Searching Algorithms:

Linear Search, Binary Search

Sorting Algorithms:

Insertion Sort, Bubble Sort, Selection Sort

Dr. Md. Abul Kashem Mia, Professor, CSE Dept, BUET

The Searching Problem

- The process of finding a particular element in an array is called searching. There two popular searching techniques:
  - *Linear search,* and
  - *Binary search*.
- The ***linear search*** compares each array element with the ***search key***.
- If the ***search key*** is a member of the array, typically the location of the search key is reported to indicate the presence of the search key in the array. Otherwise, a *sentinel* value is reported to indicate the absence of the search key in the array.

Linear Search

- Each member of the array is visited until the search key is found.

- **Example:**

  Write a program to search for the search key entered by the user in the following array:

  (9, 4, 5, 1, 7, 78, 22, 15, 96, 45)

  You can use the linear search in this example.

Linear Search

```
/* This program is an example of the Linear Search*/
#include <stdio.h>
#define SIZE 10
int LinearSearch(int [], int);
int main() {
    int a[SIZE]= {9, 4, 5, 1, 7, 78, 22, 15, 96, 45};
    int key, pos;
    printf("Enter the Search Key\n");
    scanf("%d", &key);
    pos = LinearSearch(a, key);
    if(pos == -1)
        printf("The search key is not in the array\n");
    else
        printf("The search key %d is at location %d\n", key, pos);
    return 0;
}
```

Linear Search

```
int LinearSearch (int b[ ], int skey) {
    int i;
    for (i=0; i < SIZE; i++)
        if(b[i] == skey)
            return i;
    return –1;
}
```

Binary Search

- Given a sorted array, ***Binary Search*** algorithm can be used to perform fast searching of a search key on the sorted array.
- The following program uses pointer notation to implement the binary search algorithm for the search key entered by the user in the following array:

(3, 5, 9, 11, 15, 17, 22, 25, 37, 68)

Binary Search

```
#include <stdio.h>
#define SIZE 10
int BinarySearch(int [ ], int);
int main(){
    int a[SIZE]= {3, 5, 9, 11, 15, 17, 22, 25, 37, 68};
    int key, pos;
    printf("Enter the Search Key\n");
    scanf("%d",&key);
    pos = BinarySearch(a, key);
    if(pos == –1)
        printf("The search key is not in the array\n");
    else
        printf("The search key %d is at location %d\n", key, pos);
    return 0;
}
```

Binary Search

```
int BinarySearch (int A[], int skey){
    int low=0, high=SIZE−1, middle;
    while(low <= high){
        middle = (low+high)/2;
        if (skey == A[middle])
            return middle;
        else if(skey <A[middle])
            high = middle – 1;
        else
            low = middle + 1;
    }
    return –1;
}
```

The Sorting Problem

- **Input:**
  - A sequence of n numbers $a_1, a_2, . . . , a_n$
- **Output:**
  - A permutation (reordering) $a_1', a_2', . . . , a_n'$ of the input sequence such that $a_1' \leq a_2' \leq \cdots \leq a_n'$

Why Study Sorting Algorithms?

- There are a variety of situations that we can encounter
  - Do we have randomly ordered keys?
  - Are all keys distinct?
  - How large is the set of keys to be ordered?
  - Need guaranteed performance?

- Various algorithms are better suited to some of these situations

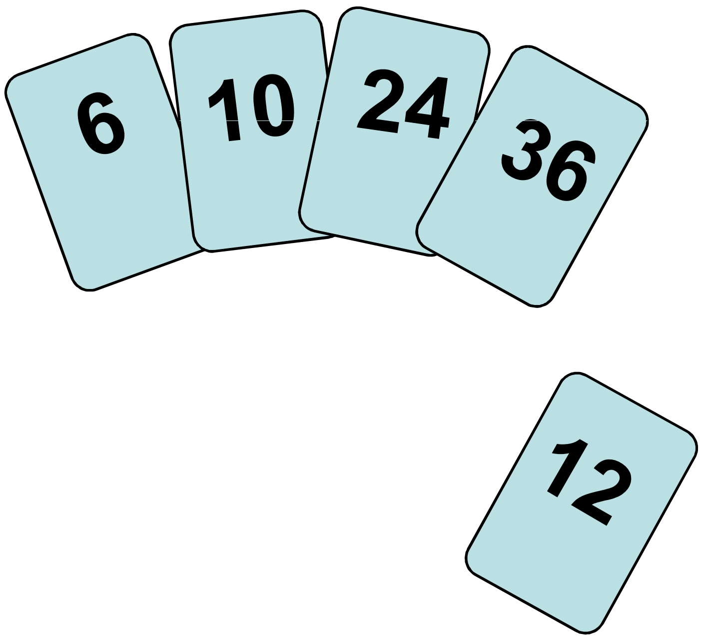
Some Definitions

- Internal Sort
  - The data to be sorted is all stored in the computer's main memory.
- External Sort
  - Some of the data to be sorted might be stored in some external, slower, device.
- In Place Sort
  - The amount of extra space required to sort the data is constant with the input size.

Insertion Sort

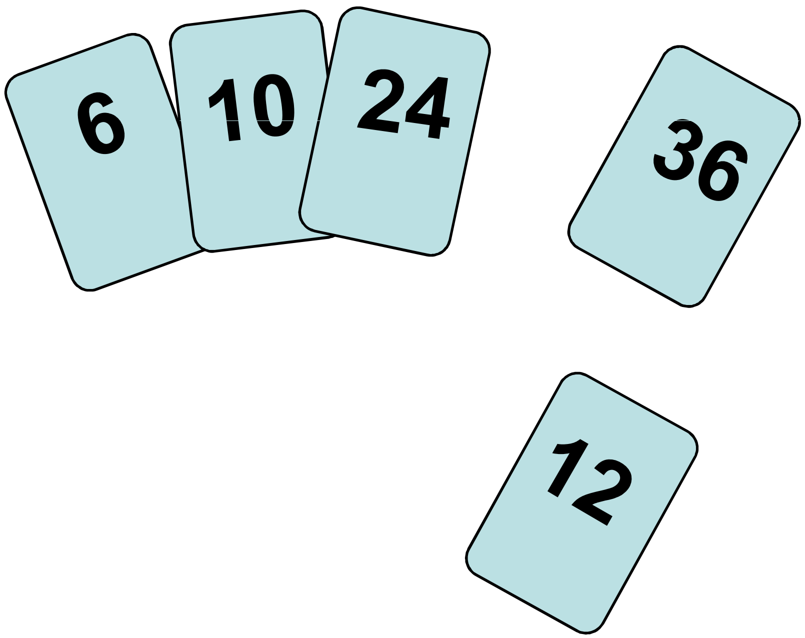
- Idea: like sorting a hand of playing cards
  - Start with an empty left hand and the cards facing down on the table.
  - Remove one card at a time from the table, and insert it into the correct position in the left hand
    - compare it with each of the cards already in the hand, from right to left
  - The cards held in the left hand are sorted
    - these cards were originally the top cards of the pile on the table

Insertion Sort



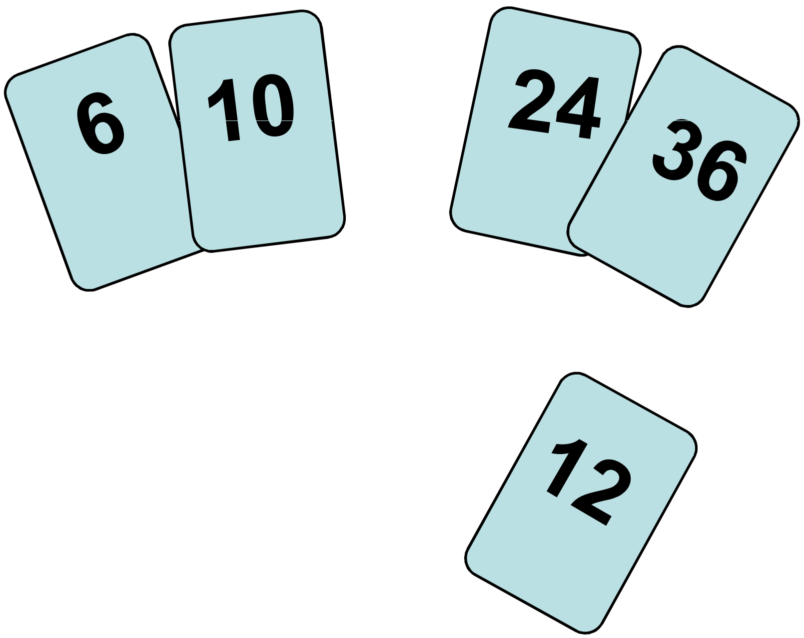
To insert 12, we need to make room for it by moving first 36 and then 24.

Insertion Sort



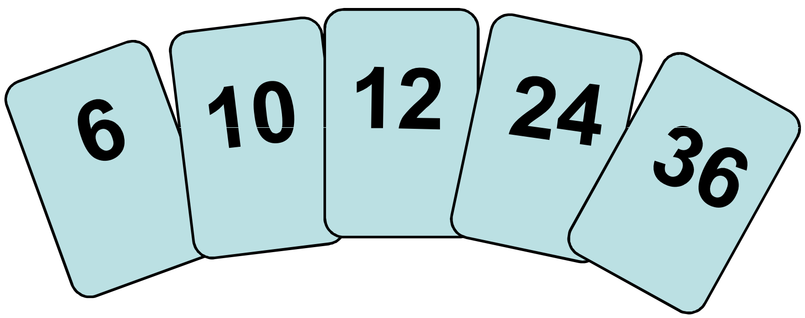
To insert 12, we need to make room for it by moving first 36 and then 24.

Insertion Sort



To insert 12, we need to make room for it by moving first 36 and then 24.

Insertion Sort



To insert 12, we need to make room for it by moving first 36 and then 24.

Insertion Sort

input array

5 2 4 6 1 3

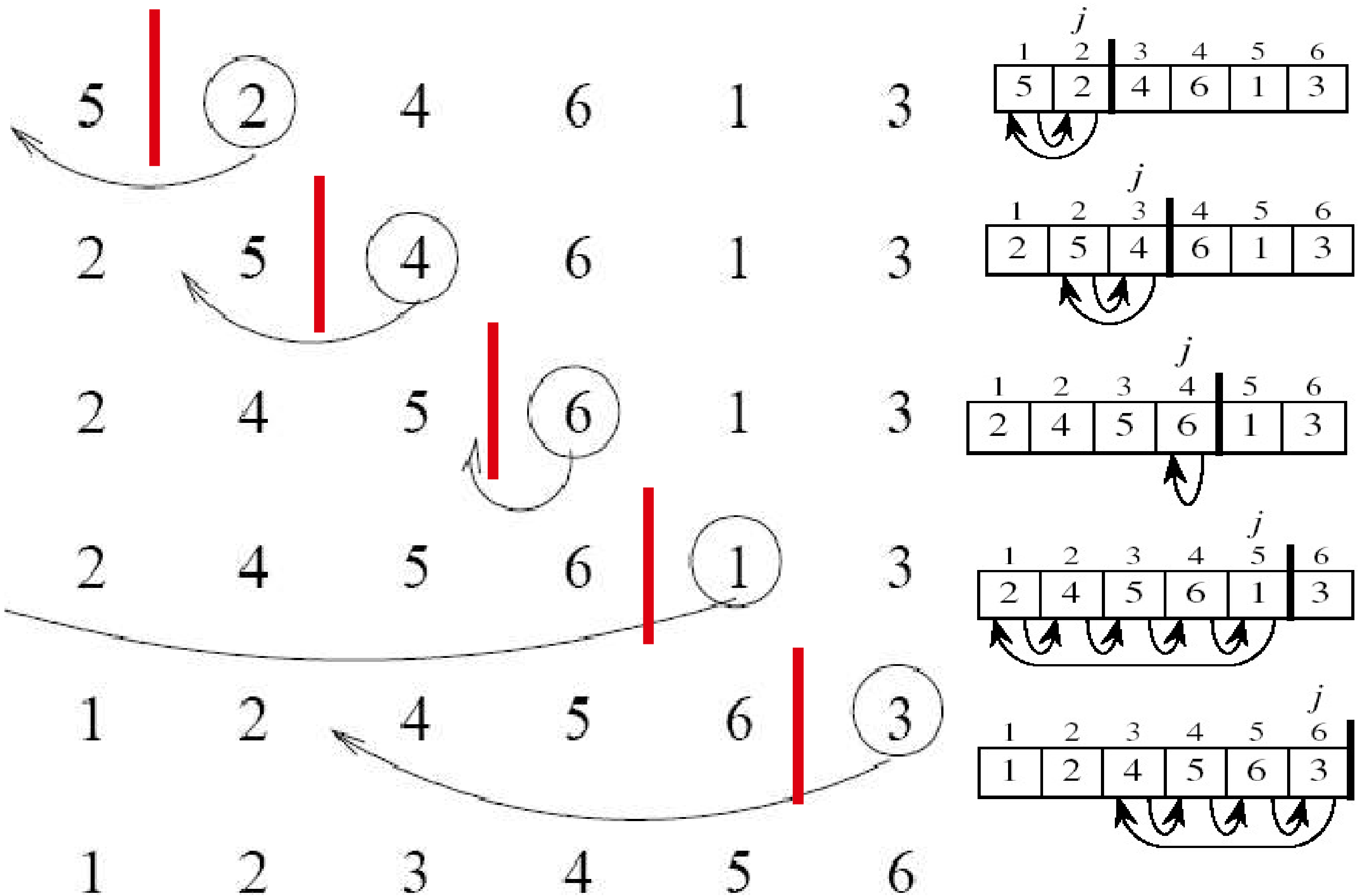
at each iteration, the array is divided in two sub-arrays:

left sub-array *right sub-array*

2 5 | 4 6 1 3

sorted *unsorted*

Insertion Sort



Insertion Sort

1	2	3	4	5	6	7	8
a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8

key

```
Alg.: INSERTION-SORT(A)
  for j ← 2 to n
      do key ← A[ j ]
         ▷Insert A[ j ] into the sorted sequence A[1 . . j -1]
         i ← j - 1
         while i > 0 and A[i] > key
             do A[i + 1] ← A[i]
                i ← i - 1
         A[i + 1] ← key
```

- Insertion sort – sorts the elements in place

Analysis of Insertion Sort

INSERTION-SORT*(A)*	cost	times
for j ← 2 **to** n	c_1	n
do key ← A[j]	c_2	$n-1$
▷Insert A[j] into the sorted sequence A[1 . . j -1]	0	$n-1$
i ← j - 1	c_4	$n-1$
while i > 0 and A[i] > key	c_5	$\sum_{j=2}^{n} t_j$
do A[i + 1] ← A[i]	c_6	$\sum_{j=2}^{n} (t_j - 1)$
i ← i – 1	c_7	$\sum_{j=2}^{n} (t_j - 1)$
A[i + 1] ← key	c_8	$n-1$

t_j: # of times the while statement is executed at iteration j

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^{n} t_j + c_6 \sum_{j=2}^{n} \left(t_j - 1\right) + c_7 \sum_{j=2}^{n} \left(t_j - 1\right) + c_8(n-1)$$

Best Case Analysis

- The array is already sorted ***"while** i > 0 and A[i] > key"*
  - A[i] ≤ key upon the first time the **while** loop test is run (when $i = j - 1$)
  - $t_j = 1$
- $T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5(n-1) + c_8(n-1)$

$= (c_1 + c_2 + c_4 + c_5 + c_8)n + (c_2 + c_4 + c_5 + c_8)$

$= an + b = \Theta(n)$

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^{n} t_j + c_6 \sum_{j=2}^{n} \left(t_j - 1\right) + c_7 \sum_{j=2}^{n} \left(t_j - 1\right) + c_8(n-1)$$

Dr. Md. Abul Kashem Mia, Professor, CSE Dept, BUET

Worst Case Analysis

- The array is in reverse sorted order ***"while** i > 0 and A[i] > key"*
  - Always A[i] > key in **while** loop test
  - Have to compare key with all elements to the left of the j-th position $\Rightarrow$ compare with j-1 elements $\Rightarrow$ t_j = j

using $\sum_{j=1}^{n} j = \frac{n(n+1)}{2}$ => $\sum_{j=2}^{n} j = \frac{n(n+1)}{2} - 1$ => $\sum_{j=2}^{n}(j-1) = \frac{n(n-1)}{2}$ *we have:*

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5\left(\frac{n(n+1)}{2} - 1\right) + c_6\frac{n(n-1)}{2} + c_7\frac{n(n-1)}{2} + c_8(n-1)$$

$$= an^2 + bn + c$$ a quadratic function of n

- T(n) = $\Theta(n^2)$ order of growth in n^2

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5\sum_{j=2}^{n} t_j + c_6\sum_{j=2}^{n}(t_j - 1) + c_7\sum_{j=2}^{n}(t_j - 1) + c_8(n-1)$$

Comparisons and Exchanges in Insertion Sort

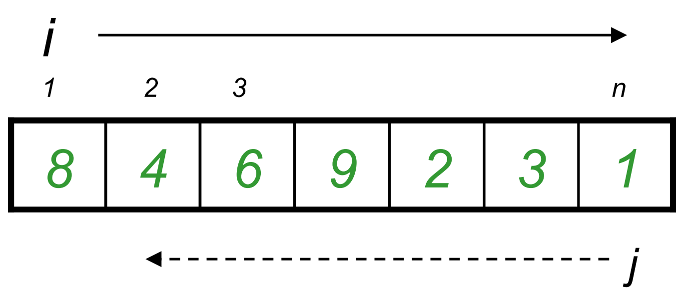
INSERTION-SORT*(A)*		cost	times
for j ← 2 **to** n		c_1	n
do key ← A[j]		c_2	$n-1$
Insert A[j] into the sorted sequence A[1 . . j -1]		0	$n-1$
i ← j - 1	$\approx n^2/2$ comparisons	c_4	$n-1$
while i > 0 and A[i] > key		c_5	$\sum_{j=2}^{n} t_j$
do A[i + 1] ← A[i]		c_6	$\sum_{j=2}^{n} (t_j - 1)$
i ← i – 1	$\approx n^2/2$ exchanges	c_7	$\sum_{j=2}^{n} (t_j - 1)$
A[i + 1] ← key		c_8	$n-1$

Insertion Sort - Summary

- Advantages
  - Good running time for "almost sorted" arrays $\Theta(n)$
- Disadvantages
  - $\Theta(n^2)$ running time in worst and average case
  - $\approx n^2/2$ comparisons and $\approx n^2/2$ exchanges

Bubble Sort

- Idea:
  - Repeatedly pass through the array
  - Swaps adjacent elements that are out of order



- Easier to implement, but slower than Insertion sort

Example

8	4	6	9	2	3	1

$i = 1$ ← - - - - - - - - - - - - - j

8	4	6	9	2	1	3

$i = 1$ ← - - - - - - - - - - j

8	4	6	9	1	2	3

$i = 1$ ← - - - - - - - j

8	4	6	1	9	2	3

$i = 1$ ← - - - - - j

8	4	1	6	9	2	3

$i = 1$ ← - - - j

8	1	4	6	9	2	3

$i = 1$ j

1	8	4	6	9	2	3

$i = 1$ j

1	8	4	6	9	2	3

$i = 2$ j

1	2	8	4	6	9	3

$i = 3$ j

1	2	3	8	4	6	9

$i = 4$ j

1	2	3	4	8	6	9

$i = 5$ j

1	2	3	4	6	8	9

$i = 6$ j

1	2	3	4	6	8	9

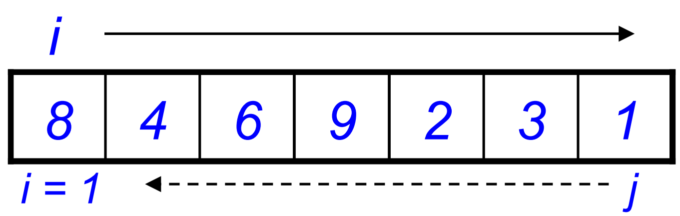
$i = 7$

j

Bubble Sort

Alg.: BUBBLESORT(A)

```
for i ← 1 to length[A]
    do for j ← length[A] downto i + 1
        do if A[j] < A[j -1]
            then exchange A[j] ↔ A[j-1]
```



Bubble-Sort Running Time

Alg.: BUBBLESORT(A)

for i ← 1 **to** length[A] c_1

do for j ← length[A] **downto** i + 1 c_2

Comparisons: ≈ $n^2/2$ **do if** A[j] < A[j -1] c_3

Exchanges: ≈ $n^2/2$ **then** exchange A[j] ↔ A[j-1] c_4

$$T(n) = c_1(n+1) + c_2\sum_{i=1}^{n}(n-i+1) + c_3\sum_{i=1}^{n}(n-i) + c_4\sum_{i=1}^{n}(n-i)$$

$$= \Theta(n) + (c_2 + c_2 + c_4)\sum_{i=1}^{n}(n-i)$$

$$\text{where } \sum_{i=1}^{n}(n-i) = \sum_{i=1}^{n}n - \sum_{i=1}^{n}i = n^2 - \frac{n(n+1)}{2} = \frac{n^2}{2} - \frac{n}{2}$$

Thus, $T(n) = \Theta(n^2)$

Dr. Md. Abul Kashem Mia, Professor, CSE Dept, BUET

Selection Sort

- Idea:
  - Find the smallest element in the array
  - Exchange it with the element in the first position
  - Find the second smallest element and exchange it with the element in the second position
  - Continue until the array is sorted

Example

8	4	6	9	2	3	1

1	4	6	9	2	3	8

1	2	6	9	4	3	8

1	2	3	9	4	6	8

1	2	3	4	9	6	8

1	2	3	4	6	9	8

1	2	3	4	6	8	9

1	2	3	4	6	8	9

Selection Sort

Alg.: SELECTION-SORT*(A)*

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

```
n ← length[A]
for j ← 1 to n - 1 do
        smallest ← j
        for i ← j + 1 to n do
                if A[i] < A[smallest]
                        then smallest ← i
        exchange A[j] ↔ A[smallest]
```


Dr. Md. Abul Kashem Mia, Professor, CSE Dept, BUET

Analysis of Selection Sort

Alg.: SELECTION-SORT(*A*)	*cost*	*times*
$n \leftarrow$ length[A]	c_1	*1*
for $j \leftarrow 1$ **to** $n - 1$	c_2	*n*
do smallest $\leftarrow j$	c_3	*n-1*
for $i \leftarrow j + 1$ **to** n	c_4	$\sum_{j=1}^{n-1}(n-j+1)$
do if A[i] < A[smallest]	c_5	$\sum_{j=1}^{n-1}(n-j)$
then smallest $\leftarrow i$	c_6	$\sum_{j=1}^{n-1}(n-j)$
exchange A[j] $\leftrightarrow$ A[smallest]	c_7	*n-1*

$\approx n^2/2$ comparisons

$\approx n$ exchanges

$$T(n) = c_1 + c_2 n + c_3(n-1) + c_4\sum_{j=1}^{n-1}(n-j+1) + c_5\sum_{j=1}^{n-1}(n-j) + c_6\sum_{j=2}^{n-1}(n-j) + c_7(n-1) = \Theta(n^2)$$