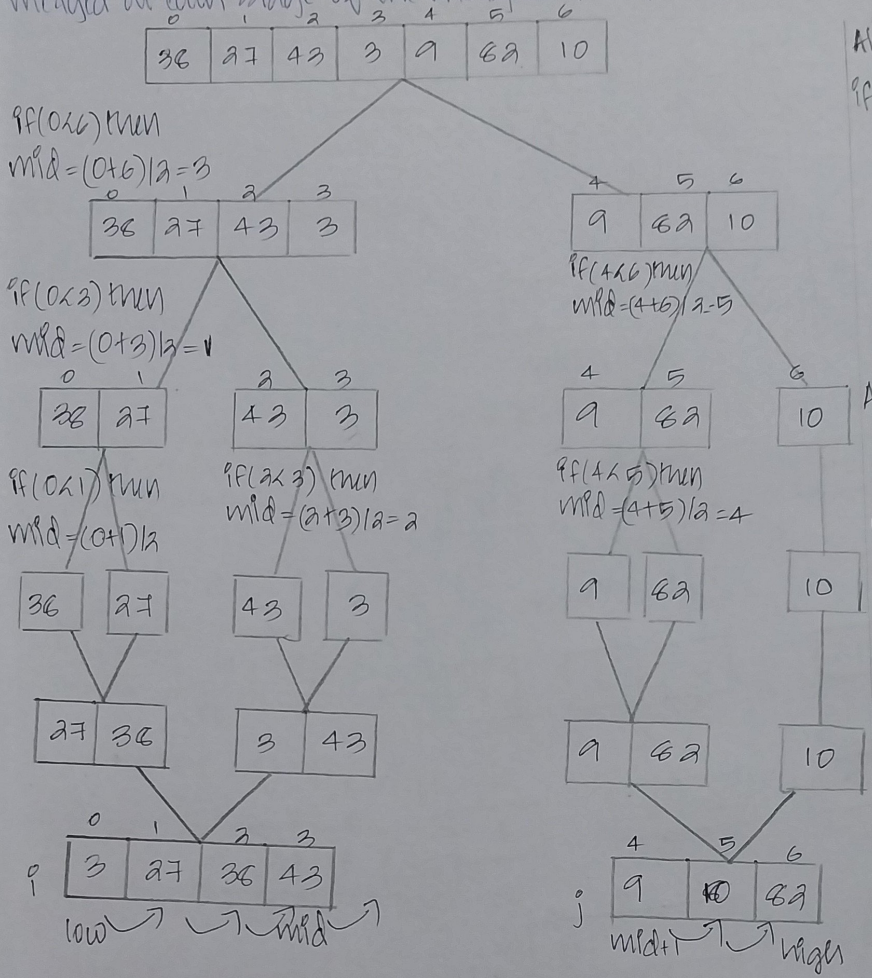
3)

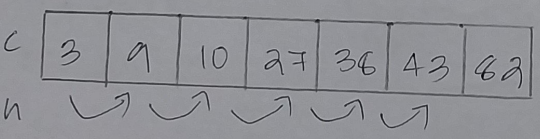


low = 4
high = 4
while (4 ≤ 4) do
{
mid = (4+4)/2 = 4
if (11 < 11) x
elseif (11 > 11) x
else return 4;
}
return 0;
}

11) Given an unsorted array a[7] = {36, 27, 43, 3, 9, 82, 10}, implement the merge sort algorithm to sort the array in ascending order. Perform

dry run, explaining and visualizing how the array is recursively divided and merged at each stage of the merge sort algorithm





while((0≤3) and (4≤6))do

1) if(3≤9) then
c[h]=3, i=i+1

2) if(27<9) x
else c[h]=9, j=4+1=5

3) if(27<10) x
else c[h]=10, j=5+1=6

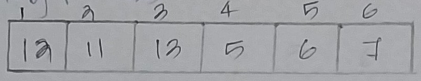
4) if(27<62) then
c[h]=27, i=1+1=2

5) if(36<62) then
c[h]=36, i=2+1=3

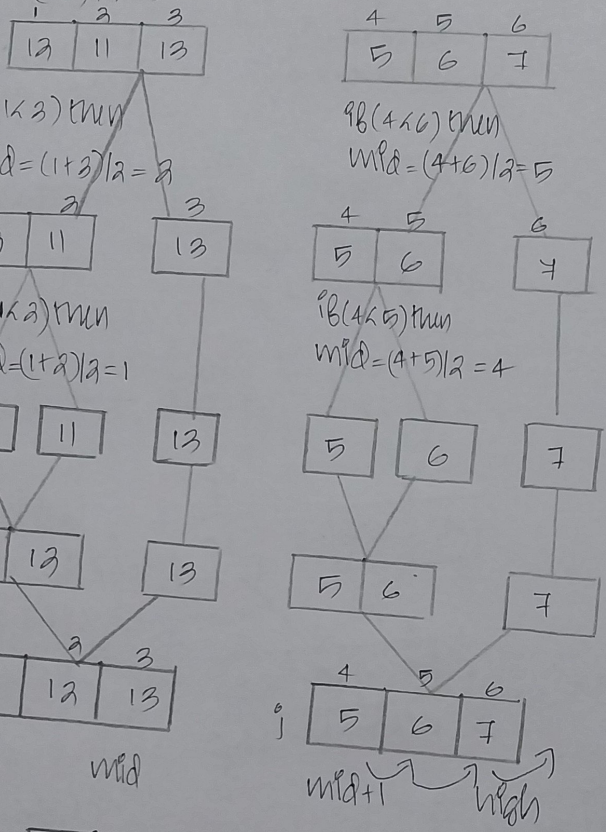
6) if(43<62) then
c[h]=43, i=3+1=4

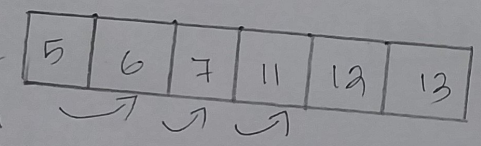
```
Algorithm Mergesort(a, low, high)
if(low<high) then {
  mid=(low+high)/2;
  Mergesort(a, low, mid)
  Mergesort(a, mid+1, high);
  Mergesort(a, low, mid, high);
}
Algorithm Merge(a, low, mid, high)
{ i=low, j=mid+1, h=low
while((i≤mid) and (j≤high)) do
{ if(a[i]<a[j]) then {
  c[h]:=a[i]
  i=i+1;
}
else {
  c[h]:=a[j]
  j=j+1;
}
  h=h+1;
while(i<=mid){
  c[h]=a[i]; i=i+1;
}
while(j<=high){
  c[h]=a[j]; j=j+1;
  h=h+1;
}
for i=low to high do
  a[i]=c[i];
}
```

iv) Formulate the Mergesort algorithm for the given array a[] = {12, 11, 13, 5, 6, 7} and sort in ascending order count and display the no of comparisons made during the sorting process



if(1<6) then
mid = (1+6)/2 = 3





while((1<3) and (4<6)) do

1) if(11<5) x
   else
   c[h] = 5; j = 4+1 = 5, h = h+1,

2) if(11<6) x
   else
   c[h] = 6; j = 5+1 = 6, h = h+1,

3) if(11<7) x
   else
   c[h] = 7; j = 6+1 = 7, h = h+1

(4)
(6)

v) Given an unsorted array a[] = {29, 10, 14, 37, 13, 20}, implement the quicksort algorithm to sort the array in ascending order and demonstrate a step by step dry run of the algorithm showing the partitioning process at each step

0	1	2	3	4	5
29	10	14	37	13	20
↑S					↑E

lb = 0, ub = 5, pivot = 29

1) while (0 < 5){
while (29 ≤ 29)
{S = 1;

0	1	2	3	4	5
29	10	14	37	13	20
	↑S				↑E

while (10 ≤ 29) { S = 2; }

0	1	2	3	4	5
29	10	14	37	13	20
		↑S			↑E

while (14 ≤ 29) { S = 3}

0	1	2	3	4	5
29	10	14	37	13	20
			↑S		↑E

while (37 ≤ 29) x
while (20 > 29) x
if (3 < 5)
{ swap (37, 20)

0	1	2	3	4	5
29	10	14	20	13	37
			↑S		↑E

2) while (3 < 5)
while (20 ≤ 29) { S = 4; }

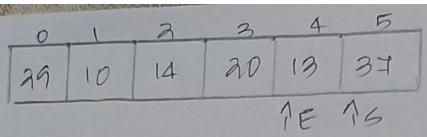
0	1	2	3	4	5
29	10	14	20	13	37
				↑S	↑E

while (13 ≤ 29) { S = 5; }

0	1	2	3	4	5
29	10	14	20	13	37
					↑S ↑E

while (37 ≤ 29) x
while (37 > 29) { end = 4 }

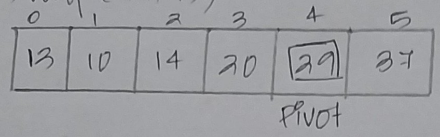
```
Algorithm quicksort (A, lb, ub)
{
if (low < high) {
    loc = partition (A, lb, ub)
    quicksort (A, lb, loc - 1);
    quicksort (A, loc + 1, ub);
    }
}

Algorithm Partition (A, lb, ub)
{
start = lb, end = ub; pivot = a[lb]
while (start < end) {
  while (a[start] <= pivot)
    { start ++ ;
    }
while (a[end] > pivot)
  { end -- ;
  }
if (start < end)
{ swap(a[start], a[end]) }
swap (pivot, a[end]);
return end;
```

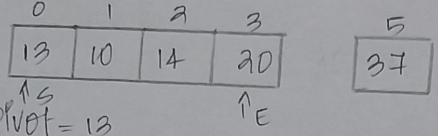


while(13 > 29) ×

if(5 < 4) ×

swap(29, 13)





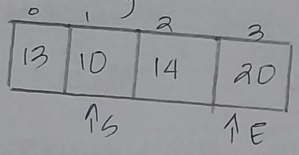
pivot = 13

lb = 0, ub = 3

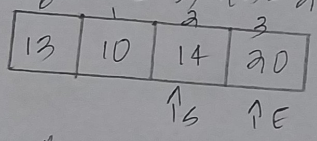
while(0 < 3)

while(13 ≤ 13)

{ s = 1 }

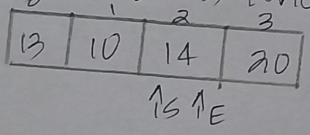


while(10 ≤ 13) { s = 2 }

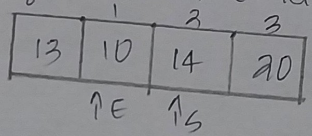


while(14 ≤ 13) ×

while(20 > 13) { end = 2; }



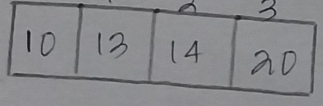
while(14 > 13) { end = 1; }



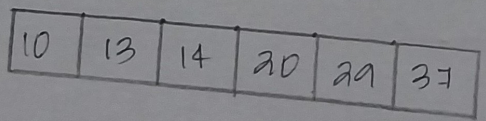
while(10 > 13) ×

if(2 < 1) ×

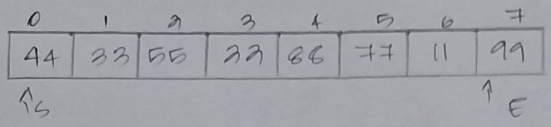
swap(13, 10)



Sorted Array:



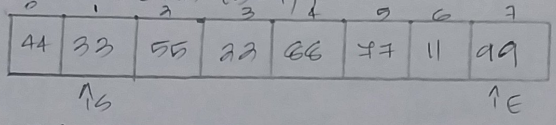
VI) Given the unsorted array a[] = {44, 33, 55, 22, 66, 77, 11, 99}, Implement the Quick sort algorithm to demonstrate the sorting process, showing the choice of pivot and partitioning at each recursion step



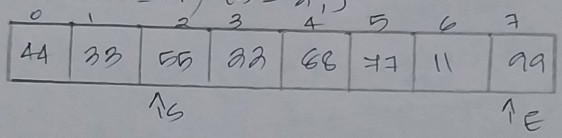
lb = 0, ub = 7, pivot = 44

while (0 < 7)

1) while (44 ≤ 44) {S = 1;}

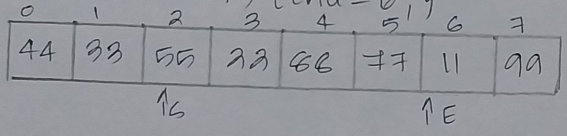


while (33 ≤ 44) {S = 2;}



while (55 ≤ 44) ×

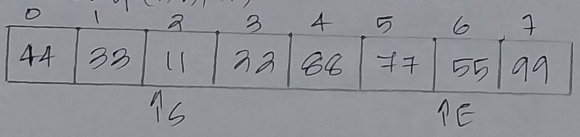
while (99 > 44) {end = 6;}



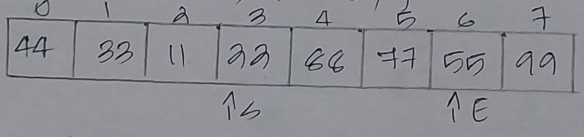
while (11 > 44) ×

if (a < c)

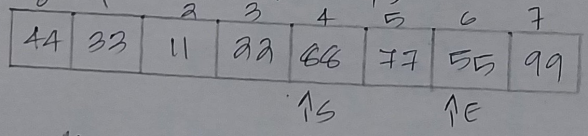
swap (55, 11)



2) while (11 ≤ 44) {S = 3;}

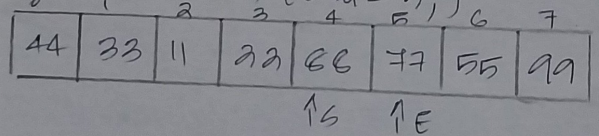


while (22 ≤ 44) {S = 4;}

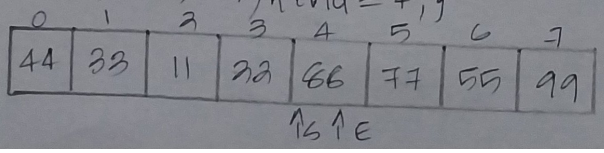


while (66 ≤ 44) ×

while (55 > 44) {end = 5;}



while (77 > 44) {end = 4;}



while(66 > 44) keep = 3;

0	1	2	3	4	5	6	7
44	33	11	22	66	77	55	99
			↑E	↑S			

while (22 > 44) X

IF (4 < 3) X

swap(22, 44)

0	1	2	3	4	5	6	7
22	33	11	44	66	77	55	99
			pivot				

0	1	2
22	33	11
↑S		↑E

lb = 0, ub = 2

pivot = 22

while (0 < 2)

1) while(22 ≤ 22) keep = 1;

0	1	2
22	33	11
↑S		↑E

while (33 ≤ 22) X

while (11 > 22) X

IF (1 < 2)

swap(33, 11)

0	1	2
22	11	33
↑S		↑E

2) while (11 ≤ 22) keep = 2;

0	1	2
22	11	33
	↑S	↑E

while (33 ≤ 22) X

while (33 > 22) keep = 1;

0	1	2
22	11	33
	↑E	↑S

while (11 > 22) X

IF (2 < 1) X

swap(11, 22)

11	22	33

4	5	6	7
66	77	55	99
↑S			↑E

lb = 4, ub = 7, pivot = 66

while (4 < 7)

1) while(66 ≤ 66) keep = 5;

4	5	6	7
66	77	55	99
	↑S		↑E

while (77 ≤ 66) keep = 6;

4	5	6	7
66	77	55	99
		↑S	↑E

while (55 ≤ 66) keep = 7;

4	5	6	7
66	77	55	99
		↑S	↑E

while (99 ≤ 66) X

while (99 > 66) keep = 6;

4	5	6	7
66	77	55	99
		↑E	↑S

while (55 > 66) X

IF (7 < 6) X

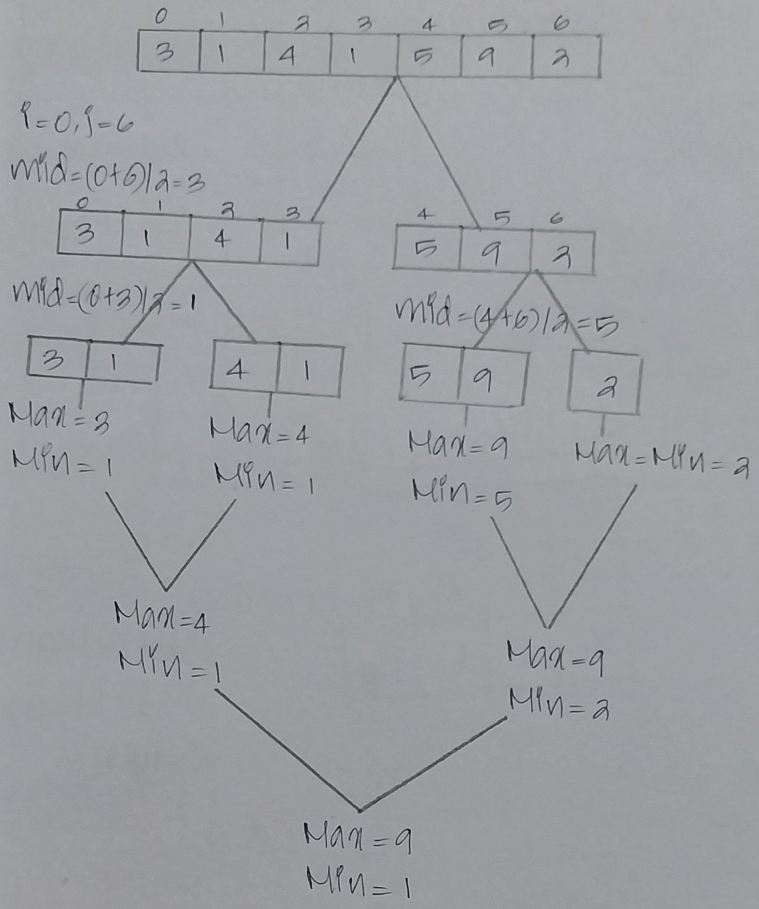
swap(66, 55)

4	5	6	7
55	77	66	99

∴ Sorted Array :

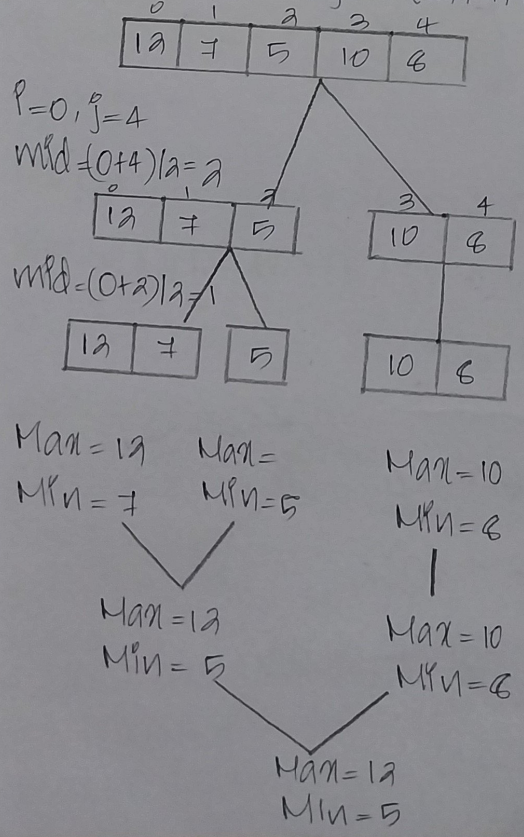
11	22	33	44	55	77	66	99

vii) Analyse and implement a divide and conquer algorithm to find the maximum and minimum values in the array [3,1,4,1,5,9,2]




```
Algorithm MaxMin(i,j,max,min)
{
if(i=j) then
   max:=min:=a[i];
else if(i=j-1) then {
 if(a[i]<a[j]) then
 {
   max:=a[j];
   min:=a[i];
 }
else
 {
   max:=a[i];
   min:=a[j];
 }}
else {
  mid=[(i+j)/2]
  MaxMin(i,mid,max,min);
  MaxMin(mid+1,j,max1,min1);
if(max<max1) then max=max1;
if(min>min1) then min=min1;
}
}
```

viii) Design a recursive divide and conquer approach to find the minimum and maximum elements in the given array of integers [18,7,5,10,6]



9x) A thief enters a store with a knapsack that can carry a maximum weight of 50kg There are three items available, each with a specific weight and value. The thief can take fractional parts of an item. Find the maximum value the thief can carry using the Greedy Algorithm.

Item	Weight	Value
1	10	60
2	20	100
3	30	120

Pi	60	100	120
Wi	10	20	30
Pi/Wi	6	5	4

Non increasing order

Pi	60	100	120
Wi	10	20	30
Xi	1	1	2/3

```
Algorithm GreedyKnapsack(m,n)
{
for i=1 to n do
   x[i]=0.0;
   u=m; p=0;
   for i=1 to n do
   {
   if(w[i]>u) then break;
    x[i]=1.0;
    u=u-w[i]; p=p+p[i];
   }
if(i≤n) then x[i]=u/w[i];
p=p+u/w[i] * p[i];
return x;
}
```

1) for i=1 to 3
   x[i]=0, u=50, p=0
for i=1 to 3
if (10>50) x
  x[1]=1
  u=50-10=40
2) for i=2 to 3
  if (20>40) x
   x[2]=1
   u=40-20=20
3) for i=3 to 3
  if (30>20) then break;
if(3≤3) then x[3]=20/30=2/3

$P_i = P_i * x_i = 60\times1+100\times1+120\times2/3 = 240$

$W_i = w_i * x_i = 10\times1+20\times1+30\times2/3 = 50$

x) A hiker has a backpack with a capacity of 60kg and wants to carry the most valuable items. There are four items, each with a given weight and value. The hiker can take fractional parts of any item. Find the maximum value that can be carried using the Greedy Algorithm

Item	weight	value
1	20	100
2	10	60
3	30	120
4	40	240

Pi	100	60	120	240
Wi	20	10	30	40
Pi/Wi	5	6	4	6

Non increasing order

Pi	60	240	100	120
Wi	10	40	20	30
Xi	1	1	1/2	0

1) For i = 1 to 4
   x[i] = 0, U = 60, P = 0
   For i = 1 to 4
   If (10 > 60) x
      x[1] = 1, U = 60 - 10 = 50
2) For i = 2 to 4
   If (40 > 50) x
      x[2] = 1, U = 50 - 40 = 10
3) For i = 3 to 4
   If (20 > 10) then break
   If (3 ≤ 4) then x[3] = 10/20 = 1/2
   actual x;

Pi = Pi * Xi = 60 x 1 + 240 x 1 + 100 x 1/2 = 350

Wi = Wi * Xi = 10 x 1 + 40 x 1 + 20 x 1/2 = 60