CS2x1:Data Structures and Algorithms

Koteswararao Kondepu

k.kondepu@iitdh.ac.in

Recap: Sorting

```
MERGESORT (A, l, r)
 if (l < r)
    mid = l+r/2;
    MERGESORT(A, l, mid);
    MERGESORT(A, mid+ 1, r);
    MERGE (A, l, mid, r)
```

```
BUBBLESORT (A)
for i=1 → A.length -1
    for j=1 → A.length -i
        if A[j] > [j+1]
           exchange A[j] with A[j+1]
```

```
BUBBLESORT (A)
int swapped = 1
for i=1 → A.length -1
    swapped = 0
   for j=1 → A.length -i
       if A[j] > [j+1]
          exchange A[j] with A[j+1]
          swapped = 1
   if swapped == 0
        break;
```

```
INSERTIONSORT (A)
for j=2 → A.length
    key = A [j]
       // Insert A[j] into the
sorted sequence A[1→j -1]
   i = j - 1
   while i> 0 and A[i] > key
        A[i+1] = A[i]
          i = i - 1
   A[i+1] = key
```

```
SELECTIONSORT (A)
n = A.length
for i = 1 to n -1
    smallest = i;
    for j = i+1 to n
        if A[smallest] > A[j]
              smallest = j
    exchange A[i] with A[smallest]
```

Sorting Algorithms Comparison

Sorting Algorithm	Best	Average	Worst
Quick sort	$\Omega(n\ logn)$	$\theta(n\ logn)$	$\mathrm{O}\ (n^2)$
Merge sort	$\Omega(n\ logn)$	$\theta(n\ logn)$	$\mathrm{O}(n\ logn)$
Heap sort	$\Omega(n\ logn)$	$\theta(n\ logn)$	$\mathrm{O}(n\ logn)$
Bubble sort	$\Omega(\mathrm{n})$	$\theta(\mathrm{n}^2)$	$\mathrm{O}(\mathrm{n}^2)$
	$\Omega(n)$ $[Improved\ version\ with\ flag\ check]$		
Radix sort	$\Omega(dn)$	$\theta(dn)$	$\mathrm{O}(dn)$
Selection sort	$\Omega(n^2)$	$\theta(n^2)$	$\mathrm{O}(n^2)$
Insertion sort	$\Omega(n)$	$\theta(n^2)$	$\mathrm{O}(n^2)$

Exercise: Sorting (1)

Consider the following sequence of numbers:

23, 32, 45, 69, 72, 73, 89, 97

*Which of the following **sorting algorithm** uses the least number of comparisons to sort the above sequence of numbers in ascending order?*

*Which of the following **sorting algorithm** uses the most number of comparisons to sort the above sequence of numbers in ascending order?*

(a) Selection sort
(b) Insertion sort
(c) Bubble sort

Exercise: Sorting (2)

*If the **<u>selection sort algorithm</u>** is used to sort an array of length 5 in ascending order, and assuming that the array elements are initially arranged in the opposite order, what is the total number of comparisons required?*

(a) 1
(b) 10
(c) 15
(d) 20

Exercise: Sorting (3)

Consider the following sequence of numbers:

0, 1, 9, 8, 10, 7, 6, 11

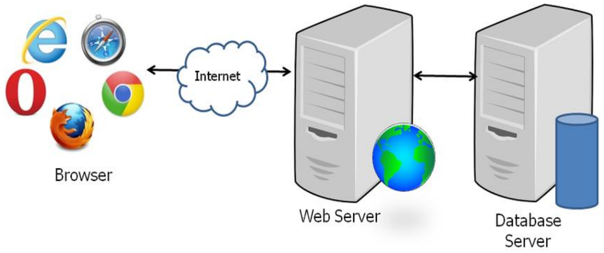
Use the ***Insertion sort algorithm*** *to arrange the sequence in ascending order.*

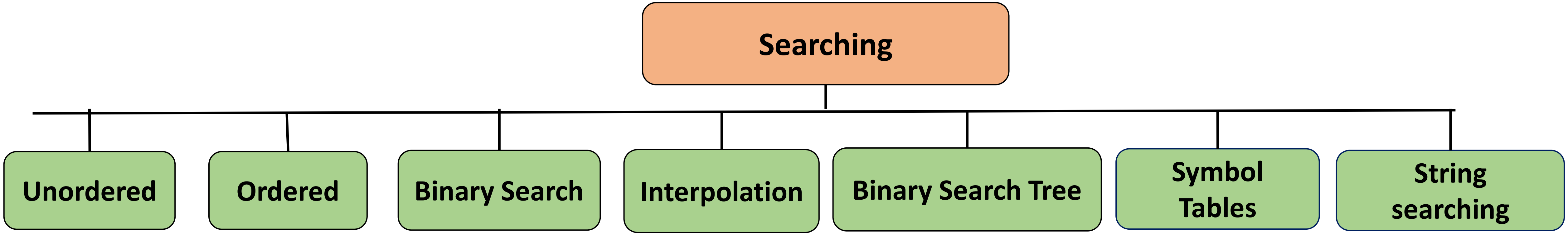
(i) What is the sequence of the elements after the end of the 6^{th} pass?

(ii) How many swaps are required to place "11" in the right place?

Searching (1)

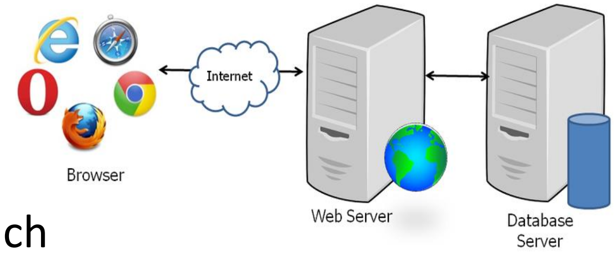
- Searching is the *__process of finding an item__* with specified properties from a collection of items
- If the data in proper order, it is easy to search the required element





Unordered Linear Search

- This falls under when *__the order of the elements is not known__* .
  - ✓ to search for an element, required to scan the complete array and see if the element is there in the given list or not



- Method:
  - ✓ I*__nput__*: Search the given element "*Key*" in the array
  - ✓ Start from the leftmost element from the array and one by one compare "*Key*" with each element of the array
  - ✓ If "*Key*" matches with an element, print the index
  - ✓ If "*Key*" doesn't match with any of the elements, return the *__exception message__*

```
LINEAR_SEARCH (A, Key)
for i = 1 to A.length
    if A[i] == Key
        print "Successful at location:" i
    else
        print "Unsuccessful"
```

Unordered Linear Search

```
LINEAR_SEARCH (A, Key)
for i = 1 to A.length
    if A[i] == Key
        print "Successful at location:" i
    else
        print "Unsuccessful"
```

Case 1: *Key* ***matches with the first element***

# of comparisons: 1

$$T(n) = 1$$

Key = 8

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

A[1] == *Key*

Case 2: *Key* ***does not exist***

# of comparisons: n

$$T(n) = n$$

Key = 12

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

Case 3: *Key* ***is present at any location in the given array***

# of comparisons: n+1/2

$$T(n) = n+1/2$$

Exercise: Unordered Linear Search

```
LINEAR_SEARCH (A, Key)
for i = 1 to A.length
    if A[i] == Key
        print "Successful at location:" i
    else
        print "Unsuccessful"
```

Consider that the given array contains 100 elements ranging from 1 to 100 in any order, how many comparisons are required to find** Key **is present at any location in the given array:

1 2 3 4 5 6 … 100

Ordered Linear Search

```
LINEAR_SEARCH_O (A, Key)
for i = 1 to A.length
    if A[i] == Key
        print "Successful at location:" i
    else if A[i] > Key
        print "Unsuccessful"
```

Key = 7

1	2	3	4	5	6	7
1	**2**	**3**	**4**	**5**	**6**	**7**

i = i+1

A[7] == *Key*

# of comparisons: 7

i = i+2

# of comparisons: 4

1	**2**	**3**	**4**	**5**	**6**	**7**

Binary Search (1)

- Search a sorted array by repeatedly dividing the search interval in half.
- Begin with an interval covering the whole array.
- If the value of the <u>*search key*</u> is less than the item in the *middle of the interval,* narrow the interval to the *lower half*, otherwise narrow it to the *upper half*.
- Repeatedly check until the value is found or the interval is empty.

- Method:
  - ✓ Compare *key* with the middle element.
  - ✓ If *key* matches with the middle element, print/return the mid index.
  - ✓ Else If *key* is greater than the mid element, then *key* can only in the right half sub array.
  - ✓ Else (*key* is smaller) recur for the left half.

Binary Search (2)

```
BINARY_SEARCH (A, Key, 1, n)
l=1; r = n
while (l ≤ r)
   m = floor((l+r)/2)
   if (Key < A[m])
       r = m - 1
   else if (Key > A [m])
       l = m + 1
    else if (Key == A[m])
       print "Successful at location:" m
    else if
       print "Unsuccessful"
```

$$T(n) = T\left(\frac{n}{2}\right) + \Theta(1)$$

a = 1; b = 2, k=0, p=0

Case 2.a a= b^k → If p > -1, then $T(n) = \Theta(n^{\log_b^a} \log^{p+1} n)$

$= O(\log n)$

Key = 9

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

m = (1+7)/2 = 4

Key < A[4]

Key > A[4]

l = m+1= 4+1 = 5

1	**2**	**3**	**4**	**6**	**8**	**9**

m = (5+7)/2 = 6

Key < A[6]

Key > A[6]

l = m+1= 6+1 = 7

1	**2**	**3**	**4**	**6**	**8**	**9**

m = (7+7)/2 = 7

Key < A[7]

Key > A[7]

Key == A[7]

print "Successful at location:" 7

Interpolation Search

```
BINARY_SEARCH (A, Key, 1, n)
l=1; r = n
while (l ≤ r)
   m = l+((Key−A[l])∗(r−l) / (A[r]−A[l]))
   if (Key < A[m])
      r = m - 1
   else if (Key > A [m])
      l = m + 1
   else if (Key == A[m])
      print "Successful at location:" m
   else if
      print "Unsuccessful"
```

$$m = l + \left(\left(\frac{(Key - A[l]) * (r - l)}{A[r] - A[l]}\right)\right)$$

Time complexity = O $(log log n)$

Key = 9

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

m = 1+(((9-1)*(7-1))/9-1) = 7

Key > A[4]

l = m+1= 4+1 = 5

1	**2**	**3**	**4**	**6**	**8**	**9**

m = (5+7)/2 = 6

Key < A[6]

Key > A[6]

l = m+1= 6+1 = 7

1	**2**	**3**	**4**	**6**	**8**	**9**

m = (7+7)/2 = 7

Key < A[7]

Key > A[7]

Key == A[7]

print "Successful at location:" 7

List of Topics [C201]

- Introduction:
  - *Data structures*
  - *Abstract data types*
  - *Analysis of algorithms.*

- Creation and manipulation of data structures:
  - *Arrays; Stacks; Queues; Linked lists; Trees; Heaps; Hash tables; Balanced trees [AVL]*; *Graphs*.

- *Algorithms for sorting and searching*, *depth-first and breadth-first search, shortest paths and minimum spanning tree*.

thank you!

email:
k.kondepu@iitdh.ac.in