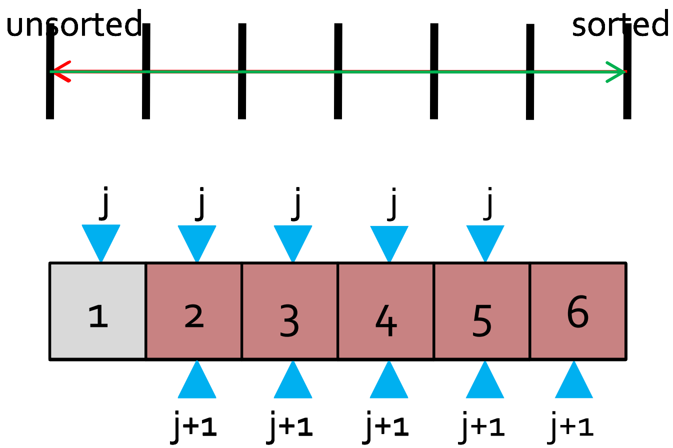
Sorting Searching

Department of CSE

Unit no 1
Sorting and searching
Data Structure
Algorithm

Om Suthar

Bubble Sort



Procedure bubble (T[1…n])

for i ← 1 **to** n-1 **do**

    for j ← 1 **to** n-i **do**

        if T[j] > T[j+1] **then**

            T[j] ↔ T[j+1]

        end if

    end for

end for

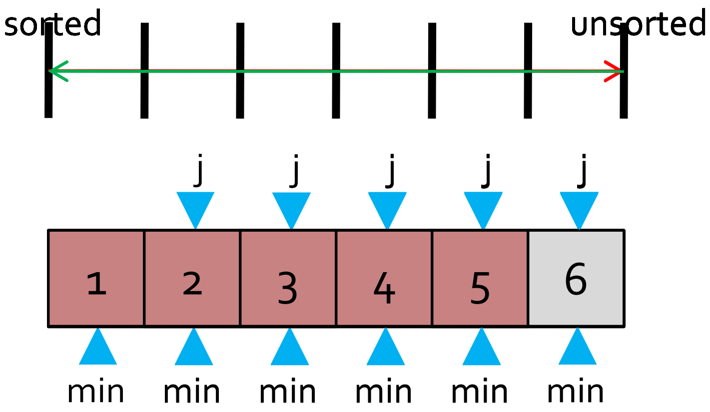
Bubble Sort

- Best Case:
  - $T(n) = O(n)$

- Worst Case:
  - $T(n) = O(n^2)$

- Average Case:
  - $T(n) = O(n^2)$

Selection Sort



Procedure selection (T[1…n])

for i ← 1 **to** n-1 **do**

min ← i

for j ← i+1 **to** n **do**

if T[j] < T[min] **then**

min ← j

end if

end for

T[min] ↔ T[i]

end for

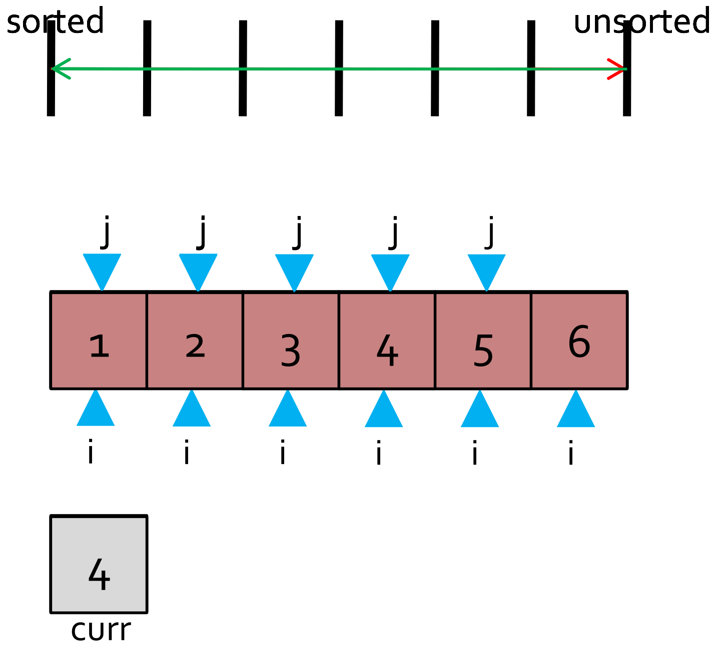
Selection Sort

- Best Case:
  - $T(n) = O(n^2)$

- Worst Case:
  - $T(n) = O(n^2)$

- Average Case:
  - $T(n) = O(n^2)$

Insertion Sort



Procedure insertion(T[1…n])

for i ← 1 **to** n **do**

curr ← T[i]

j ← i-1

while j >= 0 **and** T[j] > curr **do**

T[j+1] ← T[j]

j ← j-1

end while

T[j+1] ← curr;

end for

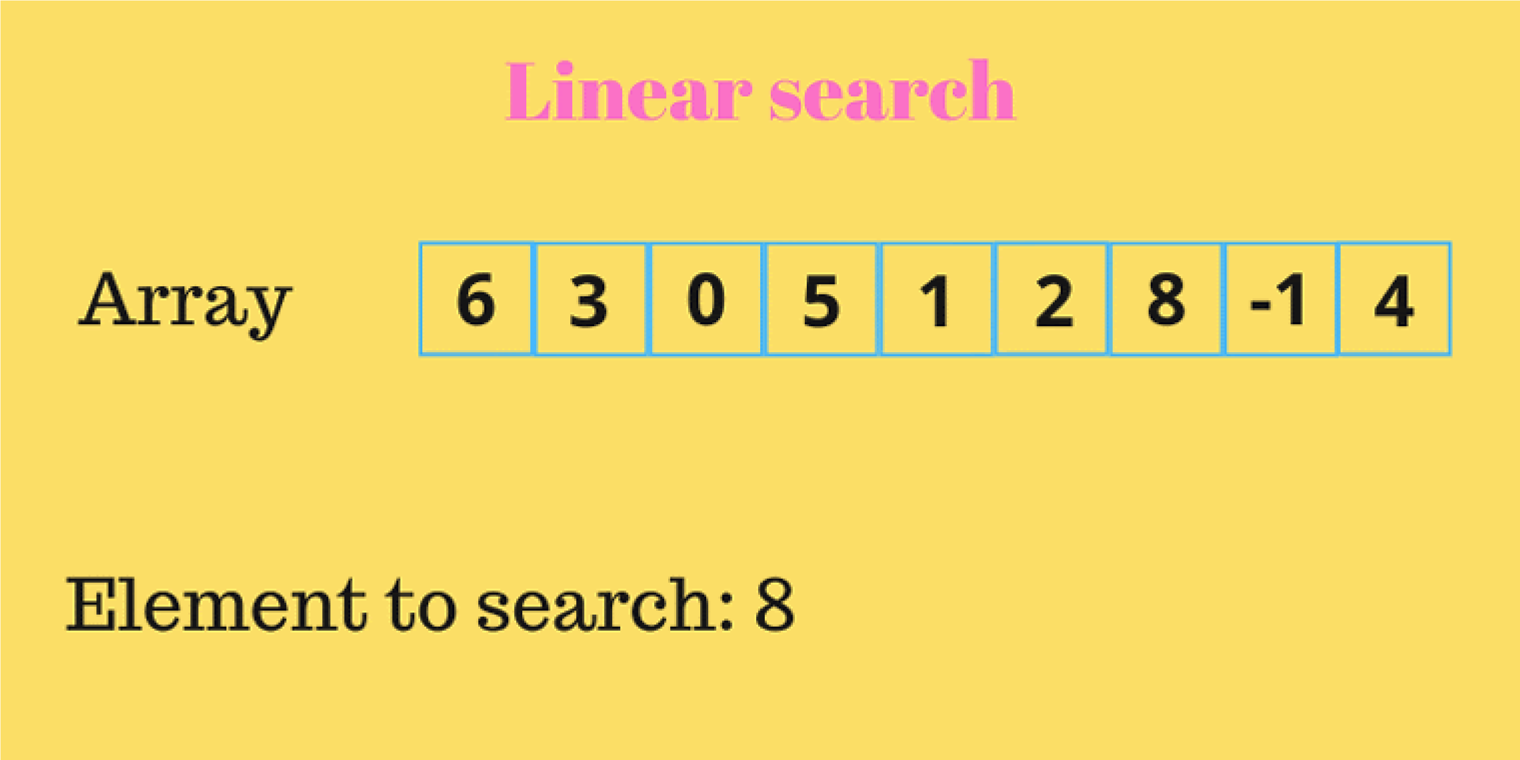
Insertion sort

- Best Case:
  - $T(n) = O(n)$
- Worst Case:
  - $T(n) = O(n^2)$
- Average Case:
  - $T(n) = O(n^2)$

Linear Search

Linear search in C to find whether a number is present in an array. If it's present, then at what location it occurs. It is also known as a **sequential search**.

we **compare** each element with the element **to search** until we find it or the list ends.



Linear Search pseudo code

```
for(i=0; i<n; i++)
{
        if(a[i] == data)
        {
                Printf("element found at location: %d", i);
                Break;
        }
}
If(i == n){
        printf("element not found");
}
```

Time Analysis

- Best Case:
- If you search for 6 and it is found at location 1 then time complexity O(1).
- Worst Case:
- If element is not in array then I loop will run for n elements. So time complexity is O(n).
- Average Case:
- Elements in random location.
- $\frac{\sum sum\ of\ all\ comparision(case)}{allcases}$
- 1+2+3+4....+n/ n
- =n(n+1)/2

Binary Search

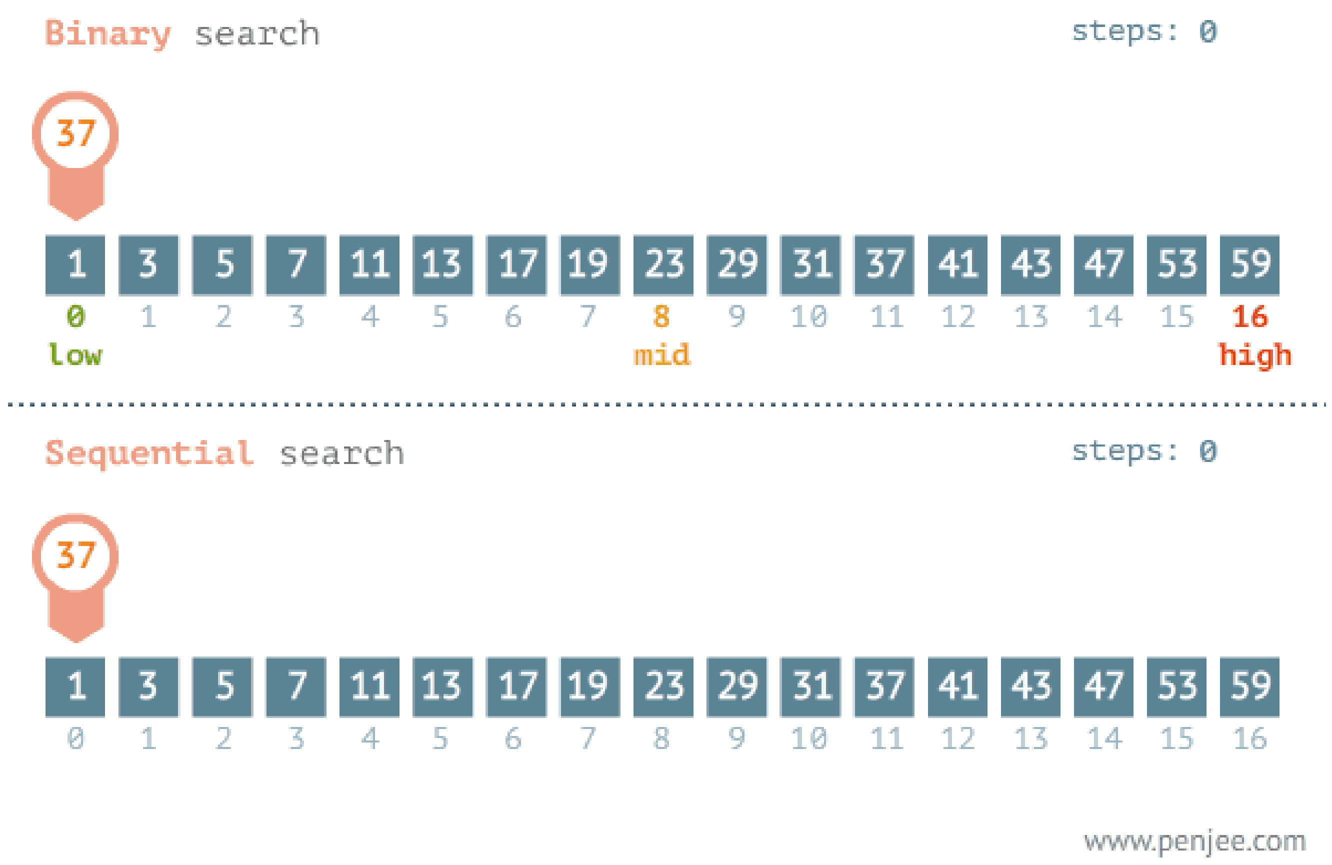
- Binary search will take **less** time than linear search.
- **Precondition:** Array **must be sorted.** If array is not sorted we can not apply algorithm.
- **Working Principle:**
- Search a sorted array by repeatedly dividing the search interval in half.
- We basically ignore half of the elements just after one comparison.

1. Compare **data** with the **middle** element.
2. If **data** matches with **middle** element, we return the **mid** index.
3. Else If **data > mid** element, then **data** can only lie in **right** half subarray after the **mid** element. So we trace for **right** half.
4. Else (**data** is smaller) trace for the **left half.**

3 Cases

- **Data == mid , return mid**
- **Data< mid , high = mid-1**
- **Data> mid, low= mid+1**

Binary Search



Binary Search Example

0	1	2	3	4	5	6	7	8	9
5	9	17	23	25	45	59	63	71	89
low									high

Data = 59

Low	High	mid
0	9	4

mid = low + high / 2

0	1	2	3	4	5	6	7	8	9
5	9	17	23	25	45	59	63	71	89
low				mid					high

Data = 59 a[mid]= a[4]= 25
data> a[mid] -------→ case 3
low= mid+1

Binary Search Example

0	1	2	3	4	5	6	7	8	9
5	9	17	23	25	45	59	63	71	89
					low				high

Data = 59

Low	High	mid
0	9	4
5	9	7

0	1	2	3	4	5	6	7	8	9
5	9	17	23	25	45	59	63	71	89
					low		mid		high

Data = 59 a[mid]= a[7]= 63
Data < a[mid] -------→ case 2
High = mid- 1

Binary Search Example

0	1	2	3	4	5	6	7	8	9
5	9	17	23	25	45	59	63	71	89
					low	high			

Data = 59

Low	High	mid
0	9	4
5	9	7
5	6	5

0	1	2	3	4	5	6	7	8	9
5	9	17	23	25	45	59	63	71	89
					low mid	high			

Data = 59 a[mid]= a[5]= 45
Data > a[mid] -------→ case 3
Low = mid+1

Binary Search Example

0	1	2	3	4	5	6	7	8	9
5	9	17	23	25	45	59	63	71	89
						low high			

Data = 59

Low	High	mid
0	9	4
5	9	7
5	6	5
6	6	6

0	1	2	3	4	5	6	7	8	9
5	9	17	23	25	45	59	63	71	89
						high mid low			

Data = 59 a[mid]= a[6]= 59
Data = a[mid] -------→ case 1
Return mid --→ 59 found.

Binary Search Example (if data is not found)

0	1	2	3	4	5	6	7	8	9
5	9	17	23	25	45	59	63	71	89
low									high

Data = 60

Low	High	mid
0	9	4

0	1	2	3	4	5	6	7	8	9
5	9	17	23	25	45	59	63	71	89
low				mid					high

Data = 60 a[mid]= a[4]= 25
data> a[mid] -------→ case 3
low= mid+1

Binary Search Example (if data is not found)

0	1	2	3	4	5	6	7	8	9
5	9	17	23	25	45	59	63	71	89
					low				high

Data = 60

Low	High	mid
0	9	4
5	9	7

0	1	2	3	4	5	6	7	8	9
5	9	17	23	25	45	59	63	71	89
					low		mid		high

Data = 60 a[mid]= a[7]= 63
Data < a[mid] -------→ case 2
High = mid-1

Binary Search Example (if data is not found)

0	1	2	3	4	5	6	7	8	9
5	9	17	23	25	45	59	63	71	89
					low	high			

Data = 60

Low	High	mid
0	9	4
5	9	7
5	6	5

0	1	2	3	4	5	6	7	8	9
5	9	17	23	25	45	59	63	71	89
					low mid	high			

Data = 60 a[mid]= a[5]= 45
data> a[mid] --------→ case 3
low= mid+1

Binary Search Example (if data is not found)

0	1	2	3	4	5	6	7	8	9
5	9	17	23	25	45	59	63	71	89

high
low

Data = 60

Low	High	mid
0	9	4
5	9	7
5	6	5
6	6	6

0	1	2	3	4	5	6	7	8	9
5	9	17	23	25	45	59	63	71	89

high
mid
low

Data = 60 a[mid]= a[6]= 59
data> a[mid] -------→ case 3
low= mid+1

Binary Search Example (if data is not found)

0	1	2	3	4	5	6	7	8	9
5	9	17	23	25	45	59	63	71	89
						high	low		

Data = 60

Low	High	mid
0	9	4
5	9	7
5	6	5
6	6	6
7	6	6

If (low> high) we should stop. And return -1. element is not found.

Algorithm for Binary Search

```
BINARY-SEARCH(A, low, high, data)
 if( low<= high){
  mid = floor((start+end)/2)
  if (A[mid]== data){
   return mid
}
  if (A[mid]>data){
   return BINARY-SEARCH(A, low, mid-1, data)
}
  if( A[mid]<data){
   return BINARY-SEARCH(A, mid+1, high, data)
}
 return FALSE // in case, element is not in the array
}
```

Time Complexity

- **O(logn) – Worst Case**
- **O(1)- Best Case.**

Thank you