Dr. Ohidujjaman Tuhin
Assistant Professor
Dept. of CSE , UIU

Divide and conquer algorithm: Quick Sort

Quick sort is an efficient sorting algorithm, serving as a systematic method for placing the elements of an array in a logical order. It is developed by Tony Hoare in 1960; it is still a very commonly used algorithm for sorting. Quick sort algorithm is a divide and conquer algorithm which sorts the given sequence in place meaning that it doesn't require extra storage. The basic idea is dividing the large list into two smaller sub-lists around an element which is called the pivot (partitioning element) such that elements in lower sub-list less than the pivot element and elements in higher sub-list is higher than the pivot element.

Mechanism of Quick Sort

Ascending order:

Let A be an array of several elements n. Here, n=13, if n>f, initially f=1, i=f+1 and j=n. If A[f]>A[i] then, i will increase by 1 (i=i+1) while j>=i. If A[j]>A[f] then j will decrease by 1(j=j-1) while j>=i. If (A[f]>A[i])) and (A[j]>A[f])) are false, and j>=i, then interchange A[i] and A[j] that is A[i]<->A[j] and increase i by 1 (i=i+1), and decrease j by 1(j=j-1). If j>=i results false then interchange A[f] and A[j] that is A[f]<->A[j], and for first sub-list n=j-1 and second sub-list f=j+1.

```
if(n>f)
        i=f+1;
        j=n;
----------
----------
end if
```

A[]

27	**99**	14	94	15	**25**	90
1	2	3	4	5	6	7
f=1	i=f+1				j=6	j=~~7~~ j=n

```
while(i<=j)
        while(A[f]>A[i])
                i=i+1;
        End while loop
        while(A[f]<A[j])
                j=j-1;
        end while loop
        if(i<=j)
        A[i]<->A[j]
        j=j-1; i=i+1;
        end if
end while loop
```

A

27	**25**	14	**94**	**15**	**99**	90
1	2	3	4	5	6	7
f=1	i=f+1			j=5	~~j=6~~	j=~~7~~
	i=~~2~~	~~i=3~~	i=4			

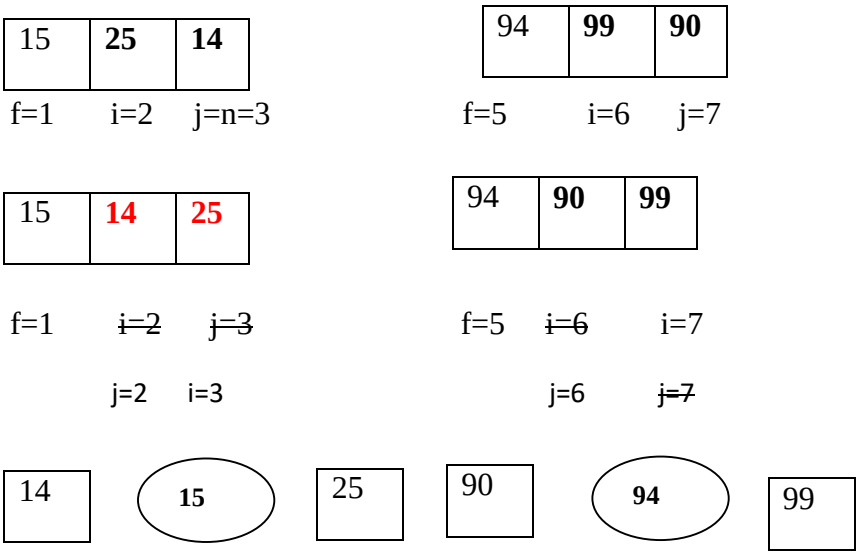
A

27	**25**	14	**15**	**94**	**99**	90
1	2	3	4	5	6	7
f=1	i=f+1		j=4	~~j=5~~	~~j=6~~	j=~~7~~
	i=~~2~~	~~i=3~~	~~i=4~~	i=5		

A

15	25	14	**27**	94	99	90
f=1	i=2	n=j-1	pivot	f=j+1	i=f+1	j=n

```
A[f]<->A[j]
Quick(A,f,j-1);
Quick(A,j+1,n);
```

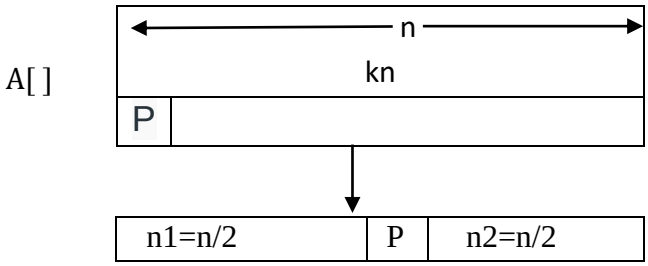


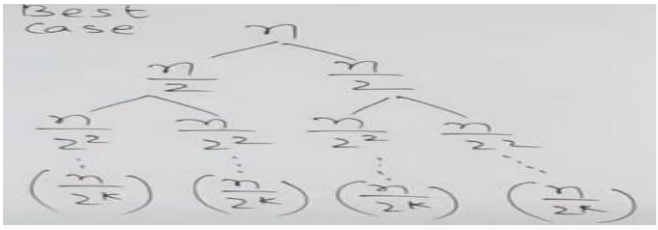
Pseudo code:

```
void  Quick(int A[],int f, int n){
   if(n>f)
         i=f+1;
         j=n;
         while(i<=j)
               while(A[f]>A[i])
                     i=i+1;
                end while loop
                 while(A[f]<A[j])
                     j=j-1;
                end while loop
                 if(i<=j)
                 A[i]<->A[j]
                 j=j-1; i=i+1;
                 end if
         end while loop
         A[f]<->A[j]
         Quick(A,f,j-1);
         Quick(A,j+1,n);
  end if
return;
}
```

```
#include<stdio.h>
#include<stdlib.h>
void Quick(int A[],int f, int n);
void main(){
            int A[20],i,n,f=1;
            fflush(stdout);
            printf("Enter the number of elemnts n:=");
            fflush(stdin);
            scanf("%d",&n);
          for(i=1;i<=n;i++)    {
            fflush(stdout);
           printf("Enter the elemnts of Array A[%d]:=",i);
            scanf("%d",&A[i]);
            }
          Quick(A,f,n);
          fflush(stdout);
           printf("After quick sort:=");
           for(i=1;i<=n;i++)   {
            fflush(stdin);
           printf("  %d",A[i]);
            }
printf("\n");
}
void Quick(int A[],int f, int n){
   int temp,i,j,temp1;
   if(n>f)    {
            i=f+1;
            j=n;
            while(i<=j)          {
                        while(A[f]>A[i]){
                           i=i+1;
                        }
                     while(A[f]<A[j]){
                                 j=j-1;
                        }
                        if(i<=j)   {
                           temp=A[i];
                           A[i]=A[j];
                           A[j]=temp;
                           i=i+1;
                           j=j-1;
                        }
               }
            temp1=A[f];
            A[f]=A[j];
            A[j]=temp1;
            Quick(A,f,j-1);
            Quick(A,j+1,n);
   }
return;
}
```

Time complexity of Quick sort Best case:





=1

$T(n)= T(n1)+T(n2)+ n$

$T(n) = 2 * T(n / 2) + n => 2^1 * T(n / 2^1) + 1.n$; [n1=n2=n/2]

$T(n) =2[2 * T((n / 2)/2) +n/2]+ 1.n$

$=4T *(n/4)+2n=> 2^2 * T(n / 2^2) + 2.n$

$=4[2T*((n / 4)/2) +n/4]+2.n$

$= 8T*(n/8)+3.n => 2^3 * T(n / 2^3) + 3.n$

$T(n) = 2^k * T(n / 2^k) + k.n$ ----------(1)

When each sub list contains single element=>

$n/2^k=1$

$2^k=n$

$k \log_2 2 = \log_2 n$ => k=logn

$T(n) = n * T(n / n) + n\log n$

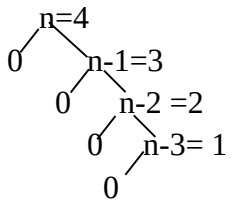
$=n.1+n \log n$

$=n\log n => O(n\log n)$

Quick sort worst case:

When elements are sorted

10	20	30	40

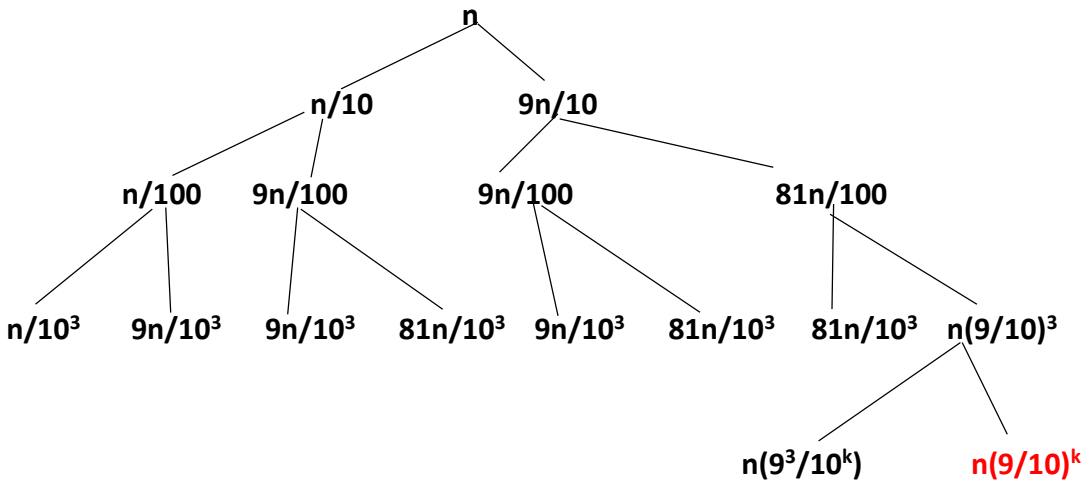


T(n)=1+2+3+----+n=n(n+1)/2

$=n^2/2+n/2$

$=O(n^2)$

Quick sort Average Case:



T(n)= T(n1)+T(n2)+ n.k1

=T(n/10)+T(9n/10)+n.k1

= [T(n/100)+T(9n/100)]+[T(9n/100)+T(81n/100)]+n.k2 ; [n.k2=n/10+9n/10+n.k1]

=[T(n/1000)+T(9n/1000)]+[2T(9n/1000)+2T(81n/1000)]+ T(81n/1000)+ T(729n/1000)+n.k3

; [n.k3=n.k2+n/100+2(9n/100)+81n/100]

$=T(n/10^3)+3T(9n/10^3)+3T(81n/10^3)+T(9/10)^3.n+n.k3$

$=T(n/10^k)+3T(9n/10^k)+3T(81n/10^k)+T(9/10)^k.n+n.k$----------(ii)

; $[n.k=n.(k-1)+n/10^{k-1}+3(9n/10^{k-1})+3(81n/10^{k-1}+n(9/10)^{k-1}]$

$(9/10)^k.n=1=>10^k=9^kn=>(10/9)^k=n=>$ $k \log_{10/9} 10/9 = \log_{10/9} n$ $=>k=\log_{10/9} n$

From (ii) we have >

T(n)=T(1)+n.logn =O(nlogn)

Divide and conquer algorithm: Merge sort

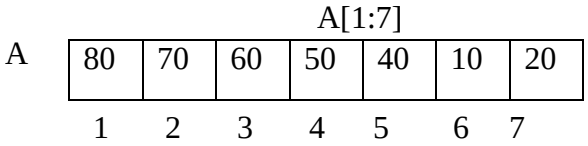
Merge Sort

Merge sort is a sorting algorithm that sorts data items into ascending or descending order, which comes under the category of comparison-based sorting. Merge sort is the divide-and-conquer algorithm to sort a given sequence of data items, which can be described as follows:

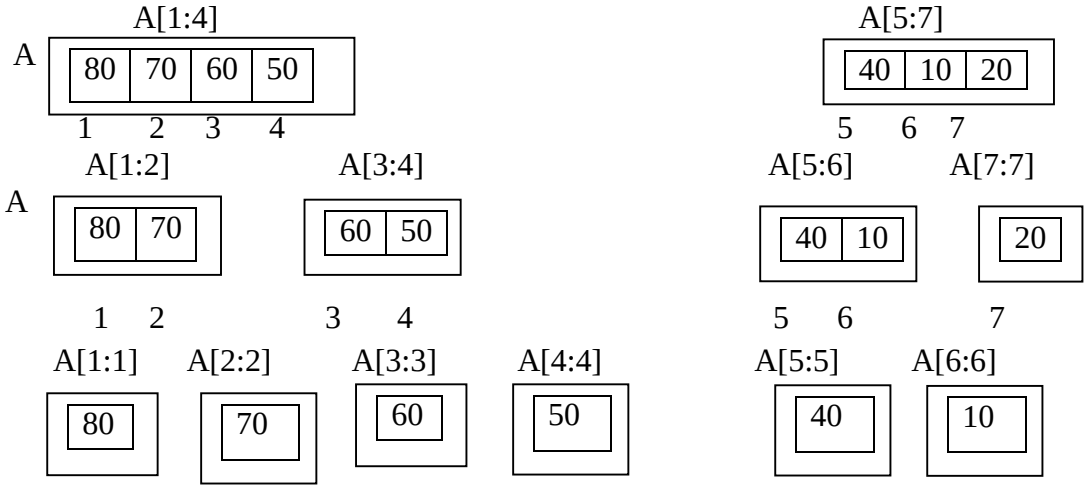
1. Recursively split the sequence into two halves (i.e. subsequences) until the subsequence contains only a single data item (i.e. singleton subsequence)
2. Now, recursively merge these subsequences back together preserving their required order (i.e. ascending or descending order).

Mechanism for dividing elements:

Let, A is an array of n=7 elements. Here, A[j : k] represents an array indicating the location of first(j) and last(k) element.



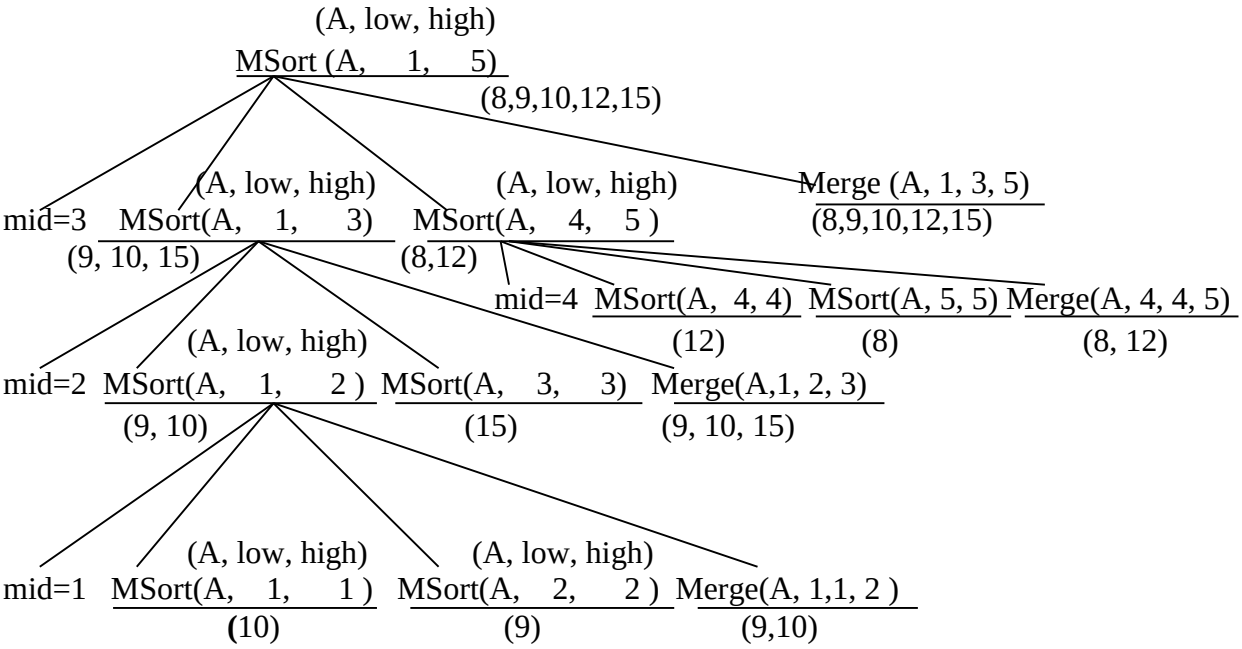
After first division we have two sub-lists 1 to 4 and 5to 7 that is each sub-list will be A [1:4] and A [5:7]. The elements in A [1:4] are splitted of size 2 (A [1:2]) and 2(A [3:4]). The two values in A [1:2] are splitted finally into one-element sub-lists and now the merging begins. Similarly the remaining sub-lists are divided for further sub-lists.



Merge Sort: **Ascending order**

A	10	9	15	12	8
	1	2	3	4	5

Let, n=5, low=1, high=n



Part 2:

```
void MSort(int A[],int low,int high)
        int mid;
        if(high>low)
                mid=(low+high)/2;
                MSort(A,low,mid);
                MSort(A,mid+1,high);
                Merge(A,low,mid,high);
        end if
end MSort( )
```

Part 3:

```
void Merge(int A[],int low,int mid,int high)
                i=low;
                j=mid+1;
                k=low;
                while((i<=mid)&&(j<=high))
                        if(A[j]<A[i])
                          B[k]=A[ j];
                           j=j+1;
                           k=k+1;
                        else
                          B[k]=A[i];
                           i=i+1;
                           k=k+1;
                        end if
                end while
                if(j>high)
                        for(p=i;p<=mid;p++)
                                B[k]=A[p];
                                k=k+1;
                        end for loop
                end if
                if(i>mid)
                        for(p=j; p<=high; p++)
                                B[k]=A[p];
                                k=k+1;
                        end for loop
                end if
                for(q=low; q<=high; q++)
                        A[q]=B[q];
                end for loop
end Merge( )
```

Part 1:

```
void main( )
{
        int low=1;
        int high=n;
        MSort(A,low,high);
        for(i=1;i<=n;i++)
        {
                printf(" %d",A[i]);
        }
}
```

Part 2:

```
void MSort(int A[],int low,int high)
        int mid;
        if(high>low)
                mid=(low+high)/2;
                MSort(A,low,mid);
                MSort(A,mid+1,high);
                Merge(A,low,mid,high);
        end if
end MSort( )
```

Part 3:

```
void Merge(int A[],int low,int mid,int high)
        i=low;
        j=mid+1;
        k=low;
        while((i<=mid)&&(j<=high))
                if(A[j]<A[i])
                  B[k]=A[j];
                   j=j+1;
                   k=k+1;
                else
                  B[k]=A[i];
                   i=i+1;
                   k=k+1;
                end if
        end while
        if(j>high)
                for(p=i;p<=mid;p++)
                        B[k]=A[p];
                        k=k+1;
                end for loop
        end if
        if(i>mid)
                for(p=j; p<=high; p++)

                        B[k]=A[p];
                        k=k+1;
                end for loop
        end if
        for(q=low; q<=high; q++)
                A[q]=B[q];
        end for loop
end Merge( )
```

```
#include<stdio.h>
void MSort(int A[],int low,int high);
void Merge(int A[],int low,int mid,int high);
int B[20];
void main(){
        int A[20],n,i;
        fflush(stdout);
        printf("Enter the number of elements n:=");
        fflush(stdin);
        scanf("%d",&n);
        for(i=1;i<=n;i++)  {
```

```
                fflush(stdout);
                printf("Enter %d element of arrary A[]:=",i);
                fflush(stdin);
                scanf("%d",&A[i]);
            }
            int low=1;
            int high=n;
            MSort(A,low,high);
        fflush(stdout);
        printf("After merge sort:=");
            for(i=1;i<=n;i++)  {
                fflush(stdout);
                printf(" %d",A[i]);
            }
            printf("\n");
}
void MSort(int A[],int low,int high){
            int mid;
            if(high>low)
            {
                mid=(low+high)/2;
                MSort(A,low,mid);
                MSort(A,mid+1,high);
                Merge(A,low,mid,high);
            }
}
void Merge(int A[],int low,int mid,int high){
            int i,j,k,p,q;
            i=low;
            j=mid+1;
            k=low;
            while((i<=mid)&&(j<=high))      {
                if(A[j]<A[i]) {
                   B[k]=A[j];
                   j=j+1;
                   k=k+1;
                }
                else {
                   B[k]=A[i];
                   i=i+1;
```

```
                    k=k+1;
                }
            }
            if(j>high) {
                for(p=i;p<=mid;p++)  {
                        B[k]=A[p];
                        k=k+1;
                }
            }
            if(i>mid)   {
                for(p=j;p<=high;p++)
                {
                        B[k]=A[p];
                        k=k+1;
                }
            }
            for(q=low;q<=high;q++)  {
                A[q]=B[q];
            }
}
```

Complexity of merge sort:

```
void MSort(int A[],int low,int high)
        int mid;
        if(high>low)
                mid=(low+high)/2;
                MSort(A,low,mid);      → n/2
                MSort(A,mid+1,high);  → n/2
                Merge(A,low,mid,high); →n
        end if
end MSort( )
```

We get T(n) from the Msort algorithm=>

$$T(n)=\begin{cases} \text{Constant} & n=1 \\ 2T(n/2)+n & n>1 \end{cases}$$

Master method:

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

where f(n)= $O(n^k (logn)^p)$; if a>=1 and b>1

Case2: if log_a^b =k

if p>-1 $O(n^k (logn)^{p+1})$

Problem:

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

We have a=2; b=2; $log_2^2 = 1$

Therefore, f(n)=$n^1 (logn)^0$ =n ; k=1; p=0

Case2:

log_a^b =k is true; p>-1; $O(n^1 (logn)^{0+1})$ =O(nlogn) (solved)