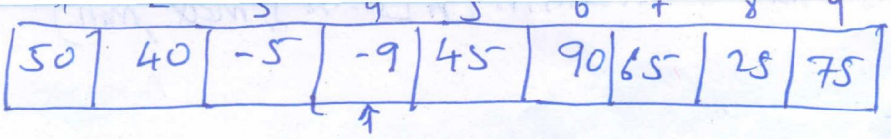
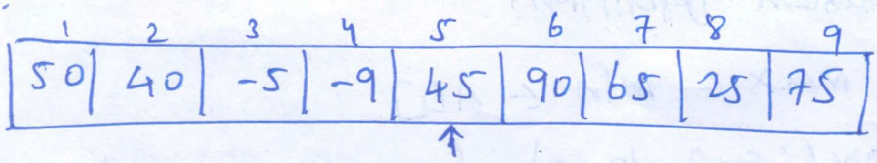
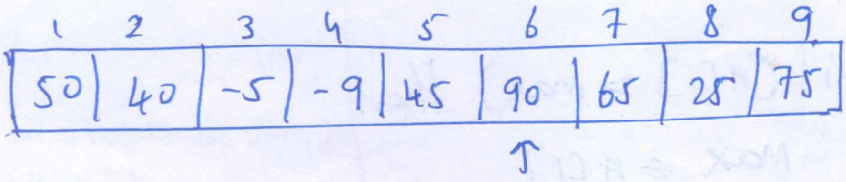


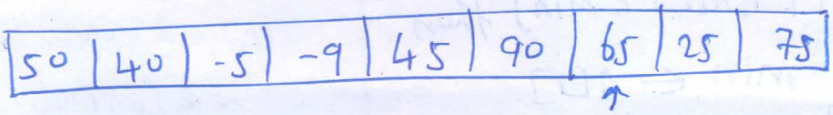
Step 5:



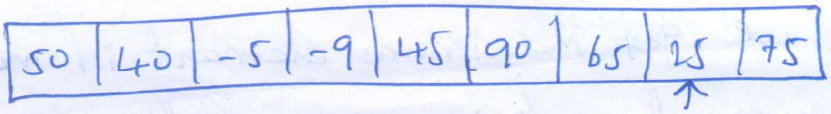
Step 6:



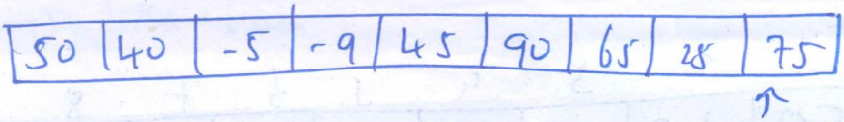
Step 7:



Step 8 :



Step 9 :



Step 10:

As we have reached at n^{th} location of the array we will terminate the procedure of finding min & max values & print min value as -9 & max value as 90.

Merge sort :-

The merge sort is a sorting algorithm that uses the divide & conquer strategy. In this method division is dynamically carried out.

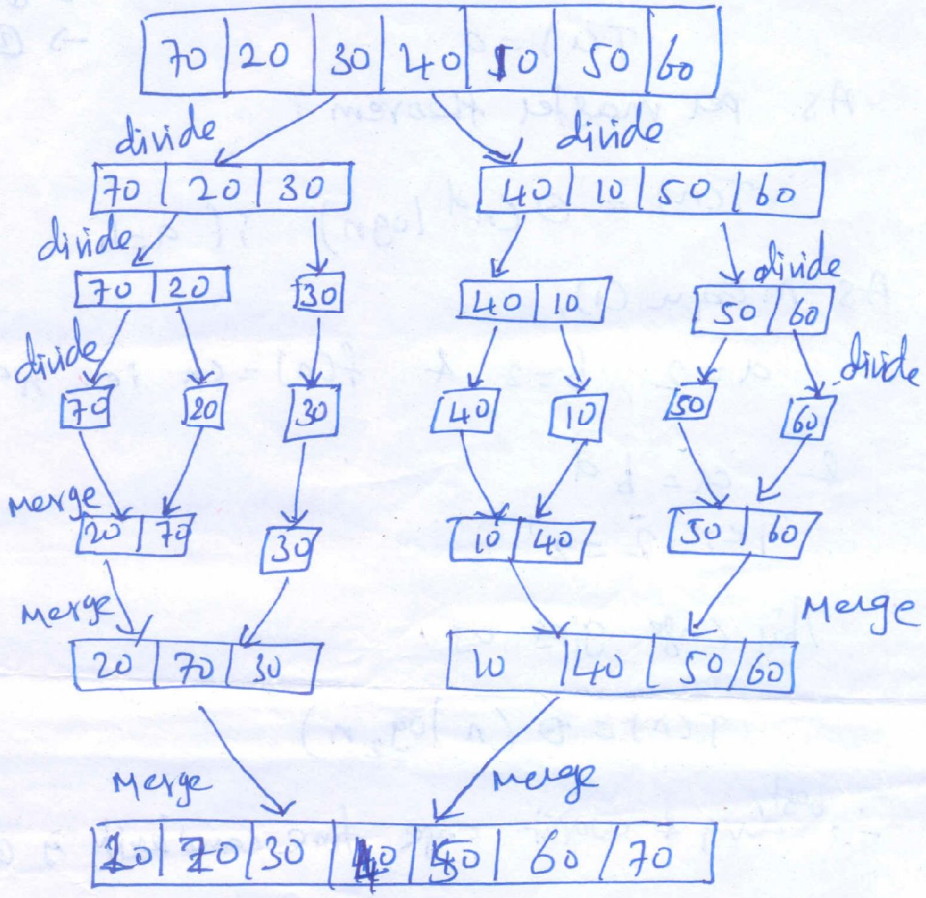
Merge sort on an input array with n elements

→ Conquer: Then sort sublist S1 & sub list S2.
→ Combine: Merge S1 & S2 into a unique sorted group.

Eg:-

70, 20, 30, 40, 10, 50, 60

Now we split this list into two sublists



Analysis

In merge sort algorithm the two recursive calls are made. Each recursive call focuses on $n/2$ elements of list. After two recursive calls one call is made to combine two sublists ie. to merge all n elements.

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

↓ Time taken by left sublist to ↓ Time taken ↓ Time taken for

... complexity of merge sort using two methods:-

→ using master theorem:

Let, the recurrence relation for merge sort is

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

$$\text{i.e., } T(n) = 2T(n/2) + cn \quad \rightarrow ①$$

$$T(1) = 0 \quad \rightarrow ②$$

As per master theorem

$$T(n) = \Theta(n^d \log n) \text{ if } a = b$$

As in equ (1),

$a = 2$, $b = 2$ & $f(n) = cn$ i.e. n^d with $d = 1$

& $a = b^d$

i.e. $2 = 2^1$

This case gives us

$$T(n) = \Theta(n \log_2 n)$$

∴ best, avg & worst case time complexity is $\Theta(n \log_2 n)$

Quick Sort:-

Quick sort is sorting algorithm that uses the divide & conquer strategy. In this method division is dynamically carried out.

→ divide: split the array into two sub array that each element in each in left & right sub array are less than & grater than middle element. The splitting of array into two sub array

List of elements sorted in array

```
int A [size], KEY, i, flag, low, high.
int BinSearch (int A[SIZE], int KEY, int low, int high);
const
low = 0
high = n-1;
{ int BM
int m;
m = (low + high)/2
if (key == A(m))
    return m;
then else if (KEY < A[m])
    Bin search (A, key, low, m-1);
    else
    Bin search (A, key, m+1, high);
}
```

```
Algorithm mergesort (int A [0--n-1] low, high)
    // Problem description : merge 2 sorted arrays into one sorted array
    // Input : Array A(0--n-1, low, high)
    // o/p : Sorted array of ele.
  if (low < high) then
  {
    mid ← (low + high)/2        → split the list at mid
    merge sort(A, low, mid)     → 1st sub list
      "     (A, mid+1, high)    → 2nd  "
    combine (A, low, mid, high) → merge of 2 sublist
  }
Algorithm Combine (A [0--n-1], low, mid, high)
{
  k ← low;  → index for array ...
  i ← low;  →  "   "  left sub list  A
```

Similarly we can write,

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

$= 2^4 T(2^{k-4}) + 4C \cdot 2^k$

$\vdots$

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

$= 2^k T(2^0) + k \cdot C \cdot 2^k$

$= 2^k T(1) + k \cdot C \cdot 2^k$ as per eqn (1) $T(1) = 0$

$2^k \cdot 0 + k \cdot C \cdot 2^k$

$0 + k \cdot C \cdot 2^k$

$= k \cdot C \cdot 2^k$

Assume $n = 2^k$ taking log on both sides

$\log_2 n = k$

$T(n) = \log_2 n \cdot Cn$

$= O(n \cdot \log_2 n)$

```
while (i <= mid & j <= high) do
{
  if (A [i] <= A [j]) then
  {
    temp [k] ← A [i]
    i ← i+1
    k ← k+1
  }
  else
  {
    temp [k] ← A [j]
    j ← j+1
    k ← k+1
  }
}
```

left

smaller ele regarding list then copy

" " right

Substitution method

$T(n) = T(n/2) + T(n/2) + cn$ for $n>1$

$T(n) = 2T(n/2) + cn$ → (3)

$T(1) = 0$ → (4)

apply substitute equation (3) assume $n = 2^k$

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

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

put $k = k-1$ then

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

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

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

$= 2^2 T(2^{k-2}) + 2 \cdot c \cdot \frac{2^k}{2} + \ldots$

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

method 1:- using master theorem

Solve eqn ①
as per master theorem

if $f(n) \in \theta(n^d)$ then

1) $T(n) = \theta(n^d)$ if $a < b^d$
2) $T(n) = \theta(n^d \log n)$ if $a = b^d$
3) $T(n) = \theta(n^{\log_b a})$ if $a > b^d$

we get,

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

$f(n) \in n^1$ $\therefore d = 1$

now $a = 2$ & $b = 2$

as from case 2 we get $a = b^d$ ie $2 = 2^1$ we get

$T(n)$ ie $C(n) = \theta(n^d \log n)$

$$C(n) = \theta(n \log n)$$

$\therefore$ Best case $\theta(n \log_2 n)$

method 2:- using Substitution:-

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

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

assume $n = 2^k$

$$C(2^k) = 2C(2^k/2) + 2^k$$

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

algorithm:-

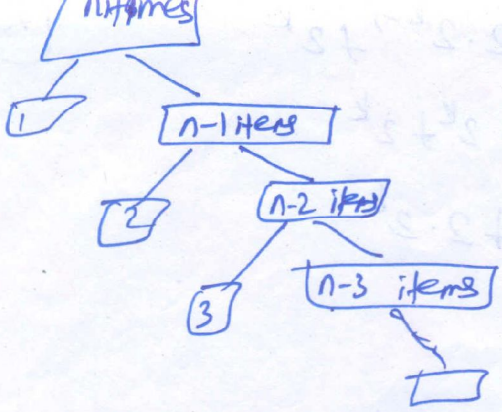
```
Algorithm quick(A[0..n-1], low, high)
  if (low < high) then
  m ← Partition(A[low..high])
  Quick(A[low..m-1])
    || (A[midH..high])

Algorithm partition(A[low..high])
  Pivot ← A[low]
    i ← low
    j ← high+1
  while (i<=j) do
  {
  while (A[i]<=Pivot) do
    i ← i+1
  while (A[j] >= Pivot) do
    j ← j-1;
  if (i<=j) then
  Swap(A[i], A[j])
  }
  Swap(A[low], A[j])
  return j
```

Analysis:-

Best case;

$C(n) = C(n/2) + C(n/2) + n \rightarrow (1)$



times req
n
n-1
...
...
1

If we assume

$A(n) = \frac{C_{avg}(n)}{(n+1)}$ then

$A(n) < A(n-1) + \frac{c'}{(n+1)}$

$A(n-1) < A(n-2) + c'/n$

$A(n-2) < A(n-3) + c'/n-1$

...

$A(1) < A(0) + c'/2$

we can write it as

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

(or)

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

Adding & cancelling these equ

$A(n) < c' * [1/2 + 1/3 + \cdots + 1/n + 1/n+1]$

But as we know

$$1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2} \approx \frac{1}{2} n^2$$

$\therefore C_{avg}(n) - (n+1) * A(n)$

$i.e.\ (n+1) * A(n) < c'(n+1) \log n$

$$C(n) = \theta(n^2) \rightarrow \text{worst case}$$

Avg Case $n * C_{avg}(n) - (n-1) * C_{avg}(n-1) = 2[C_{avg}(1) + C_{avg}(2) + \cdots + C_{avg}(n-1)] + (2n-1)$

$$n * C_{avg}(n) - (n-1) * C_{avg}(n-1) = 2 C_{avg}(n-1) + (2n-1)$$

$$n * C_{avg}(n) = (n+1) * C_{avg}(n-1) + (2n-1) < (n+1) * C_{avg}(n-1) + 2n$$

If we assume

$$(n+1) * C_{avg}(n-1) + 2n = (n+1) * C_{avg}(n-1) + c'n$$

Then,

$$(n+1) * C_{avg}(n-1) < (n+1) * C_{avg}(n-1) + c'n$$

$\div$ by $n(n+1)$ then

$$n * C_{avg}(n) < (n+1) * C_{avg}(n-1) + c'n$$

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

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

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

Substitute $C(2^{k-2})$ then

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

$= 2^2 [2C(2^{k-3}) + 2^{k-2}] + 2 \cdot 2^k$

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

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

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

Similarly we write

$C(2^k) = 2^4 C(2^{k-4}) + 4 \cdot 2^k$

$\vdots$

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

$= 2^k C(1) + k \cdot 2^k \rightarrow [C(0) = 0]$

$\therefore 2^k \cdot 0 + k \cdot 2^k$

$= k \cdot 2^k$

Now assume $n = 2^k$

$k = \log_2 n$ By taking log on both sides

$C(n) = \log_2 n \cdot n$

$= n \cdot \log_2 n$

$\therefore$ best case $= \theta(n \log_2 n)$

(Avg case
random array)

Let, $C_{avg}(n)$ denotes avg time of Quick Sort $(A[1..n])$

: Recurrence relation :

$$C(n) = C(0) + C(n-1) + n \quad \cdots (1)$$

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

$$C(n) = C(2) + C(n-3) + n \quad \cdots (3)$$

$$C(n) = C(n-1) + C(0) + n$$

Sum of all possible values by n

$$C_{avg}(n) = \frac{2}{n}\left(C(1) + C(2) + \cdots + C(n-1)\right) + n$$

$*$ both sides by n

$$n * C_{avg}(n) = 2\left(C(1) + C(2) + \cdots + C(n-1)\right) + n^2$$

Now we put $n = n-1$

$$(n-1) * C_{avg}(n-1) = 2\left(C(0) + C(1) + \cdots + C(n-2)\right) + (n-1)^2$$

$$n * C_{avg}(n) - (n-1) * C_{avg}(n-1) = 2\left[C_{avg}(1) + C_{avg}(2) + \cdots + C_{avg}(n-2)\right] + (n-1) * (n-1)$$

sanjana.I
IECFB

NO	P	R	V	T
1.	4	4	2	10
2.	4	4	2	10
3.	4	4	2	10
4.	4	4	2	10
5.	4	4	2	10
6.	4	4	2	10
7.	4	4	2	10
8.	4	4	1	9
9.	4	4	1	9
10.	4	4	2	10
11.	4	4	2	10
12.	4	4	2	10
13.	4	4	2	10
14.	4	4	2	10
15	4	4	2	10
	4	4	1.86	9.86

→ conquer: recursively sort the two sub arrays
→ combine: combine all sorted elements in group to form a list of sorted elements.

eg:-

50	30	10	90	80	20	40	70
50	30	10	90	80	20	40	70
50	30	10	90	80	20	40	70
50	30	10	90	80	20	40	70
50	30	10	40	80	20	90	70
50	30	10	40	80	20	90	70
50	30	10	40	20	80	90	70
50	30	10	40	20	80	90	
20	30	10	40	50	70	80	90
20	10	30	40	50	70	80	90
10	20	30	40	50	70	80	90

Analysis :-

Best case - $\Theta(n \log_2 n)$
Avg case - $\Theta(n \log_2 n)$
worst case - $\Theta(n^2)$

Selection sort:-

Scan the array to find its smallest element & swap it with the first element