Divide and Conquer Technique (Cont..)
&
Searching and Sorting Techniques

By
Prof. Shaik Naseera
Department of CSE
JNTUA College of Engg., Kalikiri

Objectives

- Strassen’s Matrix Multiplication
- Searching and Sorting Techniques

Matrix Multiplication: General Method

- The usual way to multiply two n x n matrices A and B, yielding result matrix 'C' as follows :

```
for i := 1 to n do
    for j :=1 to n do
        c[i, j] := 0;
        for K: = 1 to n do
            c[i, j] := c[i, j] + a[i, k] * b[k, j];
```

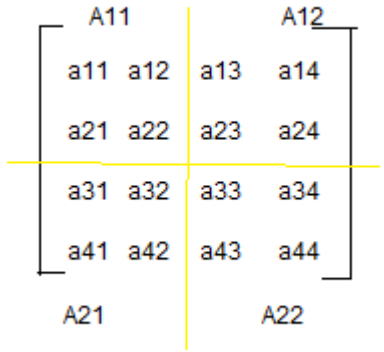
- Which leads to T (n) = O (n^3).

Divide and conquer approach

- The divide-and-conquer strategy suggests another way to compute the
- product of two *n x n matrices.*
- *For simplicity we will assume that n is a* power of 2, i.e. that there exists a nonnegative integer *k such that n = 2^k.*
- In case *n is not a power of two then enough rows and columns of zeros* may be added to both *A and B so that the resulting dimensions are a power* of two.
- Imagine that *A and B* are each partitioned into four square sub matrices, each sub matrix having dimensions *n/2 x n/2.*

- *Then the product AB can be computed by*

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$$



$$C_{11} = A_{11}B_{11} + A_{12}B_{21}$$
$$C_{12} = A_{11}B_{12} + A_{12}B_{22}$$
$$C_{21} = A_{21}B_{11} + A_{22}B_{21}$$
$$C_{22} = A_{21}B_{12} + A_{22}B_{22}$$

- The recurrence relation for the above computation is

$$T(n) = \begin{cases} b & n \leq 2 \\ 8T\left(\frac{n}{2}\right) + cn^2 & n > 2 \end{cases}$$

Solving of this recurrence relation gives $O(n^3)$.

Strassen's Method

- Strassen's insight was to find an alternative method for calculating the C_{ij}, requiring seven (n/2) x (n/2) matrix multiplications and eighteen (n/2) x (n/2) matrix additions and subtractions:
- $P = (A_{11} + A_{22}) (B_{11} + B_{22})$
- $Q = (A_{21} + A_{22}) B_{11}$
- $R = A_{11} (B_{12} – B_{22})$
- $S = A_{22} (B_{21} - B_{11})$
- $T = (A_{11} + A_{12}) B_{22}$
- $U = (A_{21} – A_{11}) (B_{11} + B_{12})$
- $V = (A_{12} – A_{22}) (B_{21} + B_{22})$
- $C_{11} = P + S – T + V$
- $C_{12} = R + T$
- $C_{21} = Q + S$
- $C_{22} = P + R - Q + U.$
- This method is used recursively to perform the seven (n/2) x (n/2) matrix multiplications, then the recurrence equation for the number of scalar multiplications performed is:

Complexity Analysis

- Recurrence Relation is

$$T(n) = \begin{cases} b & n \leq 2 \\ 7T\left(\frac{n}{2}\right) + cn^2 & n > 2 \end{cases}$$

$$\begin{aligned} T(n) &= an^2(1 + 7/4 + (7/4)^2 + \ldots + (7/4)^{k-1}) + 7^k T(1) \\ &\leq cn^2 (7/4)^{\log_2 n} + 7^{\log_2 n}, \quad c \text{ a constant} \\ &= cn^{\log_2 4 + \log_2 7 - \log_2 4} + n^{\log_2 7} \\ &= O(n^{\log_2 7}) \approx O(n^{2.81}) \end{aligned}$$

- Strassens Complexity.docx

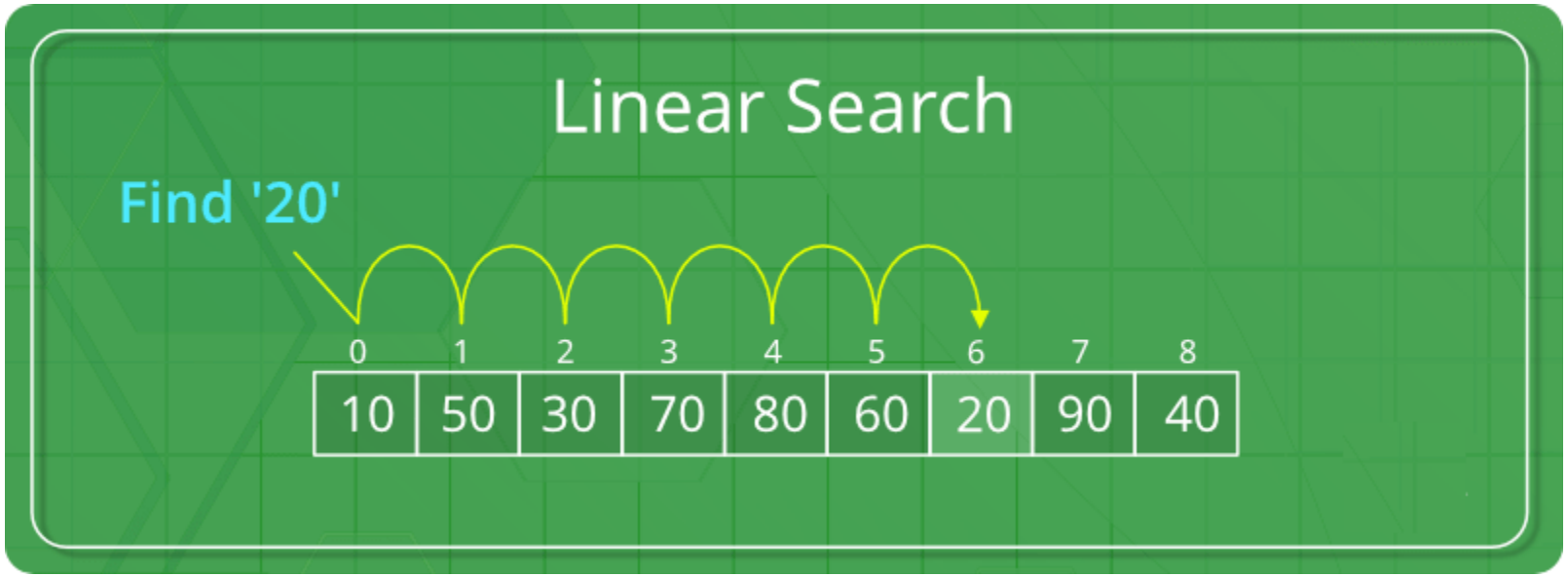
Searching & Sorting Techniques

Searching & Sorting Techniques

- Searching Techniques:
  - Linear Search
  - Binary Search
- Sorting Techniques
  - Selection Sort
  - Bubble Sort
  - Insertion Sort
  - Heap Sort
  - Quick Sort
  - Merge Sort

Eg. 1 - Linear Search

- Recursively Look at an element (constant work, c), then search the remaining elements...



Linear Search Algorithm

Algorithm 1: Linear search for a value inside an array

```
Data: A: The array to search for the value inside
      n: The length of the array
      value: The value to search for
Result: Returns the index of the element if found, or -1 otherwise
for i ← 1 to n do
    if A[i] = value then
        return i;
    end
end
return −1;
```

Time complexity

- Best Case O(1)
- Worst Case T(n)=T(n-1)+1

  =T(n-2)+2

  =T(n-3)+3

  = T(n-k)+k

  when k=n, =T(0)+n

  = O(n)

- T(n) = T(n-1) + c //“The cost of searching n elements is the cost of looking at 1 element, plus the cost of searching n-1 elements”

Eg. 1 – list of intermediates

Result at i^{th} unwinding	i
$T(n) = T(n-1) + 1c$	1
$T(n) = T(n-2) + 2c$	2
$T(n) = T(n-3) + 3c$	3
$T(n) = T(n-4) + 4c$	4

Linear Search

- An expression for the kth unwinding:

  $T(n) = T(n-k) + kc$

Selection Sort

- One of the simplest techniques used is the selection sort which works for the given lists of item or data each time by selecting one item at a time and orders along with placing it in a correct position in sequence.
- This does not require extra storage space to hold the lists.
- For selecting the item randomly this technique is helpful.

Algorithm

- The following steps are used for processing the elements in given lists of items.

  Step 1: First finds the smallest element for the given lists of item or data.

  Step 2: Replaces that item or data in the first position.

  Step 3: Next, again finds the smallest element among the lists.

  Step 4: Get replaced in second position.

  Step 5: Same procedure is followed unless the elements are sorted.

  Step 6: Returns result.

Example: Selection Sort

Algorithm

```
Algorithm selection_sort(arr):
  n = len(arr)
  for i in range(0,n-1):
          min = i
          for j in range(i+1,n):
                  if arr[j] < arr[min]
                          min = j
          if i != min:
                  temp = arr[i]
                  arr[i] = arr[min]
                  arr[min] = temp
  return arr
```

Time complexity is $O(n^2)$

Time complexity

Selection sort analysis

Iteration of outer loop	No. times array comparison performed during this iteration of outer loop
0	n-1
1	n-2
2	n-3
. . .	
last	1

So number of comparisons is

$$1 + 2 + 3 + \ldots + (n-2) + (n-1) = n * (n-1) / 2 = n^2/2 - n/2$$

As n gets large, the term n^2 dominates. We say the number if comparisons *is proportional to* n^2 and that this is a *quadratic* algorithm.

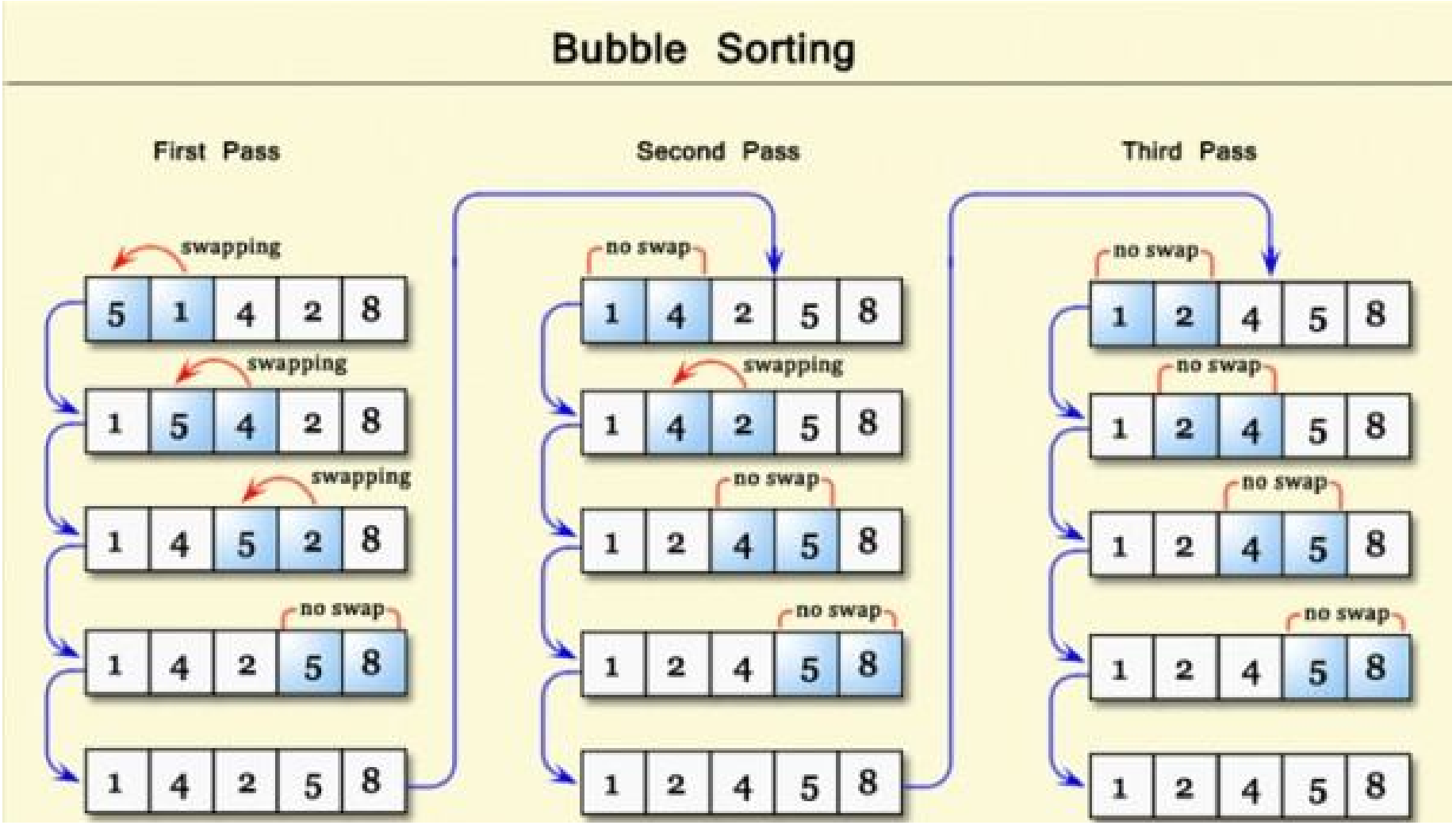
- **Advantages:**
- Before sorting the given lists, the ordering of items or datas can be initialized.
- This works even for the smaller lists of items or datas.
- Need no extra storage for the original lists of items.
- **Disadvantages:**
- For the large set of items or datas this sorting results in poor efficiency.
- Requires an N-squared number steps for sorting items for the given lists.
- Compared to selection sort, the quick sort is most efficient.

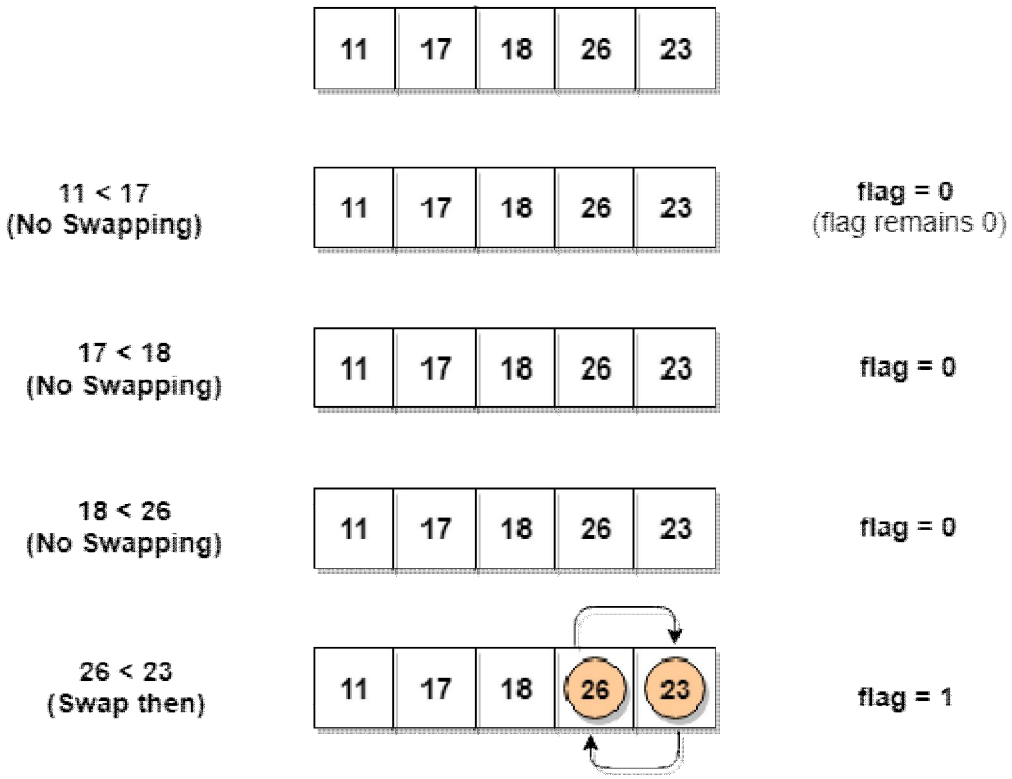
Bubble Sort

- **Bubble Sort** is a simple algorithm which is used to sort a given set of n elements provided in form of an array with n number of elements. Bubble Sort compares all the element one by one and sort them based on their values.
- If the given array has to be sorted in ascending order, then bubble sort will start by comparing the first element of the array with the second element, if the first element is greater than the second element, it will **swap** both the elements, and then move on to compare the second and the third element, and so on.
- If we have total n elements, then we need to repeat this process for n-1 times.
- It is known as **bubble sort**, because with every complete iteration the largest element in the given array, bubbles up towards the last place or the highest index, just like a water bubble rises up to the water surface.
- Sorting takes place by stepping through all the elements one-by-one and comparing it with the adjacent element and swapping them if required.

Bubble Sort

- Compares the adjacent elements, if the element on the right side is smaller then swap their positions.
- This procedure continues till the end of the list. i.e., end of the pass 1.
- At the end of the pass 1, the largest element will be at the last position.
- The above procedure is repeated for n-1 times.


Bubble Sorting
First Pass
Second Pass
Third Pass
swapping
5 1 4 2 8
swapping
1 5 4 2 8
swapping
1 4 5 2 8
no swap
1 4 2 5 8
1 4 2 5 8
no swap
1 4 2 5 8
swapping
1 4 2 5 8
no swap
1 2 4 5 8
no swap
1 2 4 5 8
1 2 4 5 8
no swap
1 2 4 5 8
no swap
1 2 4 5 8
no swap
1 2 4 5 8
1 2 4 5 8


11 17 18 26 23
11 < 17
(No Swapping)
11 17 18 26 23
flag = 0
(flag remains 0)
17 < 18
(No Swapping)
11 17 18 26 23
flag = 0
18 < 26
(No Swapping)
11 17 18 26 23
flag = 0
26 < 23
(Swap then)
11 17 18 26 23
flag = 1

Complexity

- Complexity is $O(n^2)$

```
1   Procedure bubblesort (List array, number length_of_array)
2           for i=1 to length_of_array - 1;
3                   for j=1 to length_of_array – I;
4                           if array [j] > array [j+1] then
5                                   temporary = array [j+1]
6                                   array[j+1] = array [j]
7                                   array[j] = temporary
8                           end if
9                   end of j loop
10          end of i loop
11  return array
12  End of procedure
```

Performance Improvement

```
1   Procedure bubblesort (List array, number length_of_array)
2         for i=1 to length_of_array - 1;                          <-- flag = 0
3               for j=1 to length_of_array – I;
4                     if array [j] > array [j+1] then
5                           temporary = array [j+1]
6                           array[j+1] = array [j]
7                           array[j] = temporary                   <-- flag = 1
8                     end if
9               end of j loop
10        end of i loop                                            <-- if flag = 0, break the loop
11  return array
12  End of procedure
```

The best case time complexity is O(n)

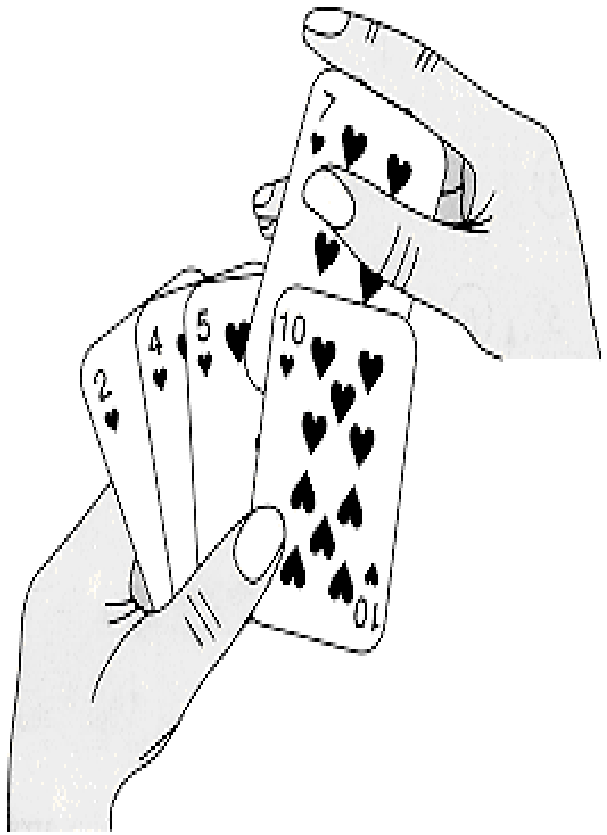
- If the elements are almost in sorted order, bubble sort is more efficient sorting technique.

Example: Insertion Sort

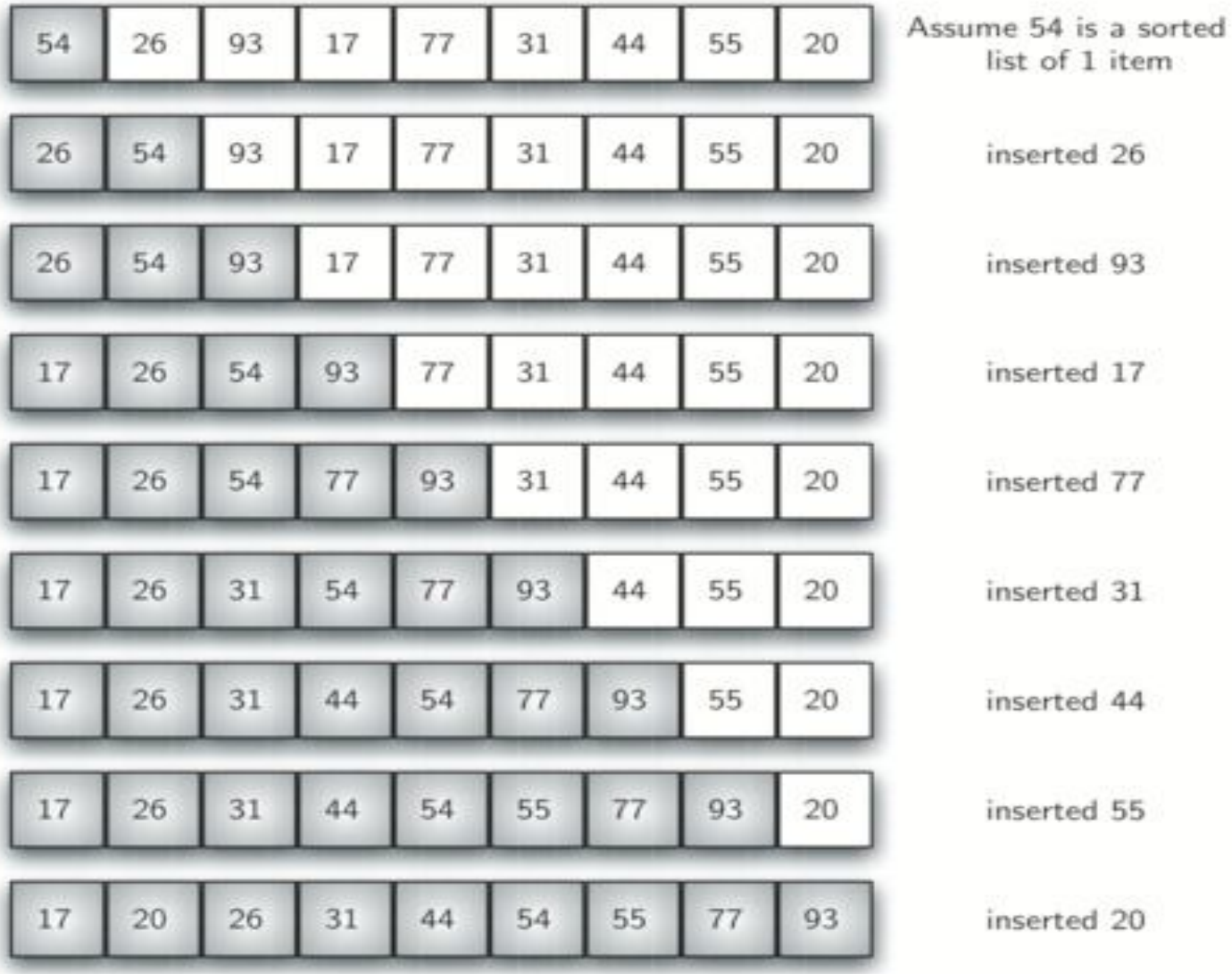
- Insertion sort is a comparison-based sorting algorithm that we will use as an example to understand some aspects of algorithmic analysis and to demonstrate how an iterative algorithm can be shown to be correct.
- The principle behind insertion sort is to remove an element from an un-sorted input list and insert in the correct position in an already-sorted, but partial list that contains elements from the input list.

Example: Sorting of cards

- We start with an empty left hand
- Cards face down on the table
- We remove one card at a time from the table and insert it into the correct position in the left hand.
- To find the correct position, we compare it with each of the cards already in the left hand, from right to left.
- At all the times, the cards held in the left hand are in sorted order.



Example2


54 26 93 17 77 31 44 55 20
Assume 54 is a sorted list of 1 item
26 54 93 17 77 31 44 55 20
inserted 26
26 54 93 17 77 31 44 55 20
inserted 93
17 26 54 93 77 31 44 55 20
inserted 17
17 26 54 77 93 31 44 55 20
inserted 77
17 26 31 54 77 93 44 55 20
inserted 31
17 26 31 44 54 77 93 55 20
inserted 44
17 26 31 44 54 55 77 93 20
inserted 55
17 20 26 31 44 54 55 77 93
inserted 20

Pseudo code

```
INSERTION-SORT(A)
1   for j = 2 to A.length
2       key = A[j]
3       // Insert A[j] into the sorted
            sequence A[1 . . j − 1].
4       i = j − 1
5       while i > 0 and A[i] > key
6           A[i + 1] = A[i]
7           i = i − 1
8       A[i + 1] = key
```

Time Complexity

- Best Case O(n)
- Average and worst case $O(n^2)$

Big O of Sorting Algorithms

Algorithm	Time Complexity (Best)	Time Complexity (Average)	Time Complexity (Worst)
Bubble Sort	$O(n)$	$O(n^2)$	$O(n^2)$
Insertion Sort	$O(n)$	$O(n^2)$	$O(n^2)$
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$

Previous Year Gate Questions

Q. NO. 1

Which one of the following is the tightest upper bound that represents the number of swaps required to sort n numbers using selection sort?

(A) O(log n)

(B) O(n)

(C) O(nLogn)

(D) O(n^2)

Answer: (B)

Explanation: To sort elements in increasing order, selection sort always picks the maximum element from remaining unsorted array and swaps it with the last element in the remaining array. So the number of swaps, it makes in n-1 which is O(n)

Q. NO. 2

GATE CSE 2003

The usual $\Theta(n^2)$ implementation of Insertion Sort to sort an array uses linear search to identify the position where an element is to be inserted into the already sorted part of the array. If, instead, we use binary search to identify the position, the worst case running time will

A remain $\Theta(n^2)$

B become $\Theta(n(\log n)^2)$

C become $\Theta(n \log n)$

D become $\Theta(n)$

Note: In case, if the binary search identifies the position 1 for the element, then all the n elements next to the position of 1 should be shifted one position to the right to accommodate the element. It requires n operations. Therefore, there would not be any change.

Q. NO. 3

GATE CSE 1999

A sorting technique is called stable if:

A. It takes O(nlog n)time

B. It maintains the relative order of occurrence of non-distinct elements

C. It uses divide and conquer paradigm

D. It takes O(n) space

Q. NO. 4

GATE CSE 1994

The recurrence relation that arises in relation with the complexity of binary search is:

A $T(n) = 2T\left(\frac{n}{2}\right) + k$, k is a constant

B $T(n) = T\left(\frac{n}{2}\right) + k$, k is a constant

C $T(n) = T\left(\frac{n}{2}\right) + \log n$

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

Q. NO. 5

GATE CSE 1996

The average number of key comparisons done on a successful sequential search in list of length n is

A $\log n$

B $\frac{n-1}{2}$

C $\frac{n}{2}$

D $\frac{n+1}{2}$

GATE CSE 2016 Set 2

Q. NO. 6

Assume that the algorithms considered here sort the input sequences in ascending order. If the input is already in ascending order, which of the following are **TRUE**?

I. Quicksort runs in $\Theta\left(n^2\right)$ time
II. Bubblesort runs in $\Theta\left(n^2\right)$ time
III. Mergesort runs in $\Theta\left(n\right)$ time
IV. Insertion sort runs in $\Theta\left(n\right)$ time

A. I and II only

B. I and III only

C. II and IV only

D. I and IV only

Quick sort: pivot is left element, so worst case
Insertion Sort: Does not require element shifting. Only outer loop executes. Best case.

GATE CSE 2016 Set 2

Q. NO. 7

Assume that the algorithms considered here sort the input sequences in ascending order. If the input is already in ascending order, which of the following are **TRUE**?

I. Quicksort runs in $\Theta\left(n^2\right)$ time

II. Bubblesort runs in $\Theta\left(n^2\right)$ time

III. Mergesort runs in $\Theta\left(n\right)$ time

IV. Insertion sort runs in $\Theta\left(n\right)$ time

A. I and II only

B. I and III only

C. II and IV only

D. I and IV only

GATE CSE 2009

Q. NO. 8

What is the number of swaps required to sort n elements using selection sort, in the worst case?

A $\Theta(n)$

B $\Theta(n \log n)$

C $\Theta(n^2)$

D $\Theta(n^2 \log n)$

GATE CSE 2007

Q. NO. 9

Which of the following sorting algorithms has the lowest worst-case complexity?

A Merge sort

B Bubble sort

C Quick sort

D Selection sort

O(nlogn)

GATE CSE 2006

Q. NO. 10

Which one of the following in place sorting algorithms needs the minimum number of swaps?

A Quick sort

B Insertion sort

C Selection sort

Only one swap need in each pass

D Heap sort

GATE CSE 1999

Q. NO. 11

Suppose we want to arrange the n numbers stored in any array such that all negative values occur before all positive ones. Minimum number of exchanges required in the worst case is

A n - 1

B n

C n + 1

D None of the above

Explanation

Worst case happens when all the positive numbers occur before the negative numbers.

In this case, take a positive number from the left side and negative number from the right side and do exchange operation. Then after n/2 exchanges operation, you will reach middle of the array and all the negative value will be present before positive value.

so in worst case n/2 exchanges required.

Q. NO. 12 **GATE CSE 1998**

Give the correct matching for the following pairs:

Group - 1

(A) $O(\log n)$
(B) $O(n)$
(C) $O(n \log n)$
(D) $O(n^2)$

Group - 2

(P) Selection
(Q) Insertion sort
(R) Binary search
(S) Merge sort

Explanation

Selection sort : $O(n^2)$
Merge sort : $O(n \log n)$
Binary search : $O(\log n)$
Insertion sort : $O(n)$

So the correct pair is A – R B – Q C – S D – P

But this option is not present so you should go with option (B) as A – R and C – S only matches with (B).

A A – R B – P C – Q D – S

B A – R B – P C – S D – Q

C A – P B – R C – S D – Q

D A – P B – S C – R D – Q

Q. NO. 13

GATE CSE 1997

The correct matching for the following pairs is

A. All pairs shortest path	1. Greedy
B. Quick Sort	2. Depth-First Search
C. Minimum weight spanning tree	3. Dynamic Programming
D. Connected Components	4. Divide and Conquer

A. A-2 B-4 C-1 D-3

B. A-3 B-4 C-1 D-2

C. A-3 B-4 C-2 D-1

D. A-4 B-1 C-2 D-3

$$T(n) = \begin{cases} b & \text{if } n \le 2 \\ 7T(\frac{n}{2}) + cn^2 & n > 2 \end{cases}$$

$$T(n) = 7T(\tfrac{n}{2}) + cn^2$$

$$= 7\left[7T(\tfrac{n}{4}) + c(\tfrac{n}{2})^2\right] + cn^2$$

$$= 7^2 T(\tfrac{n}{4}) + (\tfrac{7}{4})cn^2 + cn^2$$

$$= 7^2\left[7T(\tfrac{n}{8}) + c(\tfrac{n}{4})^2\right] + \tfrac{7}{4}cn^2 + cn^2$$

$$= 7^3 T(\tfrac{n}{8}) + (\tfrac{7}{4})^2 cn^2 + \tfrac{7}{4}cn^2 + cn^2$$

$$\vdots$$

$$= 7^k T(\tfrac{n}{2^k}) + \left[(\tfrac{7}{4})^{k-1} + (\tfrac{7}{4})^{k-2} + \cdots + 1\right]cn^2$$

Let $n = 2^k \Rightarrow k = \log n$.

$$= 7^{\log_2 n} T(1) + cn^2\left[(\tfrac{7}{4})^{k-1} + (\tfrac{7}{4})^{k-2} + \cdots + 1\right]$$

$$= n^{\log_2 7} + cn^2\left[(\tfrac{7}{4})^{k-1} + (\tfrac{7}{4})^{k-2} + \cdots + 1\right]$$

$$\le n^{\log_2 7} + an^2(\tfrac{7}{4})^k$$

$$= n^{\log_2 7} + an^2(\tfrac{7}{4})^{\log n}$$

$$= n^{\log_2 7} + an^2 n^{\log_2(\frac{7}{4})}$$

$$a^{\log_c b} = b^{\log_c a}$$

$$n^{\log 4} = n^2$$

$$= n^{\log_2 7} + a\, n^{\log 4} \cdot n^{\log_2(\frac{7}{4})}$$

$$= n^{\log_2 7} + a\, n^{\log 4 + \log_2 7 - \log 4}$$

$$= n^{\log_2 7}(1 + a) = O(n^{2.81})$$