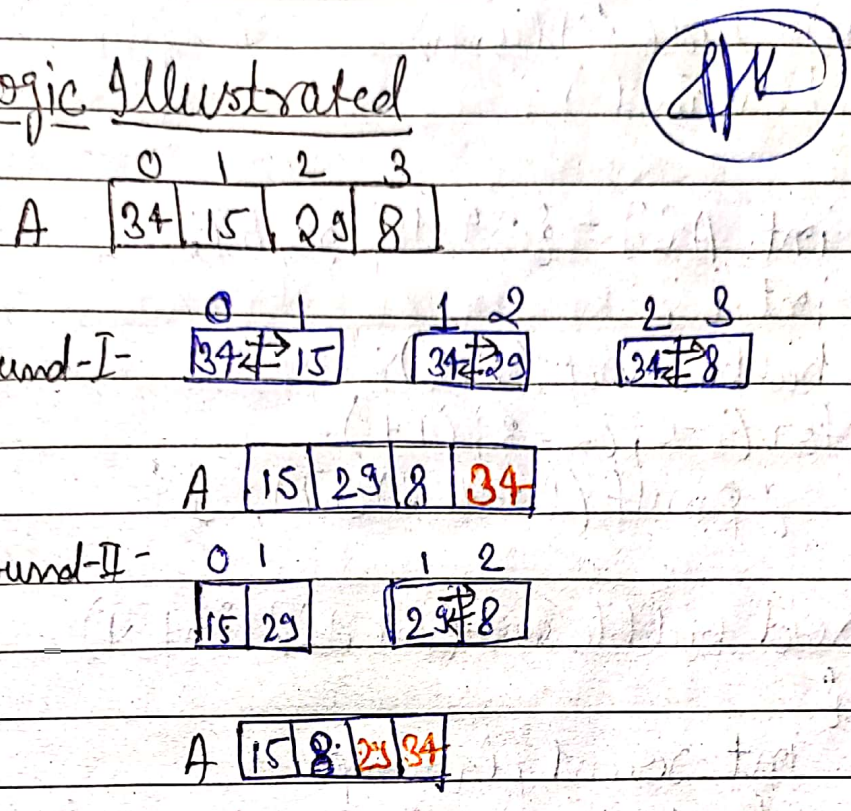
Page No.
Date / /

Bubble-Sort

Logic Illustrated



Observation - (n-1) Round comparision take place and in each round bigger value goes to last. This process is called bubble sorting.

Algorithm - Bubble_Sort(A,N); A is array of values and N is the no. of element

(I) Repeat for round = 1, 2, 3, ---N-1.
(II) Repeat for C = 0, 1, 2, ---N-1-round.
(III) If A[i] > A[i+1] then swap A[i] and A[i+1]
(IV) Return.

Page No.
Date / /

Coding In C

```
#include <stdio.h>
int main()
{
  int A[] = {34, 15, 29, 8};
  int i;
  bubble_sort(A, 4);
  for(i=0; i<=3; i++)
    { printf("%d", A[i]); }}

void bubble_sort(int A[], int N)
{
  int round, i, temp;
  for(round=1; round<=N-1; round++)
    {
      for(i=0; i<=N-1-round; i++)
        { if(A[i] > A[i+1])
            {
              temp = A[i];
              A[i] = A[i+1];
              A[i+1] = temp;
            }
        }
    }
}
```

Time complexity is $O(n^2)$.
But this time complexity is not good for large number of input. So Now we modify the algo.

Page No.
Date

Modified Bubble Sort Algo and Programs

We do modification b/c if the data is already sorted then the time complexity is O(n). If we apply previous algo the time complexity will be $O(n^2)$. So for the best case O(n).

Algo-

Bubble sort (A, N): A is an array of values and N is the number of elements.

(I) Repeat Step 2, 3, 4 for round = 1, 2, 3, ---N-1.

(II) flag = 0

(III) Repeat for i = 0, 1, 2, ---N-1-round.

If A[i] > A[i+1] then swap A[i] and A[i+1] also set flag = 1.

(IV) If flag == 0 return.

(V) Return.

Coding in C

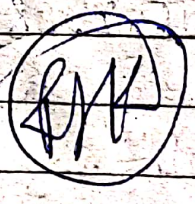
```
#include <stdio.h>
Int main()
{int A[] = { 34, 15, 29, 8};
 int i;
 bubble_sort (A, 4);
 for (i=0; i<=3; i++)
 {printf ("%d", A[i]); } }
```

Page No.
Date / /

```
void bubble_sort(int A[], int N)
{ int round, i, temp, flag;
  for (round = 1; round <= N-1; round++)
  {
    flag = 0;
    for(i = 0; i <= N-1-round; i++)
      if(A[i] > A[i+1])
      {
        flag = 1;
        temp = A[i];
        A[i] = A[i+1];
        A[i+1] = temp;
      }
    if (flag == 0) printf("%d", round);
      return;
  }
}
```

for Seeing round.

Conclusion- If data is in sorted form then time complexity is O(n).
We use flag to skipping the round if any data is already sorted.



Page No.
Date / /

Time Complexity

Worst and Average-
$O(n^2)$. Worst case occurs when array is reverse sorted.

Best case-
$O(n)$. Best case occurs when array is already sorted.

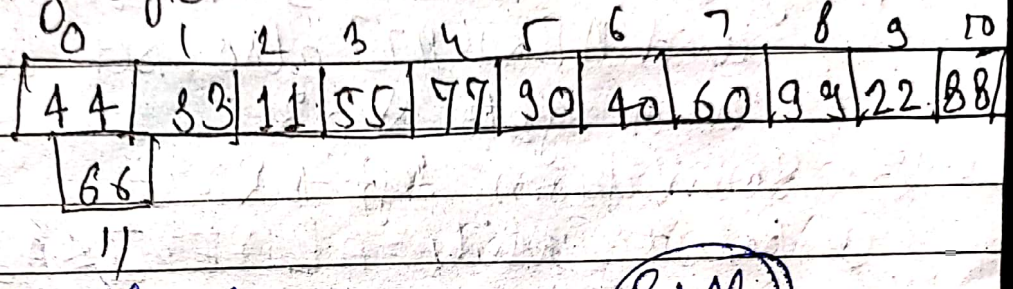
Boundary case-
Bubble Sort takes minm time($O(n)$) when elements are already sorted.

Quick Sort

Page No.
Date / /

Quick sort is an algorithm of the divide and conquer type. That is, the problem of sorting a set is reduced to the problem of sorting two smaller sets.

Working logic



- Procedure Quick.
- Algo Quick Sort.

In procedure Quick Procedure Quick -
Quick is a procedure in which 1st element is set at it's right place. And all value less then than 1st numb will be in left and all value greater than than 1st number will be in right.

Procedure -
Quick (A, N, BEG, END, LOC): Here 'A' is an array with N elements. Parameters BEG and END contains the boundary values of the sublist of 'A' to which this procedure applies. LOC keeps track of the position of the 1st element.

Page No.
Date / /

1. Set LEFT = BEG, RIGHT = END, and LOC = BEG
2. Scan from Right to left
   a) Repeat while A[LOC] ≤ A[RIGHT] and LOC ≠ RIGHT:
      RIGHT = RIGHT - 1.
   b) If LOC = = RIGHT then return
   c) If A[LOC] > A[RIGHT] then Swap A[LOC] and A[RIGHT] also set LOC = RIGHT and goto step 3
3. Scan from left to Right
   a) Repeat while A[LEFT] ≤ A[LOC] and LEFT ≠ ~~RIGHT~~ LOC:
      LEFT = LEFT + 1.
   b) If LOC = = LEFT then return
   c) If A[LEFT] > A[LOC] then swap A[LOC] and A[LEFT] also set LOC = LEFT and goto step 2.

Algorithm Quick_short

Algorithm Quicksort: This algorithm sorts an array 'A' with 'N' elements.

1. TOP = -1
2. If N > 1, then TOP = TOP + 1, LOWER[TOP] = 0, UPPER[TOP] = N - 1
3. Repeat step 4 to 7 while TOP ≠ -1.
4. POP sublist from stack BEG = LOWER[TOP], END = UPPER[TOP], TOP = TOP - 1.

44	33	11	55	77	90	40	60	99	22	88	66

5. Call QUICK (A, N, BEG, END, LOC)
6. Push left sublist onto the stack.
   If (BEG < LOC-1), then: TOP = TOP+1.
   LOWER [TOP] = BEG, UPPER [TOP] = LOC-1
7. Push right sublist onto stacks
   If (LOC+1 < END), then: TOP = TOP+1,
   LOWER [TOP] = LOC+1, UPPER [TOP] = END.
8. Exist.

Coding In C.

```
#include <stdio.h>
void quick (int [], int, int, int, int *);
#include <stdlib.h>
void quick sort (int [], int);

void quick (int A[], int BEG, int END,
            int *LOCPTR)
{
   int LEFT, RIGHT, temp;
   LEFT = BEG;
   RIGHT = END;
   *LOCPTR = BEG;
   while (A[LOC] <= A[RIGHT] &&
          LOC = RIGHT)
   RIGHT--;
   if (LOC == RIGHT)
```

```
        return;
    if (A[*LOCPTR] > A[RIGHT]) {
        temp = A[LOCPTR];
        A[LOCPTR] = A[RIGHT];
        A[RIGHT] = temp;
        *LOCPTR = RIGHT;
    }
    while (A[LEFT] <= A[*LOCPTR] && LEFT != *LOCPTR)
        LEFT++;
    if (*LOCPTR == LEFT)
        return;
    if (A[LEFT] > A[*LOCPTR]) {
        temp = A[LEFT];
        A[LEFT] = A[*LOCPTR];
        A[*LOCPTR] = temp;
        *LOCPTR = LEFT;
    }
}
void quick_sort (int A[], int N)
{
    int BEG, END, LOC, TOP = -1;
    int LOWER[10], UPPER[10];
    if (N > 1) {
        top++;
        LOWER[TOP] = 0;
        UPPER[TOP] = N-1;
    }
```

```
while (TOP!=-1){
    BEG = LOWER[TOP];
    END = UPPER[TOP];
    TOP--;
    quick(A,N,BEG,END,&LOC);
    if(BEG<LOC-1){
        TOP++;
        LOWER[TOP]=BEG;
        UPPER[TOP]=LOC-1;
    }
    if(LOC+1<END){
        TOP++;
        LOWER[TOP]=LOC+1;
        UPPER[TOP]=END;
    }
}
int main()
{ int A[]={44,33,11,55,77,90,40,60
           99,22,88,66};
    quick sort(A,12);
    for(i=0; i<=11; i++)
        printf("%d", A[i]);
}
```

Time Complexity

Time taken by Quick sort is general. Can be written as following:

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

The 1st two term are for two recursive calls, the last term is for the partition process. K is the number of elements which are smaller than pivot.

Worst case- The worst case occurs when the partition process always picks greatest or smallest element as pivot.

∴ for worst case

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

Best case- The best occurs when the partition process always picks the middle element as pivot.

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

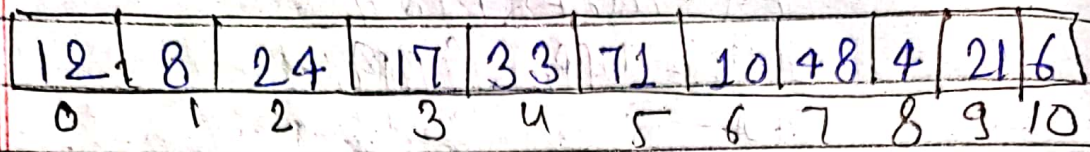
Average case-

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

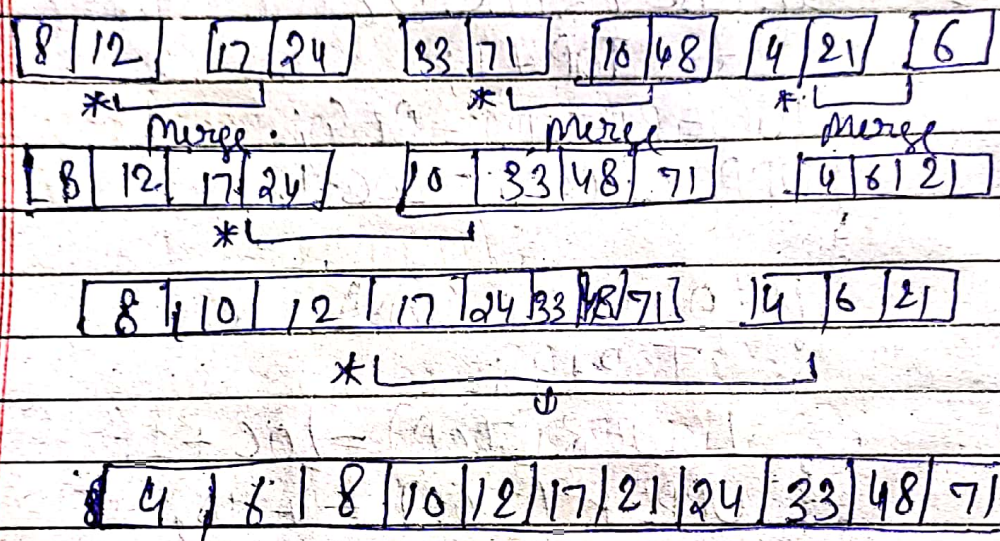
MERGE SORT

Page No.
Date / /

Logic-



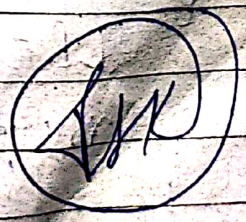
Ist merging & two element by making pair



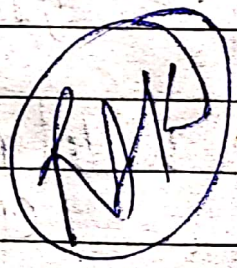
* → Merge and sort.

```
#include<stdio.h>
#include<stdlib.h>
void Merge(int [],int,int,int[],int,int,in[],int);
void merge sort(int[], int);
void mergepass(int[], int, int, int[]);
int main()
{
  int i, a[] = { 11,66,55,33,66,77,99,88,
                 22,44,584};
for(i=0, i<=10; i++)
    Printf("%d", a[i]);
merge_sort (a, 11);
```

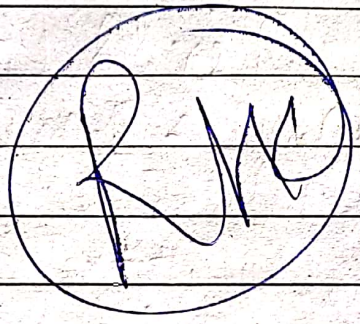
```
printf("\n");
for(i=0; i<=10; i++)
    Printf("%d", a[i]);
}
void merge (int A[], int n1, int index, int B[],
            int n2, int index2, int C[], int
            index)
{
  while(n1 && n2){
    if(A[index1] < B[index2]){
      C[index] = A[index1];
      index++;
      n1--;
    }
    else{
      C[index] = B[index2];
      index--;
      index2++;
      n2--;
    }
  } // end of while

  while(n1){
    C[index] = A[index1];
    index++;
    index1++;
    n1--;
  }
```



```
while(n2){
    c[indx] = B[indx2];
    indx++;
    indx2++;
    n2--;
  }
}
void merge_sort(int A[], int N)
{
  int L=1, B[11];
  while(L<N){
    mergepass(A, N, L, B);
    mergepass(B, N, 2*L, A);
    L++*L;
  }
}
void mergepass(int A[], int N, int L, int B[])
{
  int j, LB;
  int Q, S, R;
  Q = N/(2*L);
  S = 2*L*Q;
  R = N-S;
for(j=0; j<Q; j++)
{
  LB = (2*j)*L;
  merge(A, L, LB, A, L, LB+L, B, LB);
}
```



```
if (R <= L) {
    for (j=0 ; j < R ; j++) {
        B [S+j] = A [S+j];
    }
else
    {
        merge (A, L, S, A, R-L, S+L, B, S);
    }
}
```



Page No.
Date / / (KM2)

Time Complexity —

Merge sort is a recursive algo. and time complexity can be expressed as following recurrence relation.

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

The above recurrence can be solved either using recurrence tree method or Master method. It falls in case II of Master Method and soln of the recurrence is $\Theta(n \log n)$

Worst case - $n \log n$

Best case - $n \log n$

Average case - $n \log n$

As merge sort always divides the array into two halves and take linear time to merge two halves.

Page No.
Date / /

Insertion Sort

Defination- Number compare apne age wale se.

2	1	3	4

Here 4 compare with 2, 1, 3
3 " " 2, 1 and so on

t = 3

7	3	5	4	2	6
3	7	5	4	2	6

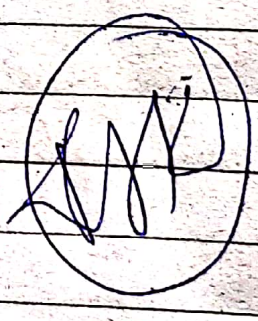
t = 5
t = 4

3	7	5	4	2	6
3	5	7	4	2	6

Don't put the t value in a[0] b/c if there is any (-) indexing then result will be wrong but here a[0] is start.

3	5	7	4	2	6
3	5	7	7	2	6
3	4	5	7	2	6

Logic-




```
for(i=1; i<n; i++)
{       i=1
  Key = a[i]
  j = i-1
  while (j>=0 && a[j]>Key)
    { a[j+1] = a[j];
      j = j-1; }
  a[j+1] = Key; }
```

Page No.
Date / /

Time Complexity

Time complexity is of $O(n^2)$

Boundary cases - Insertion sort take max^m time to sort if elements are sorted in reverse order. And it takes minimum time $(O(n))$ when elements are already sorted.