Merge Sort Algorithm

Divide and conquer strategy

Two parts of algorithm

Divide: Until we get single ton elements

Combine: combine partitions in the sorted order to get a final sorted list

partitionInMergeSort(arr,start,end)

begin

If start < end then

mid = (start+end)/2

partitionInMergeSort(arr,start,mid)

partitionInMergeSort(arr,mid+1,end)

combine(arr,start,mid,end)

endif

end

combine(arr,start,mid,end)

begin

length1= mid-start+1

length2=end-mid

for i=0 to length1

left\_arr[i]=arr[start+i]

endfor

for j=0 to length2

right\_arr[j]=arr[mid+1+j]

endfor

m=0,n=0,l=start

//until one array gets exhausted

while( m<length1 && n<length2)

if(left\_arr[m]>=right\_arr[n])

comb\_arr[l]=left\_arr[m]

m=m+1

else

comb\_arr[l]=right\_arr[n]

n++

endif

l++

endwhile

//if elements present in left array

While m<length1

comb\_arr[l]=left\_arr[m]

m++

l++

endwhile

//if elements present in right array

while(n<length2)

comb\_arr[i]=right\_arr[n]

n++

l++

endwhile

return comb\_arr

end

Sorting algorithms Complexity analysis:

Bubble Sort:

Best case : O(n)---no swaps ,if array already sorted in the expected order(ascending/descending)

Worst case:O(n2)---if array sorted in reverse of expected order

Average case :O(n2)--- random array

Selection sort:

In all Best ,worst and average case analysis ,complexity is O(n2)

Max element need to compare with all elements of list

Insertion Sort:

Best case : O(n)

Array sorted already

Worst case and average case : O(n2)

Quick Sort:

Worst case : O(n2)--- array elements sorted in reverse order of the expected

Best case : pivot placed in the middle of the list making almost equal partition

o(n log(n))

MergeSort : Best, average and worst case , in all scenarios its O(n log(n))

Hashing and Hashtable:

To make searching simpler

HashTable is a data structure which can be implemented either by array or by linked list

Hashing is a technique to place or to retrieve an element from hashtable

Hashing uses hash function

Modulo is used as hash function

Ex:

Hash function is %10

Tree:

Root node

Leaf nodes

Path

Internal nodes

External nodes

Height

Depth

Siblings

Parent and Grandparents

Ancestors and Descendents

Degree

Edge

Binary Tree:

Binary Search Tree

Complete Binary Tree

Heap

Max heap and Min heap

Heap Sort:

heapSort(arr)

Begin

n=arr.length

for i= n/2-1 to 0

heapify(arr,n,i)

End for

End

heapfy(arr,n,i)

Begin

parent\_index=i

left\_index=2 \* i+1

right\_index=2\* i +2

If left\_index<n && arr[left\_index]>arr[parent\_index] then

parent\_index=left\_index

End if

If right\_index< n && arr[right\_index]>arr[parent\_index] then

parent\_index=right\_index

Endif

if(parent\_index != i)

swap(arr[i] ,arr[parent\_index])

Endif

heapify(arr,n,parent\_index)

end