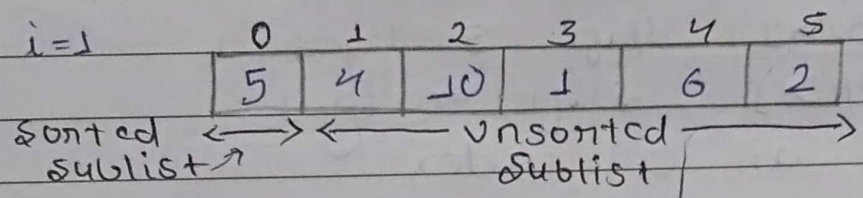
Insertion Sort Algorithm →

In this technique the given array is divided into two parts sorted sublist and unsorted sublist from left to right and take one item from unsorted sublist and going insert at correct location in sorted sublist.

Ex → i=1



n=6

temp
4

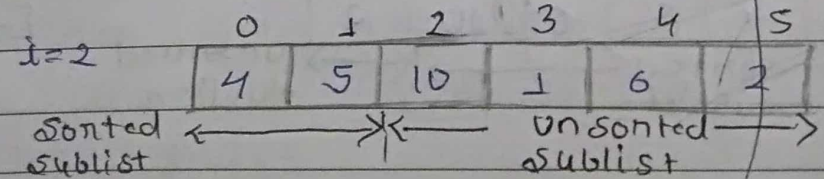
Here by default we are going to take first item in sorted sublist., Because in this way we have only one item this means that it is always sorted.

Round 1 → step 1) Copy 4 in temp

2) Compare 4 with 5 (elements in sorted sublist)

3) If 5 > 4,

Then shift 5 by 1 and copy temp = 4 at 0 index



temp
10

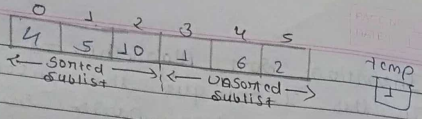
Round 2 → step 1) Copy 10 in temp.

2) Compare 5 with 10.

3) Here 5 is not greater than 10 so 5 will Not shift by 1

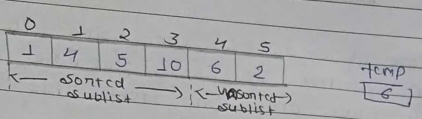
4) 10 will be at index 2

i=3



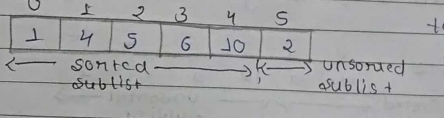
Round 03 → 1) Copy 1 in temp
2) Compare 1 with 10
3) Here 10 > 1, shift 10 by 1.
Now 1 will compare with each and every element in sorted sublist.
4) Here 5 > 1, shift 5 by 1
5) Here 4 > 1, shift 4 by 1
6) Copy 1 at 0 index

i=4



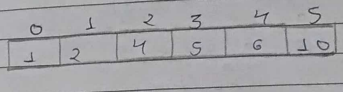
Round 4 → 1) Copy 6 in temp
2) Compare 6 with 10, Here 6 < 10, shift 10 by 1
3) Compare 6 with 5, Here 6 > 5, so,
4) Copy 6 at index 3

i = 5



Round 05 → 1) Copy 2 in temp
2) Compare 2 with 10
3) 10 > 2 → shift 10 by 1
4) 6 > 2 → shift 6 by 1
5) 5 > 2 → shift 5 by 1
6) 4 > 2 → shift 4 by 1
7) 1 is not greater than 2, copy 2 at index 1

Sorted array



Algorithm →

```
For (int i=1 ; i<n ; i++)  → loop for unsorted sublist
{
  int temp = arr[i] ;
  int j = i-1 ;
  while (j ≥ 0 && arr[j] > temp)  → loop for sorted sublist and comparison
  {
    arr[j+1] = arr[j] ;  → shift
    j--;
  }
  arr[j+1] = temp;  → copy temp at appropriate place.
}
```

Bubble sort Algorithm → In this sorting technique we compare two adjacent elements. If there elements are in correct order then its fine and if there elements are not in correct order then swaping take place.

We are going to arrange elements in ascending order.

Ex →

0	1	2	3	4
15	16	6	8	5

n = 5

15 < 16 (No swapping)

0	1	2	3	4
15	16	6	8	5

16 > 6 (swap)

Pass 1

0	1	2	3	4
15	6	16	8	5

i = 0

16 > 8 (swap)

0	1	2	3	4
15	6	8	16	5

16 > 5 (swap)

0	1	2	3	4
15	6	8	5	16

→ one element at correct

0	1	2	3	4
15	6	8	5	16

15 > 6 (swap)

Pass 2

6 15 8 5 16 — i = 1

15 > 8 (swap)

6 8 15 5 16

15 > 5 (swap)

Unnecessary comparision ← 6 8 5 15 16 → (two elements are at correct position)

15 < 16 (No swap)

6 8 5 15 16

No swap

Pass 3

6 8 5 15 16 — i = 2

i > 5 (swap)

Unnecessary comparision ← 6 5 8 15 16

i < 15 (No swap)

6 5 8 15 16 → three elements are at correct position

Pass 4

6 5 8 15 16

i > 6 (swap)

5 6 8 15 16

(No swap)

5 6 8 15 16

No swap

5 6 8 15 16 → 4 elements are at correct position

No swap

Here total 5 elements are there if 4 elements are at correct position then 5th one will be automatically at correct possition

No. of passes = (No. of elements) − 1

In above example in each pass No. of comparision is 4 ⇒ n−1

Algorithm →

```
For (int i=0 ; i<n-1 ; i++)   <-- loop for passes
{
  For (int j=0 ; j<n-1 ; j++)  <-- loop for comparisons
  {
    if (arr[j] > arr[j+1])
    {
      int temp = arr[j];
      arr[j] = arr[j+1];
      arr[j+1] = temp;
    }
  }
}
```

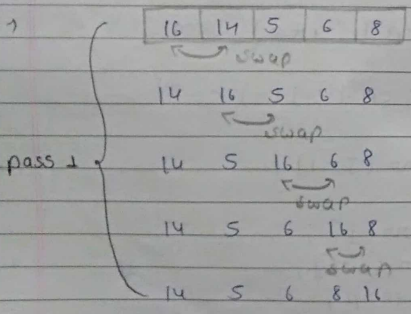
} swaping

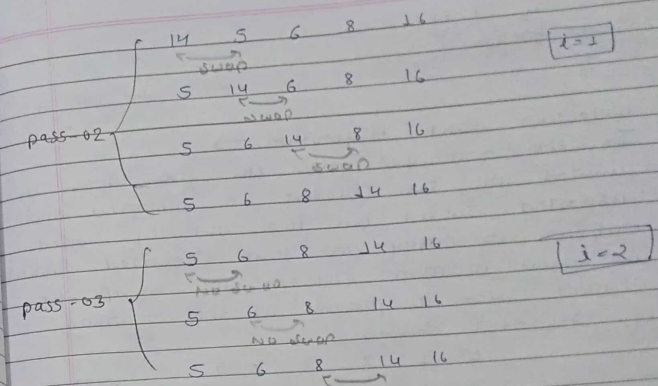
Improved algorithm →

```
For (int j=0 ; i<n-1 ; i++)
{
  For (int j=0 ; j<n-1-i ; j++)      comparision depend on i
  {
    if (arr[j] > arr[j+1])
    {
      int temp = arr[j];
      arr[j] = arr[j+1];
      arr[j+1] = temp;
    }
  }
}
```

* Suppose we have an array of size 10 and it is sorted after 3 passes but according to algorithm No. of passes = 9
Here we use flag variable

Ex →





If we get any pass in which No swapping is there then we can break our passing loop and array is sorted.

Optimized algorithm →

```
For (int i=0 ; i<n-1 ; i++)
{
  int flag=0;
  For (int j=0 ; j<n-1-j ; j++)
  {
    if (arr[j] > arr[j+1])
    {
      int temp = arr[j];
      arr[j] = arr[j+1];
      arr[j+1] = temp;
      flag ++;
    }
  }
}
```

```
if (flag == 0)
break;
}
```

Selection Sort Algorithm →

In this technique also the given array is divided into two sublist, i.e. sorted and unsorted sublist, but here initially sorted sublist is empty and from unsorted sublist we found minimum element and swap with element which is present at the starting of unsorted sublist.

Ex→

0	1	2	3	4	5
7	4	10	8	3	1

n = 6

sorted ← unsorted sublist → (min)

Pass-1

1	4	10	8	3	7

sorted ← unsorted sublist →

Pass-2

1	3	10	8	4	7

Pass-03

1	3	4	8	10	7

Pass-04)

1	3	4	7	10	8

Pass-05)

1	3	4	7	8	10

Algorithm →

```
For (int i=0 ; i<n-1 ; i++)        ⟶ for passes
{
  int min = i;
  For (j = i+1 ; j<n ; j++)        ⟶ for finding minimum element
  {
    if (arr[j] < arr[min])
    {
      min = j;
    }
  }
  if (min != i)
  {
    swap (a[i], a[min])            ⟶ swapping
  }
}
```

Quick Sort Algorithm →

In this type of sorting technique a key (pivot) element is chosen, pivot element may be starting element, last element or any random element.

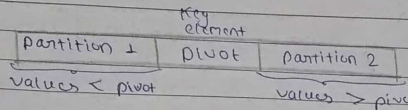
It is based on divide and conquer technique. In this technique complete array is divide into subarrays and dividing of arrays is called partitioning.

Ex→

0	1	2	3	4	5	6
10	15	1	2	9	16	11

↑ pivot element

partition take place in such a way that all elements less than pivot move to left side and all elements greater than pivot move to right side and element equal to pivot element move either left or right.



0	1	2	3	4	5	6
2	1	9	10	15	11	16

may be sorted or may not be — Appropriate place — may be sorted or may not be

2, 1, 9
1, 2, 9
9, 2, 1
} 0-2

16, 15, 11
11, 15, 16
16, 11, 15
} 4-6

same algorithm also apply for sub partition also

(Ex-)

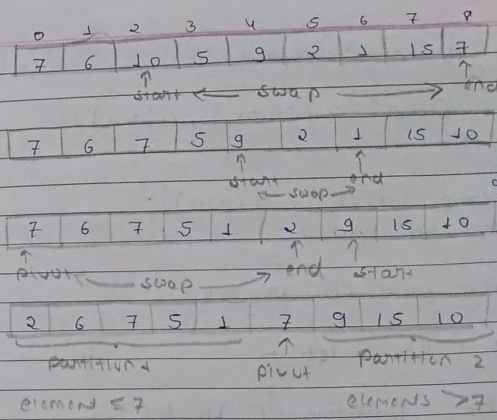
0	1	2	3	4	5	6	7	8
7	6	10	5	9	2	1	15	7

lower bound — upper bound

pivot = 7
a[0]

0	1	2	3	4	5	6	7	8
7	6	10	5	9	2	1	15	7

start → if item ≤ pivot start++;
← end if (item > pivot) end--



- same algo for these partition also

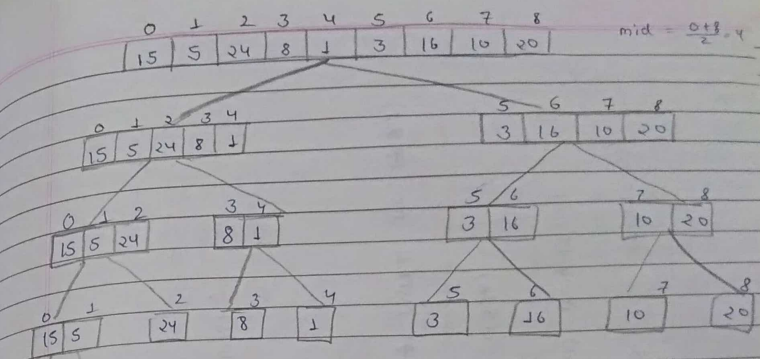
Algorithm →

```
partition (A, lb, ub)
{
  pivot = A[lb];
  start = lb;
  end = ub;
  while (start < end)
  {
    while (a[start] ≤ pivot)
    {
      start++;
    }
    while (a[end] > pivot)
    {
      end--;
    }
```

```
    if (start < end)
    {
      swap (a[start], a[end]);
    }
  }
  swap (a[lb] ; a[end]);
  return end;
}
Quick sort (A, lb, ub)
{
  if ( lb < ub )
  {
    loc = partition (A, lb, ub);
    QuickSort ( A, lb, loc-1);
    Quick sort ( A, loc+1, ub);
  }
}
```

Merge sort Algorithm →

This technique also work on the principle of divide and conquer rule in this case the complete list is divided into n sublist and each sublist contain 1 element. Now we merge the sublist and keep on merging each sublist to get a sorted list.



Now each sublist will have only one element.

Algorithm → Mergesort (A, lb, ub)

```
{
  if ( lb < ub )
  {
    mid = (lb + ub)/2 ;
    mergesort ( A, lb, mid);
    mergesort (A, mid+1, ub);
  }
  Merge (A, lb, mid, ub)
  {
    i = lb;
    j = mid+1;
    k = lb;
    while (i <= mid && j <= ub)
    {
      if (a[i] <= a[j])
      {
        b[k] = a[i];
        i++;
      }
      else
      {
        b[k] = a[j];
        j++;
```

Recursive call

→ function

```
if (i>mid)
{
while (j<=ub)
{
b[k] = a[j];
j++;
k++;
}
}
else
{
while (i<=mid)
{
b[k] = a[i];
i++;
k++;
}
}
For (k=lb; k<=ub; k++)
{
a[k] = b[k];
}
}
```

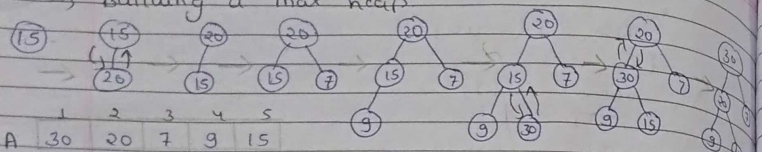
Heap Sorting → Heap is a tree based data structure. In Heap sorting we have to follow two steps →

1) To creat a heap or build a heap
2) deletion of elements from heap

After deletion of all data we find that data is sorted in increasing order

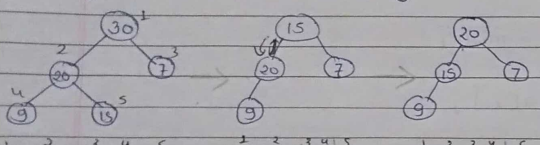
Ex → A | 15 | 20 | 7 | 9 | 30 |

→ Building a max heap



	1	2	3	4	5
A	30	20	7	9	15

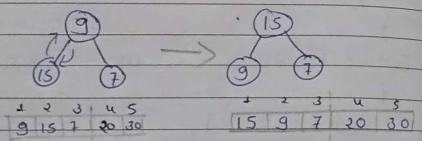
→ Deletion of element → root node element is deleted firstly and is replaced by most last element



* Here 30 will be deleted replace by 15

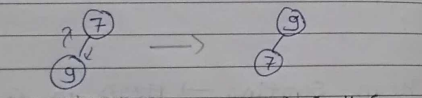
	1	2	3	4	5
	30	20	7	9	15
A	15	20	7	9	30
A	20	15	7	9	30

→ Now 20 will be deleted and replace by 9



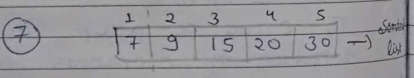
	1	2	3	4	5
	9	15	7	20	30
	15	9	7	20	30

→ Now 15 will be deleted and Replace by 7



	1	2	3	4	5
	7	9	15	20	30
	9	7	15	20	30

→ 9 will be deleted and replace by 7



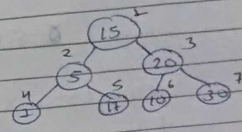
→ TC of heap sort = O(nlogn)

* Heapify method → Heapify is the process of creating a heap data structure from a binary tree. it is used to create a min or max heap

Ex → A | 15 | 5 | 20 | 1 | 17 | 10 | 30 |

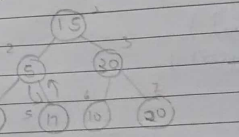
	1	2	3	4	5	6	7
A	15	5	20	1	17	10	30

index start from 1

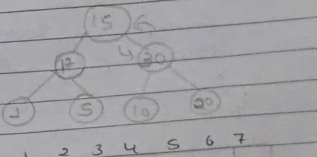


* Leaf node → $\left[\frac{N}{2}+1\right]$ to N nodes.

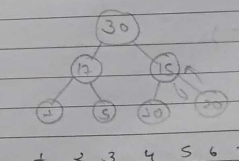
* Heapify method will start from $\frac{n}{2}$ non leaf nodes



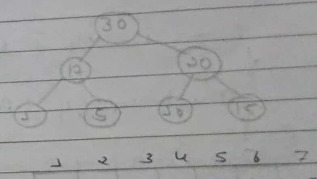
	1	2	3	4	5	6	7
	15	5	30	1	17	10	20



	1	2	3	4	5	6	7
	15	17	30	1	5	10	20



	1	2	3	4	5	6	7
	30	17	15	1	5	10	20



	1	2	3	4	5	6	7
	30	17	20	1	5	10	15

→ Algorithm →

```
Max heapify (A, n, i)
{
  int i = largest;
  int l = (2*i);
```

```
int r = (2*i)+1;
while (l <= n && a[l] > a[largest])
{
 largest = l;
}
while (r <= n && a[r] > a[largest])
{
 largest = r;
}
if (largest != i)
{
swap (a[largest], a[i])
maxheapify (A, n, largest)
}
}

HeapSort (A, n)
{
For (i = n/2; i >= 1; i--)
{
 maxheapify (A, n, i);
}
For (i=n; i >= 1; i--)
{
 swap (arr[1], arr[i])
 maxheapify (A, n-1)
}
}
```

building of heap

deletion

Radix sort → The idea of Radix sort is to do digit sort starting from least significant digit to most significant digit. Radix sort is non comparison based algorithm

Ex) 904
46
5
74
62
1

* firstly we have to place zeroe's in order to maintain three digits

→

	unsorted	arranged	sorted
904	001	001	001
046	062	904	005
005	904	005	046
074	074	046	062
062	005	062	074
001	046	074	904

* If two number having same least significant digit that arrange them in their significant order

Hashing → It is basically a searching technique with a constant time complexity of O(1) whereas in linear or binary search time complexity is O(n) and O(logn). It is basically a concept of storing and retrieving the data Here we ~~are~~ use Hash table, hashfunction and key

m=10 → hashtable size

$h(K) = K_i \% m$ → hash function (using division mod methods)

→ other method mid square, folding

Here Key = 6, 26

If we calculate the address value to store Key '6' using hashfunction then hash value = $6 \% 10 = 6$

store at index 6

0 to m-1	
0	
1	
2	
3	
4	
5	
6	6
7	
8	
9	

For 26, $26 \% 10 = 6$ → also store at index 6

∴ Collision occur

* To Resolve these collisions we have some techniques

Types of hasing →

1) open hashing (closed Addressing) → chaning
2) closed hashing (open Addressing)

- linear probing
- Quadratic Probing
- double hasting

1) open hasting - Closed Addressing [chaning]
linked list used to store data.

Ex)

Array[ ] → 3, 2, 9, 6, 11, 13, 7, 12

Used division method and closed addressing to store the values

$h(k) = 2k + 3$ $\quad m = 10$

Division method → $h(K) = K \% m$

0	
1	9
2	
3	
4	
5	6
6	
7	2
8	
9	3

11 | x

7 | 12 | x

13 | x

Key	location	probes
3	$[(2\times3)+3]\%10 = 9$	
2	$[(2\times2)+3]\%10 = 7$	
9	$[(2\times9)+3]\%10 = 1$	
6	$[(2\times6)+3]\%10 = 5$	
11	$[(2\times11)+3]\%10 = 5$	
13	$[(2\times13)+3]\%10 = 9$	
7	$[(2\times7)+3]\%10 = 7$	
12	$[(2\times12)+3]\%10 = 7$	

* linear probing → whenever open addressing is asked we always use linear probing by default.

→ same example for open addressing.

0	13
1	9
2	12
3	
4	
5	6
6	11
7	2
8	7
9	3

Key	location	probes
3	$[(2\times3)+3]\%10 = 9$	1
2	$[(2\times2)+3]\%10 = 7$	1
9	$[(2\times9)+3]\%10 = 1$	1
6	$[(2\times6)+3]\%10 = 5$	1
11	$[(2\times11)+3]\%10 = 5$	2
13	$[(2\times13)+3]\%10 = 9$	2
7	$[(2\times7)+3]\%10 = 7$	2
12	$[(2\times12)+3]\%10 = 7$	6

total probes = 16

Order of elements in hash table → 13, 9, 12, –, –, 6, 11, 2, 7, 3

linear probing → In case of collision insert K_i at first free location from $(u+1)\%m$ where $i=0$ to $m-1$

ICICI Bank Demat Services

ICICI Bank Limited, CPC-Demat Services, Ground Floor, B-Wing, Autumn Estate, Chandivali Farm Road, Opp. Mhada Colony, Chandivali, Andheri (East), Mumbai - 400072.
(Important Note : Please do not send requests directly to this address. Kindly route all your requests through the nearest ICICI Bank Demat Servicing Branches)

Application for Closure of Demat Account (NSDL/CDSL)

Date D D M M Y Y Y Y DP ID Client ID (of account to be closed)

I / We hereby request you to close my / our Demat account with you as per following details:

Sole/First Holder

Second Holder

Third Holder

*** Please tick the applicable option(s):** (*Marked is a Mandatory field)

☐ Option A (There are no balances / holdings in this account)

☐ Option B

(Transfer the balances / holdings in this account as per details given)

☐ Transfer to my / our own account (Provide target account details and enclose Client Master Report of Target Account)

☐ Transfer to any other account (Submit duly filled Delivery Instruction Slip signed by all holders)

Target Account Details

☐ NSDL — DP ID

☐ CDSL — Client ID

☐ Option C [Rematerialise / Reconvert (Submit duly filled Remat / Reconversion Request Form - For Mutual Fund units)]

*** Please tick the reason for closing the Demat Account:** (*Marked is a Mandatory field)

☐ Moving to new area/abroad where ICICI Bank does not have a branch
☐ High demat charges
☐ Consolidation of accounts
☐ Unsatisfactory services
☐ Stopped trading forever
☐ Others (Please specify) _____

Recovery of dues

☐ **Direct Debit**

Please debit my ICICI Bank Account(A/c No.) for recovery of any pending dues against my account

☐ **Cheque Payment**

Cheque Number..... drawn on Bank.....

☐ **Cash Payment** ☐ **UPI Payment** .....

Refund of charges

*Please provide operative Savings Bank account number for refund of pro rata AMC charges (if any). In case of Non ICICI Bank account, please provide a cancelled cheque copy.

☐ ICICI Bank Account ☐ Non ICICI Bank Account

Bank Account No.

DECLARATION: In case of Account Closure due to Shifting of Account:
I/We declare and confirm that all the transactions in my/our Demat Account are true/authentic.

V - April 2022

SIGNATURE OF SOLE/FIRST HOLDER | SIGNATURE OF SECOND HOLDER | SIGNATURE OF THIRD HOLDER

Acknowledgement

We hereby acknowledge the receipt of the your request for closing the below mentioned Demat Account subject to verification:

DP ID Client ID

Demat Account Holder's Name _____

Received by _____

Bank Official Signature _____ Branch _____ SR No _____

Date D D M M Y Y Y Y