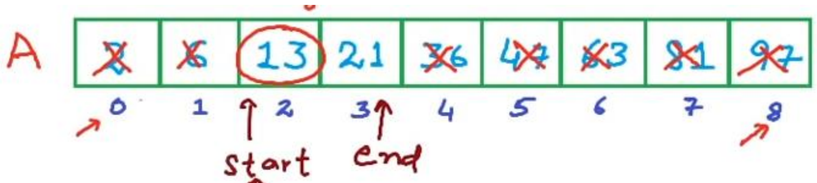
1. Linear search

```c
#include<stdio.h>

int main()
{

    int n=0,m=0,i;

    printf("Enter the number of elements u wanna sort: \n");
    scanf("%d",&n);

    printf("Enter the elements of unsorted array: \n");

    int arr[n];

    for(i=0;i<n;i++)
    {
        scanf("%d",&arr[i]);
    }

    printf("\n Elements of unsorted array are : \n");
    for(i=0;i<n;i++)
    {
        printf("%d",arr[i]);
    }

    printf("\n Enter the element u wanna search: ");
    scanf("%d",&m);



    for(i=0;i<n;i++)
    {
        if(m==arr[i])
        {
            printf("The element %d is found at %d index ",m,i);
        }
        else if(i==n-1)
        {
            printf("\n Not found\n");
        }
    }


}
```

2. Binary search



x = 13

Case 1 : x == A[mid]

Case 2 : x < A[mid]

Case 3 : x > A[mid]

```
BinarySearch (A, n, x)
{  start ← 0
   end ← n-1
while (start <= end)
  {  mid ← (start + end)/2
if A[mid] == x
     return mid
   else if x < A[mid]
   {  end ← mid - 1
   }
   else
   {  start ← mid + 1
   }
}
return -1
}
```

```c
int binary(int arr[],int n,int key)
{
    int low,high,mid;

    low=0,high=n-1;

    while(low<=high)
    {
        mid=(low+high)/2;

        if(arr[mid]==key)
        {
            return mid;
        }
        else if(key<arr[mid])
        {
            high=mid-1;
        }
        else
        {
            low=mid+1;
        }
    }
    return -1;
}
```

3. Bubble sort

A

1	2	3	4	5	7
0	1	2	3	4	5

2, 7, 4, 1, 5, 3

⇒ 2, 4, 1, 5, 3, 7

⇒ 2, 1, 4, 3, 5, 7

⇒ 1, 2, 3, 4, 5, 7

└ no change after Pass 4 & 5

```
BubbleSort (A,n)
{
    for k ← 1 to n-1
    {
        for i ← 0 to n-2
        {
            if (A[i] > A[i+1])
            {
                swap (A[i], A[i+1])
            }
        }
    }
}
```

F

$T(n) = (n-1) \times (n-1) \times c$

$= cn^2 - 2cn + 1$

$= O(n^2)$

Best-case : $O(n)$

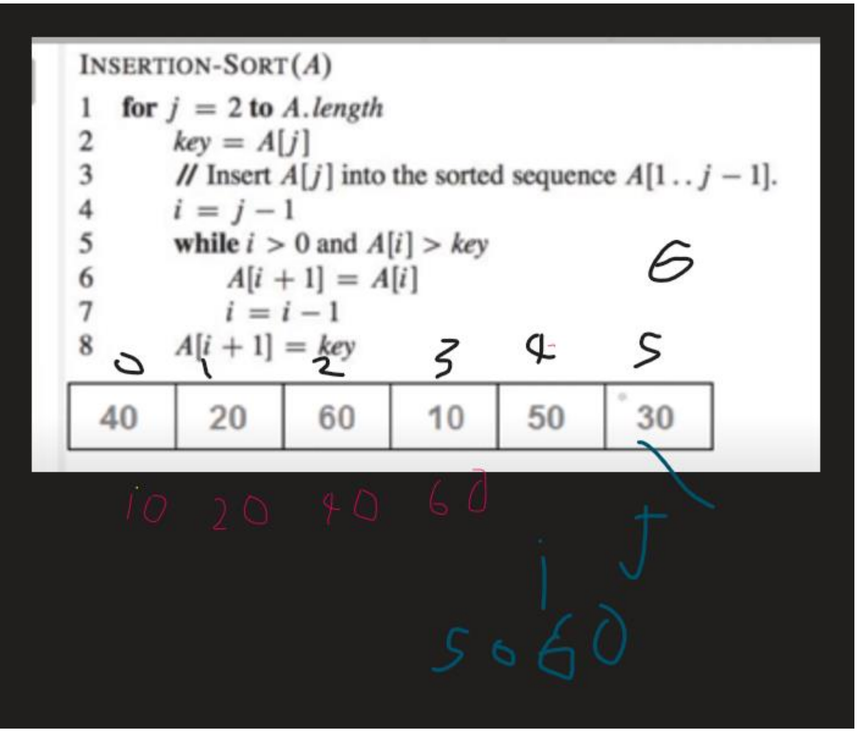
```
BubbleSort (A,n)
{
    for k ← 1 to n-1
    {
        flag ← 0
        for i ← 0 to n-k-1
        {
            if (A[i] > A[i+1])
            {
                swap (A[i], A[i+1])
                flag ← 1
            }
        }
        if(flag == 0) break
    }
}
```

```c
int bubble(int arr[],int n)
{
    int k,i;

    for(k=0;k<n;k++)
    {
        int flag=0;
        for(i=0;i<n-k-1;i++)
        {
            if(arr[i]>arr[i+1])
            {
                swap(&arr[i],&arr[i+1]);
            flag=1;
            }
        }
        if(flag==0)break;
    }
}
```

```c
int swap(int *x,int *y)
{
    int temp;
    temp=*x;
    *x=*y;
    *y=temp;
}
```

4. Insertion sort

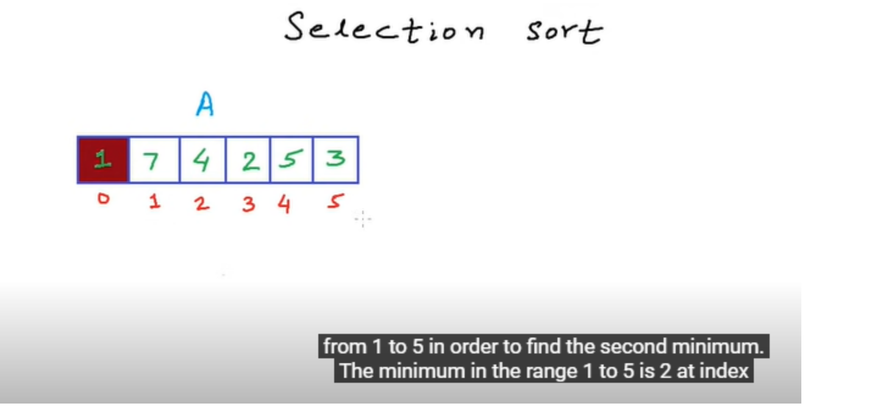
F

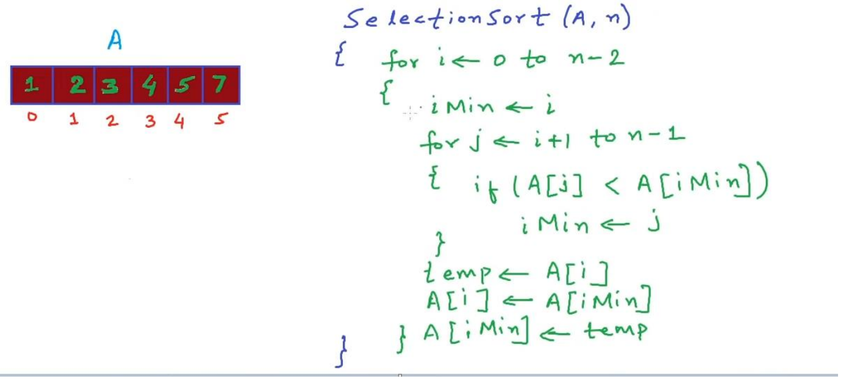



```c
int InsertionSort(int arr[],int n)
{
    int j,i;

    for(j=1;j<n;j++)
    {
        int key=arr[j];
        i=j-1;

        while(i>=0  &&  arr[i]>=key)
        {
            arr[i+1]=arr[i];
            i=i-1;
        }
        arr[i+1]=key;
    }
}
```

5. Selection sort

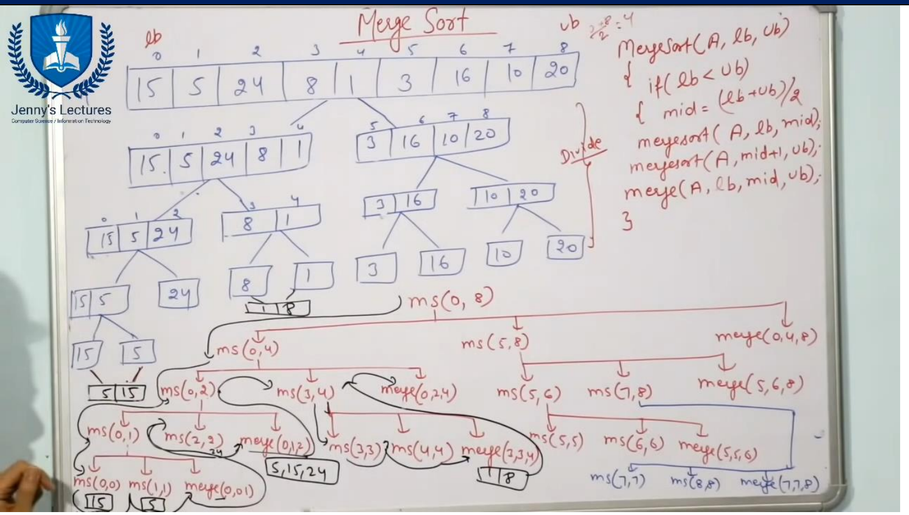
F

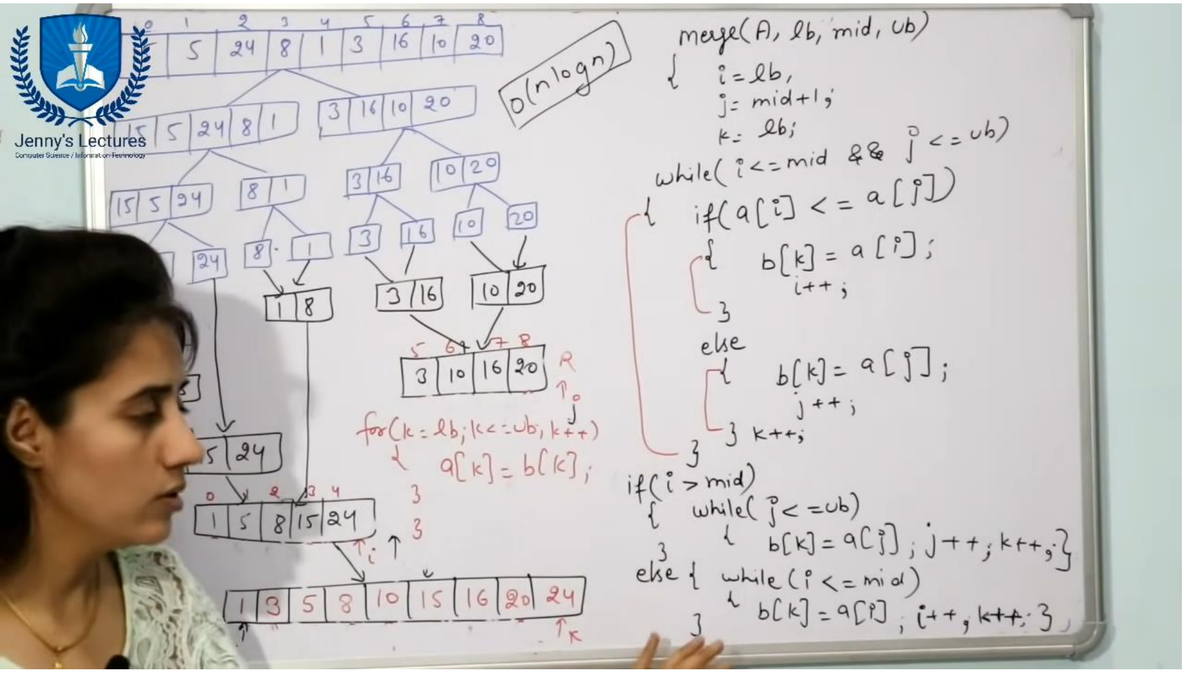





```c
int SelectionSort(int arr[],int n)
{
    int min,j,i;

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

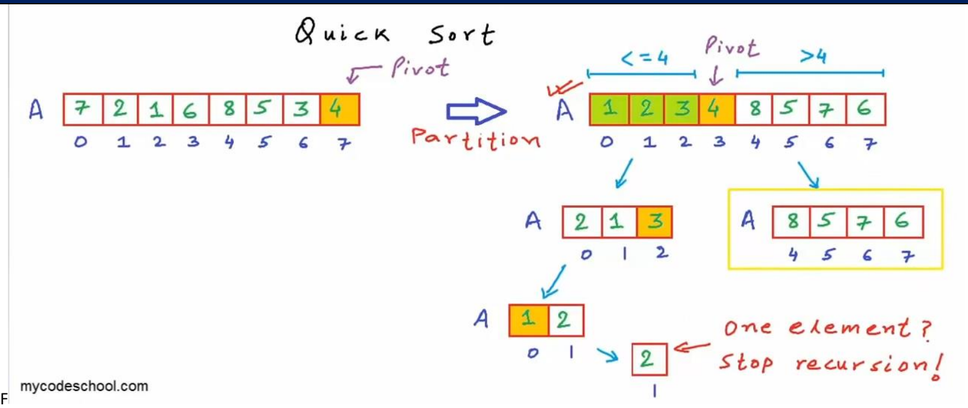
6. Merge sort

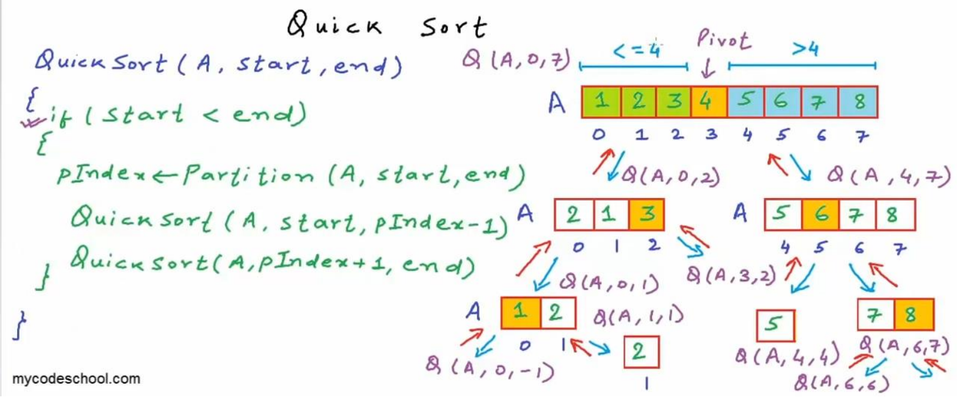


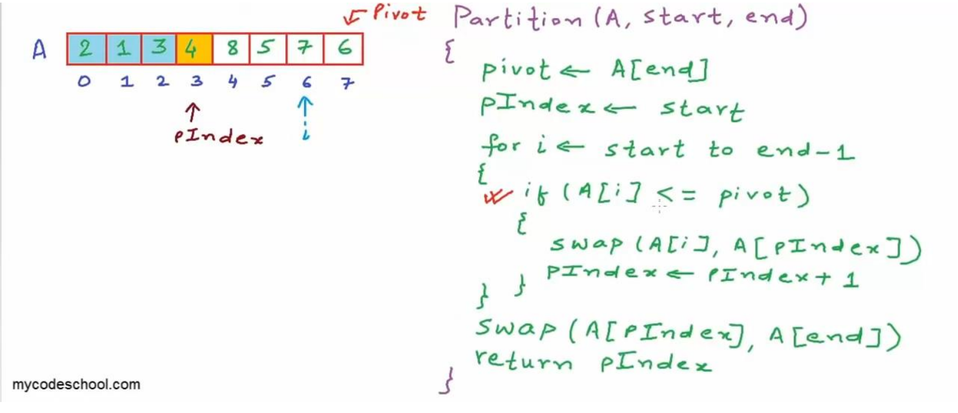



```c
int MergeSort(int arr[],int lb,int ub)
{
    int mid;

    if(lb<ub)
    {
        mid=(lb+ub)/2;
        MergeSort(arr,lb,mid);
        MergeSort(arr,mid+1,ub);
        Merge(arr,lb,mid,ub);
    }
}

int Merge(int a[],int lb,int mid,int ub)
{
    int i,j,k,b[100];
    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++;
        }k++;
    }

            while(i<=mid)
            {
                b[k]=a[i];
                i++;
                k++;
            }

            while(j<=ub)
            {
                b[k]=a[j];
                j++;
                k++;
            }

            for(k=lb;k<=ub;k++)
            {
                a[k]=b[k];
            }
}
```

7. Quick Sort

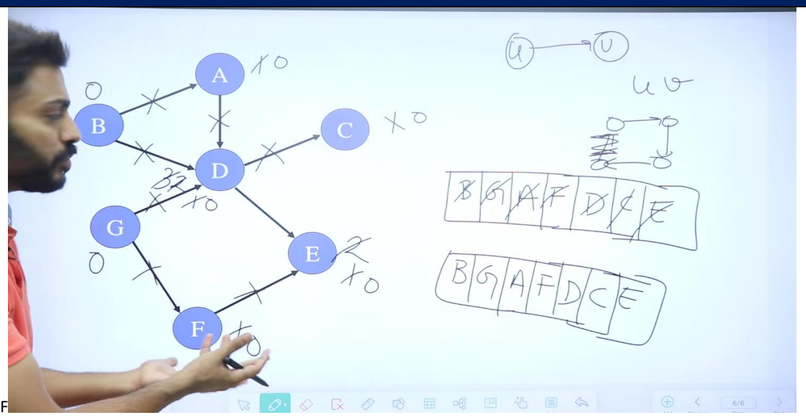




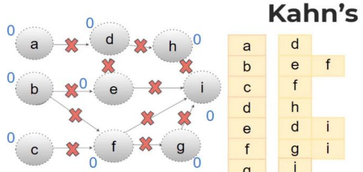



```c
int QuickSort(int arr[],int start,int end)
{

    if(start<end)
    {
        int pIndex=Partition(arr,start,end);
        QuickSort(arr,start,pIndex-1);
        QuickSort(arr,pIndex+1,end);
    }
}

int Partition(int a[],int start,int end)
{
    int i;

    int pivot=a[end];
    int pIndex=start;

    for(i=start;i<end;i++)
    {
        if(a[i]<pivot)
        {
            swap(&a[i],&a[pIndex]);
            pIndex++;
        }
    }
    swap(&a[pIndex],&a[end]);
    return pIndex;
}

int swap(int *a,int *b)
{
    int temp=*a;
    *a=*b;
    *b=temp;
}
```

8. Topological sort



Kahn's algorithm



Adjacency List

a	d
b	e f
c	f
d	h
e	d i
f	g i
g	i
h	i
i	

Topo order

a	b	c	e	f	d	g	h	i

Q

- Calculate indegree of all vertices
- Initialize visited count to 0
- Where to start? **Vertices with indegree 0**
- Add all vertices with indegree 0 to a Q
- DeQ a vertex and
  - Add it to Topo order list
  - Decrement indegree of its adjacent vertices by 1
  - If indegree of any vertex becomes 0, add it to Q
  - Increment visited count
- Stop when Q is empty
- If visited count equals total nodes, then print topo order
- If not, it indicates presence of cycle

- Calculate indegree of all vertices
- Initialize visited count to 0
- Where to start? **Vertices with indegree 0**
- Add all vertices with indegree 0 to a Q
- DeQ a vertex and
  - Add it to Topo order list
  - Decrement indegree of its adjacent vertices by 1
  - If indegree of any vertex becomes 0, add it to Q
  - Increment visited count
- Stop when Q is empty
- If visited count equals total nodes, then print topo order
- If not, it indicates presence of cycle

a	d
b	e f
c	f
d	h
e	d i
f	g i
g	i
h	i
i	

```
indegree[] = {0} * V
for each vertex v in graph G
      for each adj vertex av of v
            indegree[av]++;                    V+E
visited = 0

Q=[] , topo = []
for each vertex v in graph G
      If indegree[v] == 0
            add v to Q                         V

while Q not empty
      v ← deQ
      add v to topo
      for each adj vertex av of v
            indegree[av]--
            if indegree[av] == 0
                  add av to Q                  E
      Visited++

if visited != V
      error msg - cycle found - not  DAG
```

```c
#include <stdbool.h>
#define MAX_VERTICES 100

/* Structure for a vertex in a graph */
typedef struct {
    int in_degree;    /* Number of incoming edges */
    int out_degree;   /* Number of outgoing edges */
    int edges[MAX_VERTICES];  /* Adjacency list */
} vertex;

/* Graph structure */
typedef struct {
    vertex vertices[MAX_VERTICES];  /* Array of vertices */
    int num_vertices;   /* Number of vertices in the graph */
} graph;

/* Queue structure for storing vertices */
typedef struct {
    int vertices[MAX_VERTICES];  /* Array of vertices */
    int front, rear; /* Indexes of front and rear elements */
} queue;

/* Initialize an empty queue */
void init_queue(queue *q) {
    q->front = 0;
    q->rear = -1;
}

/* Check if the queue is empty */
bool is_queue_empty(queue *q) {
    return q->front > q->rear;
}

/* Add a vertex to the queue */
void enqueue(queue *q, int v) {
    q->vertices[++q->rear] = v;
}

/* Remove a vertex from the queue */
int dequeue(queue *q) {
    return q->vertices[q->front++];
}

/* Perform topological sorting on the given graph */
void topological_sort(graph *g, int *sorted_vertices) {
    queue q;
    init_queue(&q);

    /* Find all vertices with no incoming edges and add them to the queue */
    for (int i = 0; i < g->num_vertices; i++) {
        if (g->vertices[i].in_degree == 0) {
            enqueue(&q, i);
        }
    }

    int index = 0;
    while (!is_queue_empty(&q)) {
        /* Remove a vertex from the queue and add it to the sorted list */
        int v = dequeue(&q);
        sorted_vertices[index++] = v;

        /* Decrement the in-degree of each vertex in the adjacency list */
        for (int i = 0; i < g->vertices[v].out_degree; i++) {
            int w = g->vertices[v].edges[i];
            g->vertices[w].in_degree--;

            /* If a vertex's in-degree becomes 0, add it to the queue */
            if (g->vertices[w].in_degree == 0) {
                enqueue(&q, w);
            }
        }
    }
}
```

```c
int main(void) {
    /* Create the graph */
    graph g;
    g.num_vertices = 6;

    /* Set up the vertices and edges */
    g.vertices[0].in_degree = 1;
    g.vertices[0].out_degree = 1;
    g.vertices[0].edges[0] = 1;

    g.vertices[1].in_degree = 1;
    g.vertices[1].out_degree = 1;
    g.vertices[1].edges[0] = 2;

    g.vertices[2].in_degree = 1;
    g.vertices[2].out_degree = 1;
    g.vertices[2].edges[0] = 3;

    g.vertices[3].in_degree = 1;
    g.vertices[3].out_degree = 1;
    g.vertices[3].edges[0] = 4;

    g.vertices[4].in_degree = 2;
    g.vertices[4].out_degree = 0;
    g.vertices[4].edges[0] = 2;
    g.vertices[4].edges[1] = 5;

    g.vertices[5].in_degree = 2;
    g.vertices[5].out_degree = 0;
    g.vertices[5].edges[0] = 3;
    g.vertices[5].edges[1] = 5;

    /* Perform topological sorting */
    int sorted_vertices[g.num_vertices];
    topological_sort(&g, sorted_vertices);

    /* Print the sorted vertices */
    for (int i = 0; i < g.num_vertices; i++) {
        printf("%d ", sorted_vertices[i]);
    }
    printf("\n");

    return 0;
}
```

When you run this program, it should output the following:

Copy code

```
450123
```