MODULE 5

SORTING TECHNIQUES

Sorting method can be classified into two.

1)Internal Sorting

2)External Sorting

In internal sorting the data to be sorted is placed in main memory. In external sorting the data is placed in external memory such as hard disk, floppy disk etc. The internal sorting can be classified into

1)n^2 sorting

2)n log n sorting

n^2 sorting - it can be classified into

1)Bubble sort

2) Selection sort 3) Insertion sort

nlogn sorting - It can be classified into

1) Merge sort

2)Quick sort

3) Heap sort

Bubble sort

Bubblesort(a[],n)

Input- an array a[size], n is the no. of element currently present in array

Output- Sorted array

DS- Array

Algorithm

1. Start
2. i=0
3. While i<n-1
    1. j=0
    2. While j<n-1-i
        1. If a[j]>a[j+1]
            1. temp=a[j]
            2. a[j]=a[j+1]
            3. a[j+1]=temp
        2. end if
        3. j=j+1

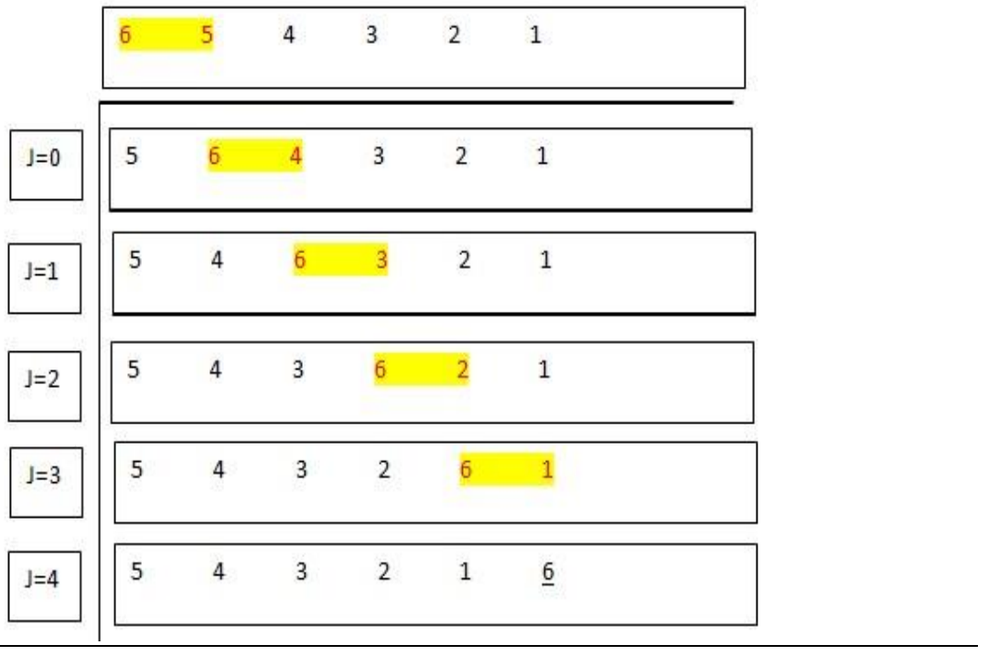
3.end while

4.i=i+1

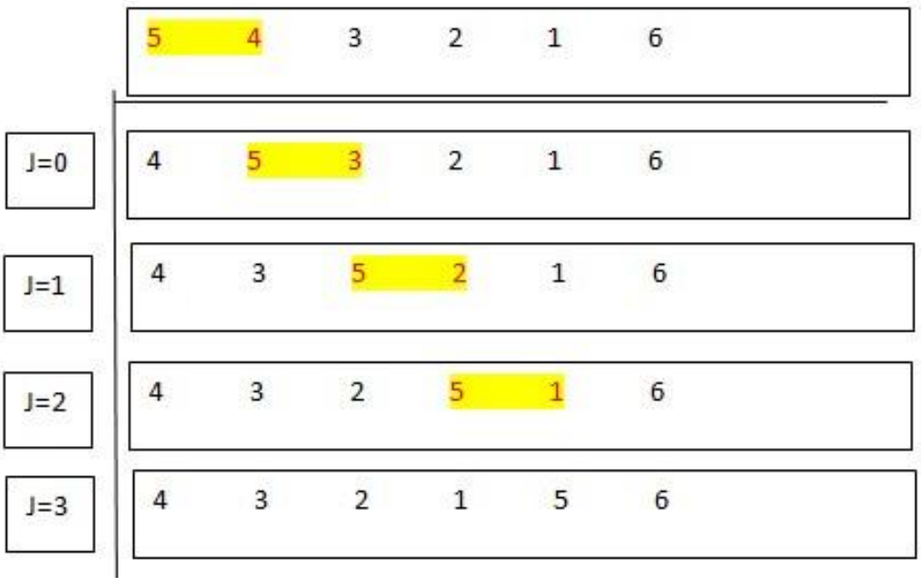
4. end while

Here first element is compared with second element. If first one is greater than second element then swap each other. Then second element is compared with third element . If second element is greater than third element then perform swapping. This process is continued until the comparison of (n-1)th element with nth element. These process continues (n-1) times. Consider the example-

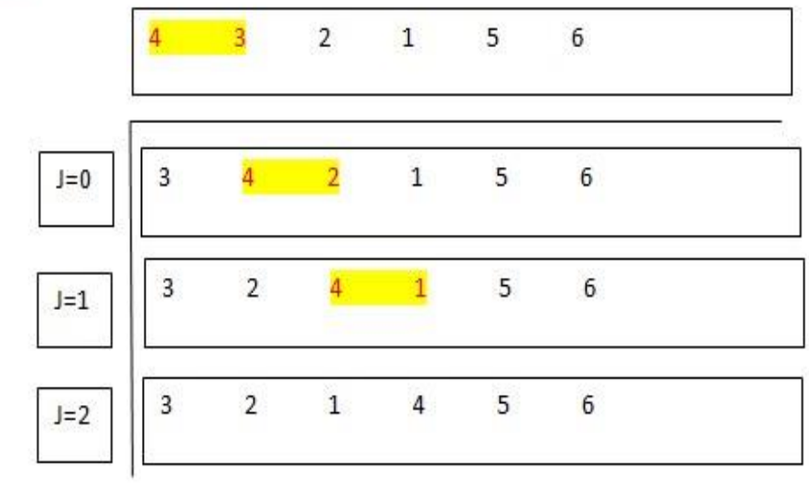
i=0



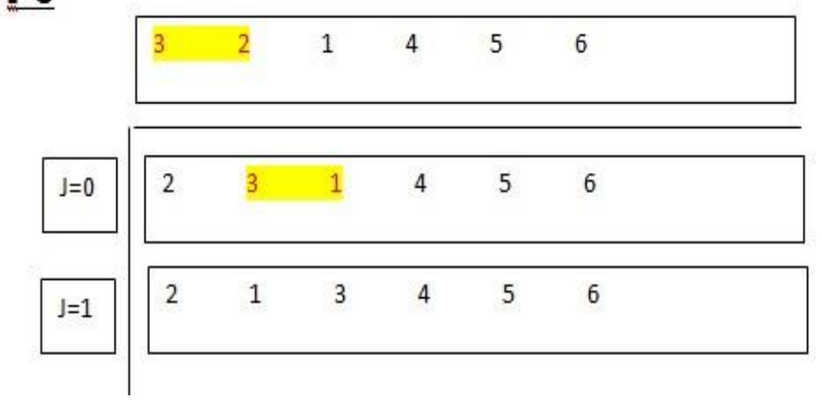
i=1



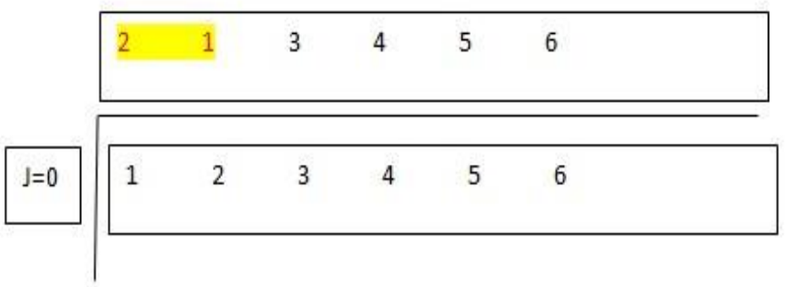
i=2



i=3



i=4



Thus i have values from 0 to n-1 and j have values from 0 to n-1-i.

Analysis

Here during the first iteration n-1 comparisons are required. During the second iteration n-2 comparisons are required etc..during last iteration, 1 comparison is required. Therefore total comparisons

= (n-1)+(n-2)+(n-3)+…..+1 $n(n-1)$

$=$ $\frac{}{2}$

$=O(n^2)$

SELECTION SORT

Selectionsort(a[],n)

Input: An unsorted array a[], n is the no.of elements

Output: a sorted array

DS: Array

Algorithm

1: Start

2. i=0

3. while i<n-1 do

    1. j=i+1

    2. small=i

    3. while j<n do

        1.if a[small]>a[j]

        1. small=j

        2. end if

        3. j=j+1

        4. end while

        5. if i!=small

            1. temp=a[i]

            2. a[i]=a[small]

            3. a[small]=temp

        6. end if

        7. i=i+1

4. end while

5. stop

Here the first element in the array is selected as the smallest element. Then check for any element smaller than this from the second position to the last. If found any, then they are interchanged. Similarly next we check for the smallest element from third position to last. And the process continues upto (n-1)th position.

Analysis

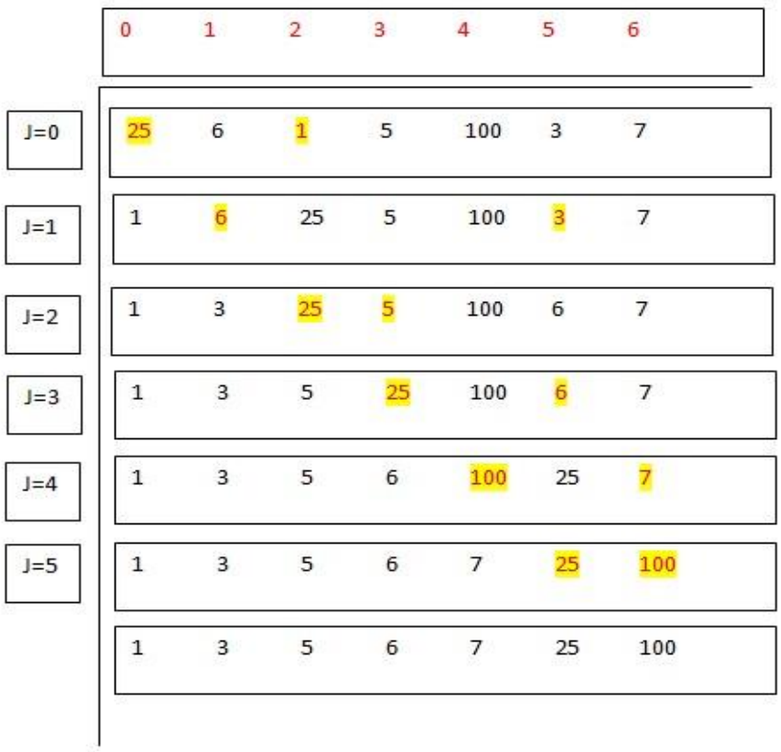
Here first iteration when i=0 takes n-1 comparisons, during second iteration takes n-2 comparison and so on.

Total no. of comparisons=(n-1)+(n-2)+(n-3)+.....+2+1 $n(n-1)$

$$= \frac{\qquad}{2}$$

$$= O(n^2)$$

Example



INSERTION SORT

insertionsort(a[],n)

Input: An unsorted array a[], n is the no.of elements

Output: a sorted array

DS: Array

Algorithm

1. Start
2. i=1
3. While i<n
    1. j=i
    2. While a[j]<a[j-1] and j>0
        1. t=a[j]
        2. a[j]=a[j-1]
        3. a[j-1]=t
        4. j=j-1
    3. end while
    4. i=i+1

4.end while

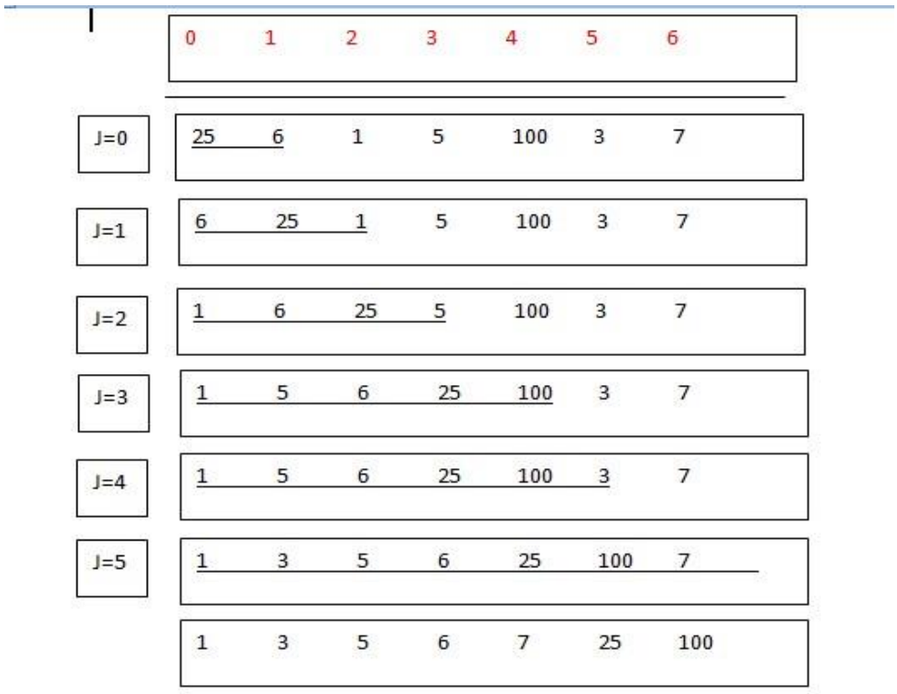
5. stop

Here first iteration takes 1 comparison, 2nd iteration takes 2 comparison, and last iteration takes (n-1) comparison

Total comparison=1+2+…+n-1 $n(n-1)$

$$= \frac{\quad}{2}$$

= O(n^2) Example



MERGE SORT

mergesort(start,end)

Input: An unsorted array a[], n is the no.of elements, start indicates lower bound of the array, end indicates the upper bound of the array

Output: a sorted array

DS: Array

Algorithm

1.start

2. if(start!= end)

    1. mid=(start+end)/2
    2. mergesort(start,mid)
    3.mergesort(mid+1,end)
    4. merge(start,mid,end)

3. end if

4.stop

Algorithm for merge

Merge(start,mid,end)

1. i=start

2. j=mid+1
3. k=start
4. while i<=mid and j<= end do
    1. if a[i]<=a[j]
        1. temp[k]=a[i]
        2. i=i+1
        3. k=k+1
    2. Else
        1. Temp[k] =a[j]
        2. J=j+1
        3. k=k+1
    3. end if

5.end while

6. while i<=mid do
    1. temp[k]=a[i]
    2. i=i+1
    3. k=k+1
7. end while
8. While j<=end
    1. Temp[k]=a[j]
    2. j=j+1
    3. k=k+1
9. end while
10. k=start
11. while k<=end
    1. a[k]=temp[k]
    2. k=k+1
12. end while
13.stop

Merge sort follows the strategy divide and conquer method. Here the given base array is divided into two sub lists. These 2 sub lists is again divided into 4 sub lists. The process is continued until subsists contain single element. Then repeatedly merge these two sub lists to a single sub list. So that a sorted array is created from sorted sub lists. The process continues until a sub list contains all the elements that are sorted**.**

Analysis of merge sort

Suppose the merge sort follows the recurrence relation.

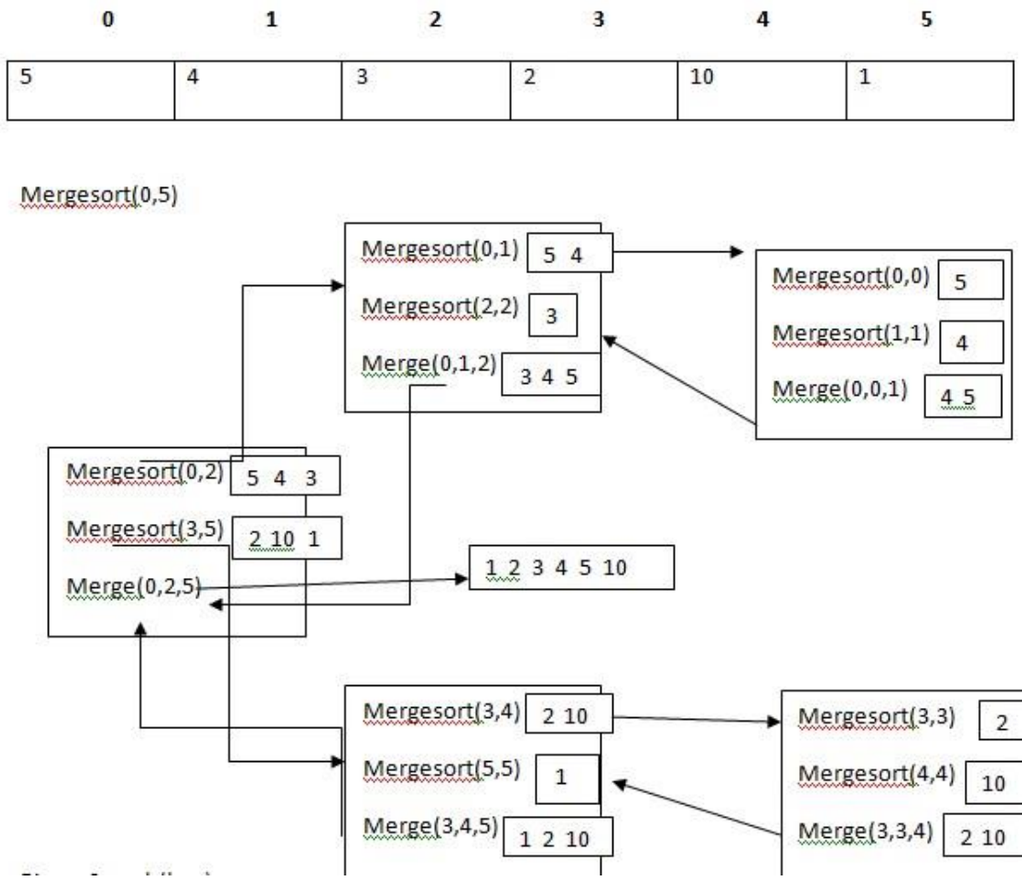
$$T(n)=2T(n/2)+n$$

Comparing with standard recurrence equation T(n)=aT(n/b)+f(n)

a=2, b=2, f(n)=n then $n^{\log_b a}=n$

therefore time complexity is $\Theta(n\log n)$

Consider the following example



QUICK SORT

Quicksort(lb,ub)

Input: An unsorted array a[], n is the no.of elements, lb indicates lower bound of the array, ub indicates the upper bound of the array

Output: a sorted array

DS: Array

Algorithm

1.start

2.if lb<ub

1. loc=partition(lb,ub)
2. quicksort(lb,loc-1)
3. quicksort(loc+1,ub)

3. end if
4. stop

Algorithm for partition

Partition(lb,ub)

1. pivot=a[lb]
2. up=ub
3. down=lb
4. while down< up
    1. while pivot>=a[down] and down<=up
        1.down=down+1
    2. end while
    3. while a[up]>pivot
        1.up=up-1
    4. end while
    5. if down< = up
        1. swap(a[down], a[up])
    6. end if
5. end while
6. swap(a[lb],a[up])
7. return up
8. stop

Quick sort algorithm basically takes the following steps

1. Choose a pivot element(the pivot element may be in any position). Normally first element is chosen as pivot
2. Perform partition function in such a way that all the elements which are lesser than pivot goes to the left part of the array and all the elements greater than pivot go to the right part of the array. The partition function also places pivot in exact position.
3. Recursively perform quick sort algorithm in these two sub arrays

Analysis

The recurrence relation ofquick sort in worst case is

$$T(n)=T(n/2)+T(n/2)+\Theta(n_2)$$

$$T(n)=2T(n/2)+\Theta(n^2)$$

Comparing with the standard recurrence equation

T(n)=aT(n/b)+f(n)

Substitute the values of a and b in $n_{\log_b a}$and compare it with f(n).

In this case a=2, b=2 then $n_{\log_b a}$=n

Whereas f(n)= n_2. Therefore time complexity of quick sort is $\Theta(n_2)$