Design and Analysis of Algorithms

L13: MergeSort & Quicksort

Dr. Ram P Rustagi
Sem IV (2019-H1)
Dept of CSE, KSIT/KSSEM
rprustagi@ksit.edu.in

Resources

- Text book 1: Levitin (Mergesort)
-

MergeSort

- Problem: Given a set of N elements, sort the elements in ascending (or descending) order
  - Assume that these elements are in an array of size N
- Approaches
  - Divide and Conquer approach

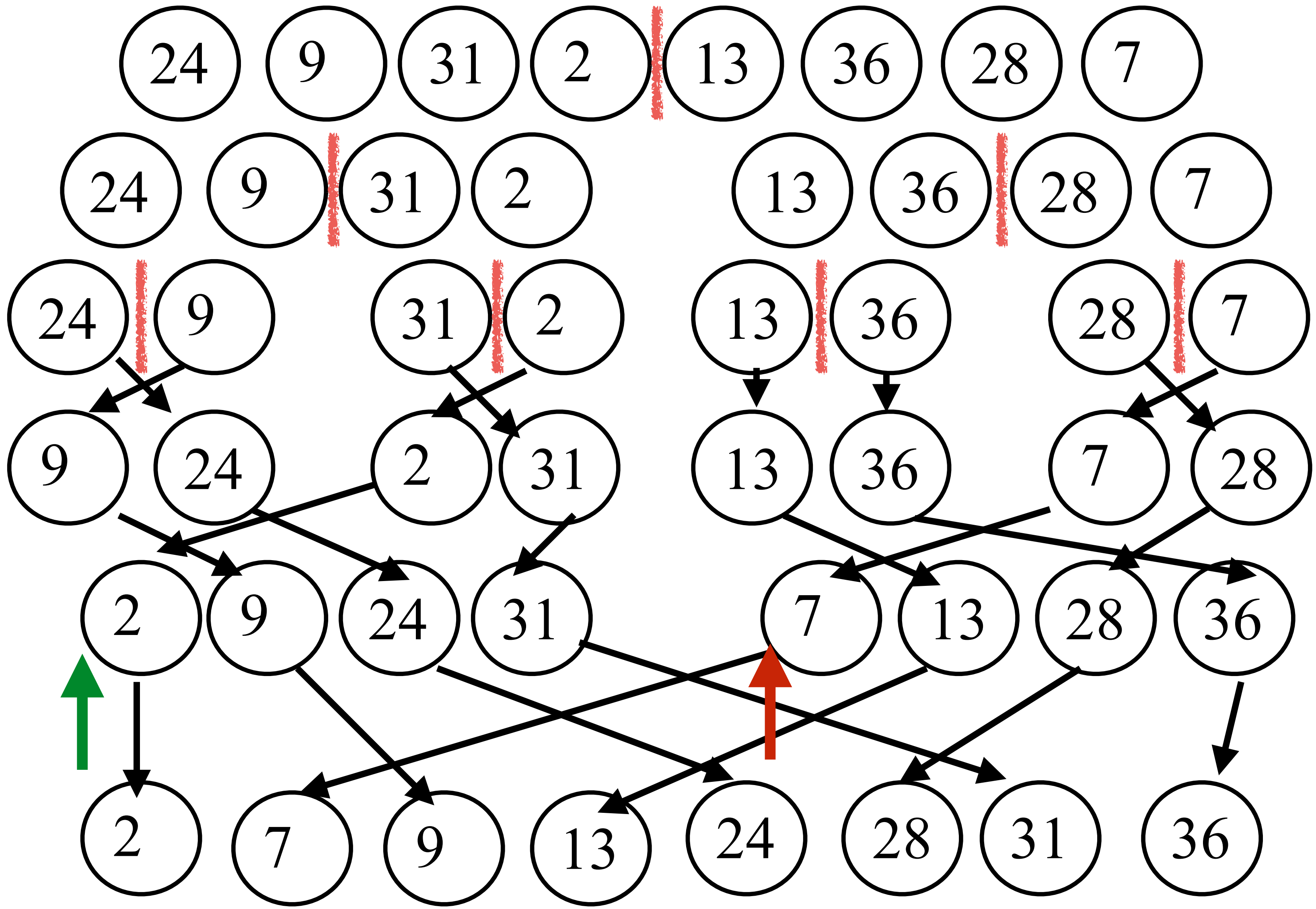
Sort Algorithms

- Bubble sort
- Selection sort
- Insertion sort
- Mergesort
- Quicksort
- Shell sort
- Heap sort
- Radix sort

MergeSort

- Basic idea
  - Take two sorted list and merge them into a single sorted list.
- Approach
  - Keep dividing the elements into (almost) equal half size (recursively) till sublist becomes of size 1
  - List of size `1` is sorted by default
  - Merge the sorted lists and keep repeating (recursively back)
  - When all the lists are merged, all elements are sorted.

MergeSort Example



MergeSort

- Split array A[1:n] into about equal halves
  - Make copies of each half in arrays B and C
- Sort arrays B and C recursively
- Merge sorted arrays B and C into A as follows:
  - Repeat until one of the arrays becomes empty
    - Compare the first elements of the remaining unprocessed portions of the arrays
    - Copy the smaller of the two into A,
      - Increment the index of the array (smaller)
  - Once all elements in one of the arrays are copied
    - Copy the remaining unprocessed elements from the other array into A.

Algo: MergeSort

- Algo `MergeSort(1,n,A[])`

```
#Sort array A recursive by merging
#i/p: unsorted array A[1:n]
#o/p: sorted array A[1:n]
if n>1, then
  copy A[1:n/2] to B[1:n/2]
  copy A[n/2+1:n] to C[1:n/2]
  Mergesort(1,n/2,B) #recursive
  Mergesort(1,n/2,C) #recursive
  Merge(B,C,A) # merge two arrays
```

Algo: MergeSort

- **Algo** `Merge(B[1:p],C[1:q],A[1:p+q])`

```
#maintain one index for each array
i←1; j←1; k←1;
while (i<p+1) and (j<q+1) do
   if (B[i]≤C[j]), then
      A[k] ← B[i]
      i ← i+1
   else
      A[k] ← C[j]
      j ← j+1
   k ← k+1
if (i > p) then #B has been fully copied to A
   copy C[j:q] to A[k:p+q]
else
   copy B[i:p] to A[k:p+q]
```

MergeSort: Analysis

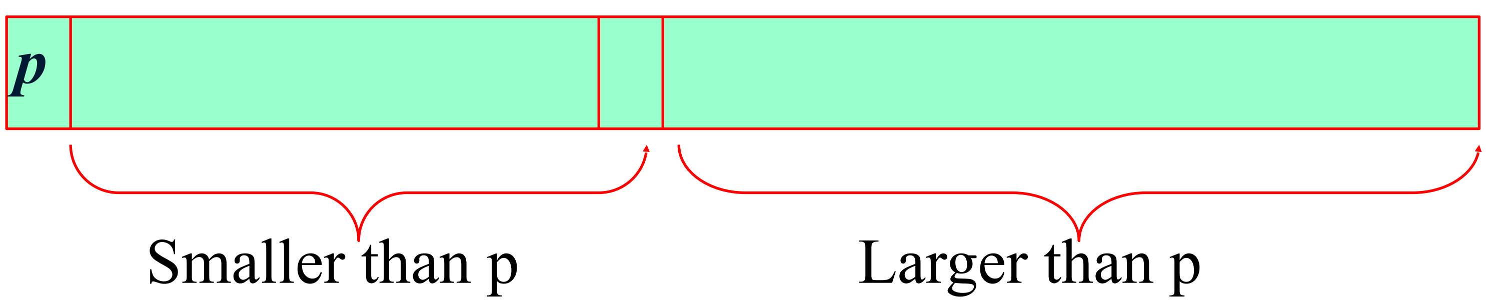
- Each step of Mergesort
  - Two recursive invocations of size $n/2$: $2T(n/2)$
  - Merging of two n/2 array into one array of size n
    - Time complexity: n
- Recurrence relation for time complexity becomes

$$
\begin{aligned}
T(n) &= 2T(n/2) + n \\
&= 2(2T(n/4)+n/2)+n = 2^2T(n/2^2)+n+n \\
&= \ldots \\
&= 2^kT(n/2^k)+n+\ldots(\log_2 n \text{ times}) \\
&= n*T(1)+n\log_2 n = n + n\log_2 n \\
&= \Theta(n\log_2 n)
\end{aligned}
$$

- Space complexity $= \Theta(n)$

QuickSort

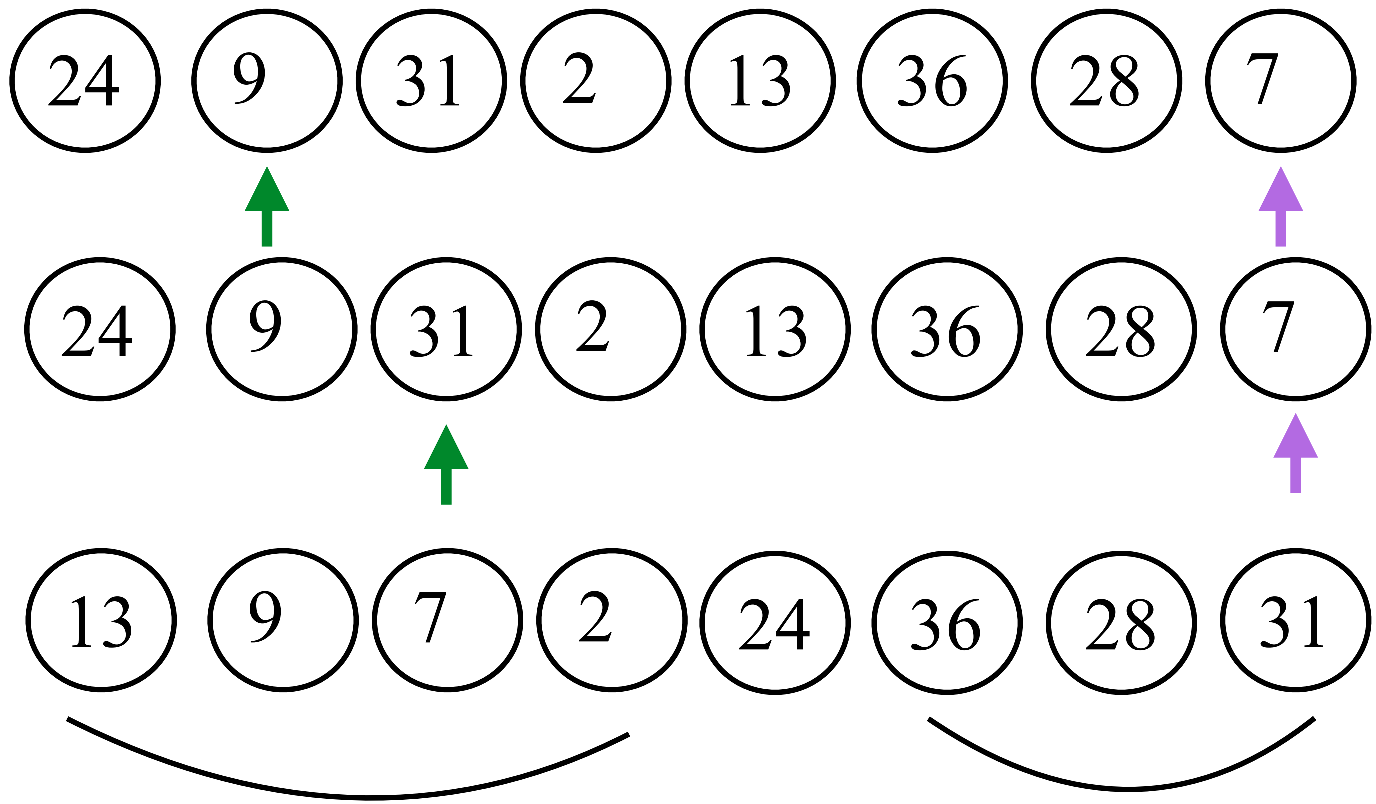
- A highly efficient algorithm
- Divides input array in smaller arrays using a specified value (pivot)
  – One array contains smaller values
  – Other array contains larger values
- Exchange the pivot with last element in first array
  – pivot is in its final position
- Sort the sub arrays recursively



QuickSort Algorithm: Steps

- Select a *pivot* (partitioning element) e.g. 1st element
- Rearrange the array as follows
  - All elements in first s positions are $\leq$ pivot
  - All elements in remaining n-s positions $\geq$pivot
- Repeat the process

Quicksort



Quicksort

- **Algo** `quicksort(l,r,A[])`

```
#i/p:array[l,r] defined by left and right indices
#o/p: array[l,r] sorted in ascending order

if l<r
   s ← partition(l,r,A[l,r])
   Quicksort(l,s-1,A[l,s-1])
   Quicksort(s+1,r,A[s+1,r])
```

Quicksort

- **Algo** `partition(l,r,A[])`

```
p←A[l]
i←l; j←r+1
repeat
  repeat
    i←i+1
  until A[i]≥p or i>r
  repeat
    j←j-1
  until A[j]≤p or j==l
  swap(A[i],A[j])
until i≥j
swap(A[i],A[j]) #undo last swap when i≥j
swap(A[l],A[j])
return j
```

Analysis: QuickSort

- Best case: split is approximately in the middle
  $T(n) = 2T(n/2) + \Theta(n)$
  $= \Theta(n\log_2 n)$
- Worst case: split is at the end e.g. sorted array
  $T(n) = T(n-1) + \Theta(n)$
  $= \Theta(n^2)$
- Average case:
  $T(n) = \Theta(n\log_2 n)$
- Improvements (20-25%)
  – Better pivot selection : take median
  – Use insertion sort on smallar array size
  – Eliminate recursion and use iteration

Summary

- Mergesort
  – Not in place sort
- Quicksort
  – In place sort
  – Practically used on large data