Divide & Conquer

Prepared by Pawan K. Mishra

A general theme…

$$\text{Prob}(A[1..n]) = \begin{cases} \text{Combine}\ (\text{Prob}(A[1..m]),\ \text{Prob}(A[1..m+1])), & \text{if } n > n_0 \\ \text{Basis solution}, & \text{if } n = n_0 \end{cases}$$

A general theme…

$$\text{Prob}(A[1..n]) = \begin{cases} \text{Combine}(\text{Prob}(A[1..m]), \text{Prob}(A[1..m+1])), & \text{if } n > n_0 \\ \text{Basis solution}, & \text{if } n = n_0 \end{cases}$$

m = n/2, n/4 e.g.. Merge Sort, Binary Search…

m depends on the partition e.g.. Quick Sort

Meaning of D&C

Divide the problem into a number of subproblems that are smaller instances of the same problem.

Conquer the subproblems by solving them recursively. If the subproblem sizes are small enough, however, just solve the subproblems in a straightforward manner.

Combine the solutions to the subproblems into the solution for the original problem

Order Statistics-I

P1. Finding the minimum of n elements in an array.

P2. Finding the maximum and the minimum of n elements in an array.

P3. Finding the maximum and the second maximum of n elements in an array.

P1. Finding the minimum of n elements in an Array

```
fmin(A, begin, end)
    If (begin==end) return A[begin]
    Else
        mid= ⌊(begin+end-1)/2)⌋
    Return min{fmin(begin,mid), fmin(mid+1, end)}
```

P1. Finding the minimum of n elements in an array

```
fmin(A, begin, end)
    If (begin==end) return A[begin]
    Else
        mid= ⌊(begin+end-1)/2)⌋
    Return min{fmin(begin,mid), fmin(mid+1, end)}
```

T(n/2) T(n/2)

P1. Finding the minimum of n elements in an array

fmin(A, begin, end)

If (begin==end) return A[begin]

Else

mid= $\lfloor$(begin+end-1)/2)$\rfloor$

Return min{fmin(begin,mid), fmin(mid+1, end)}

1 T(n/2) T(n/2)

P1. Finding the minimum of n elements in an array

fmin(A, begin, end)

If (begin==end) return A[begin]

Else

mid= ⌊(begin+end-1)/2⌋

Return min{fmin(begin,mid), fmin(mid+1, end)}

1 T(n/2) T(n/2)

$T(n)= 1 + T(n/2)+ T(n/2)= 2T(n/2)+1$

P1. Finding the minimum of n elements in an array

fmin(A, begin, end)

If (begin==end) return A[begin] → T(1)=1

Else

mid= $\lfloor$(begin+end-1)/2)$\rfloor$

Return min{fmin(begin,mid), fmin(mid+1, end)}

1 T(n/2) T(n/2)

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

Can we divide in other fractions?

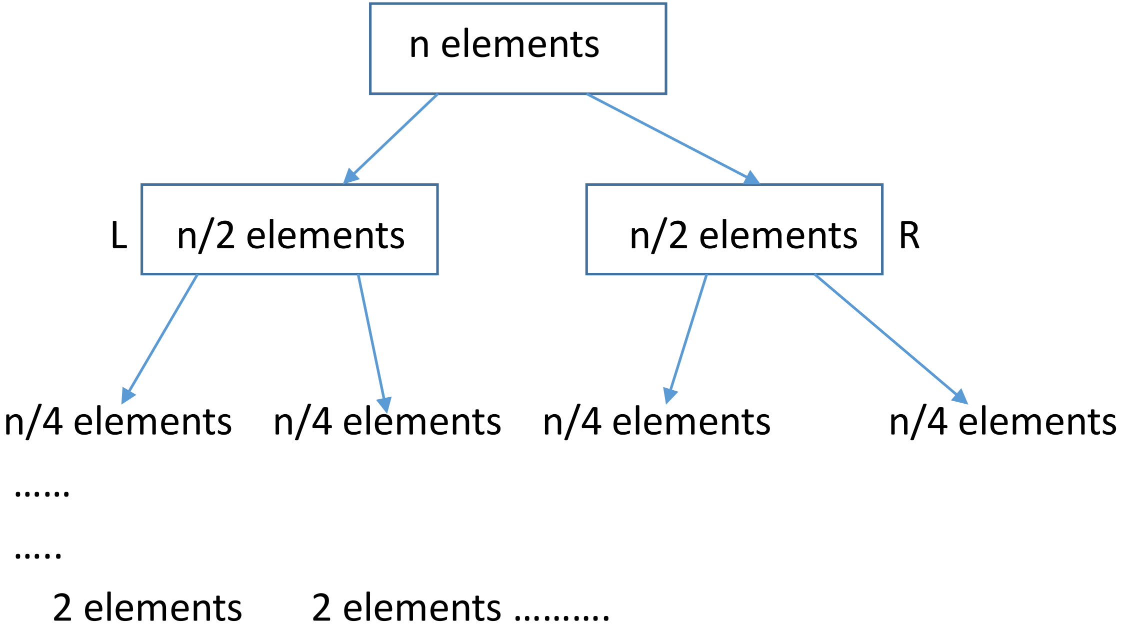
n/3, 2n/3

$$T(n)=T(n/3)+T(2n/3)+1 \text{ ??}$$

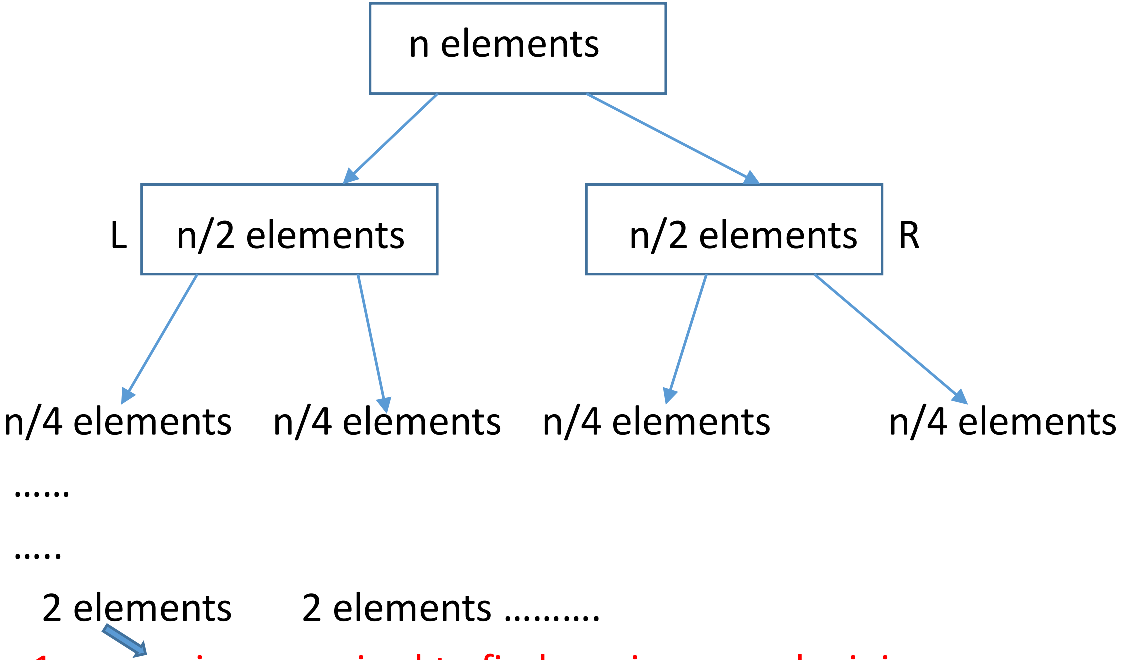
n/3, n/3, n/3

$$T(n)=3T(n/3)+1 \text{ ??}$$

P2. Finding the maximum and the minimum of n elements in an array.

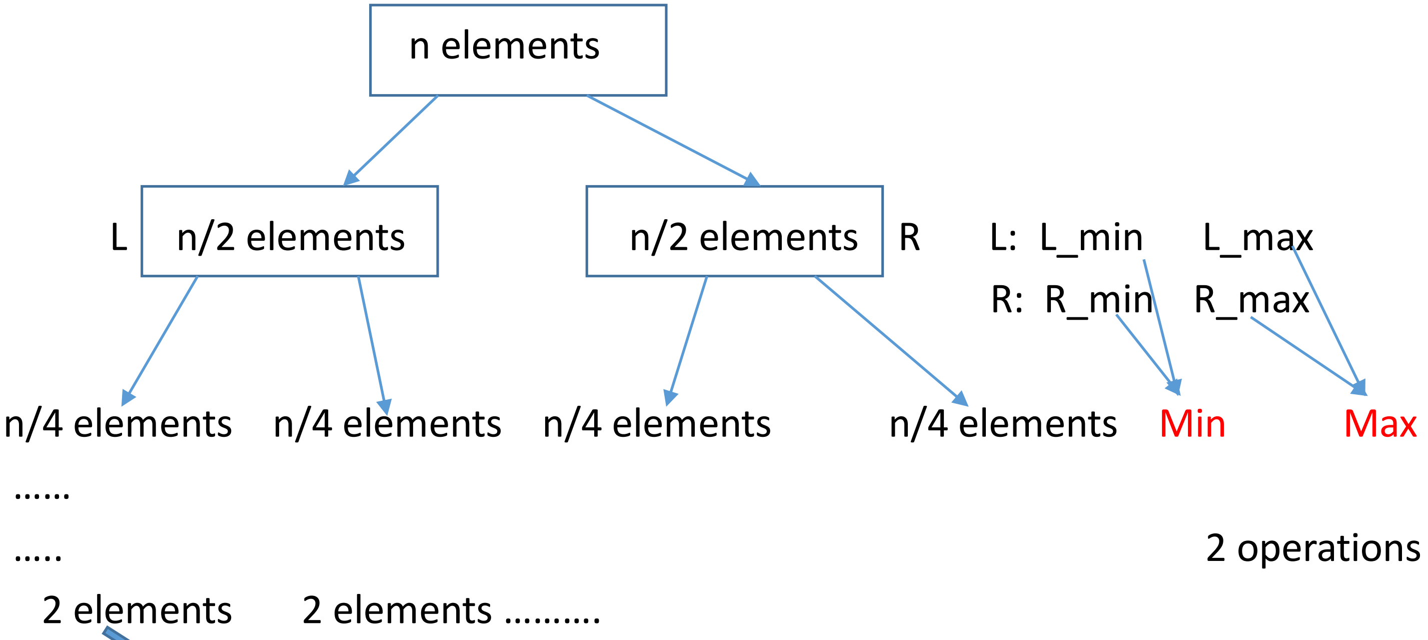


P2. Finding the maximum and the minimum of n elements in an array.



1 comparison required to find maximum and minimum.

P2. Finding the maximum and the minimum of n elements in an array.



1 comparison require to find maximum and minimum.

P2. Finding the maximum and the minimum of n elements in an array.

n elements

L n/2 elements   n/2 elements R

L: L_min   L_max

R: R_min   R_max

n/4 elements   n/4 elements   n/4 elements   n/4 elements

Min   Max

……

…..

2 operations

2 elements   2 elements ………

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

1 comparison require to find maximum and minimum.

$T(2)=1$

P3. Finding the maximum and the second maximum of n elements in an array.

n elements

L n/2 elements

n/2 elements R

L: L_max L_smax

R: R_max R_smax

n/4 elements n/4 elements n/4 elements n/4 elements

Max SMax

2 operations

……

…..

2 elements 2 elements ………

1 comparison require to find maximum and second maximum.

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

$T(2)=1$

P4. Binary Search

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

P5. Merge Sort

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

$T(2)=\text{constant}$

Merging two sorted arrays ??

Refreshment-I

Consider the following modification to merge sort algorithm: divide the input arrays into thirds(rather than halves), recursively sort each third and finally combine the results using three way Merge subroutine.

What is the running time taken by this successive merging algorithm, as a function of n , ignoring constant factors and lower order terms?

a) n

b) $n\log n$

c) $n(\log n)^2$

d) $n^2\log n$

Refreshment-II

Suppose you are given k sorted arrays, each with n elements, and you want to combine them into a single array of kn elements.

One approach is to use Merge subroutine like wise, first merging the first two arrays, then merging the result with the third array, then with the fourth array, and so on until you merge in the k-th array.

What is the running time taken by this successive merging algorithm, as a function of n and k, ignoring constant factors and lower order terms?

a) nlogk
b) nk
c) nklogk
d) nklogn
e) nk^2
f) n^2k

Template: D&C

```
RecursiveD&C(A[1..n])
        solve base case (n_0)
         if (n > n_0)
             do something(A)
             A_1= extract(A)
             A_2= extract(A)
             B_1= RecursiveD&C(A_1)
             B_2= RecursiveD&C(A_2)
              B  =rebuild(B_1, B_2)
           Return B
```

Template: D&C

```
RecursiveD&C(A[1..n])
        solve base case (n_0)                  ----  in constant time
         if (n > n_0)
             dosomething(A)                     other steps depend on problem
             A_1= extract(A)
             A_2= extract(A)
             B_1= RecursiveD&C(A_1)
             B_2= RecursiveD&C(A_2)
              B  =rebuild(B_1, B_2)
           Return B
```

Template: D&C

RecursiveD&C(A[1..n])

solve base case (n_0) ---- in constant time

if (n > n_0)

dosomething(A) other steps depend on problem

A_1= extract(A)

A_2= extract(A)

B_1= RecursiveD&C(A_1)

B_2= RecursiveD&C(A_2)

B =rebuild(B_1, B_2)

Return B

dosomething(A) findmin, findmid, findmax, findmedian

extract(A) ---- divide(A), partition(A)

rebuild(B_1,B_2)– merge, compare

Template: D&C

RecursiveD&C(A[1..n])

solve base case (n_0) ---- in constant time

if (n > n_0)

dosomething(A) other steps depend on problem

A_1= extract(A)

A_2= extract(A)

B_1= RecursiveD&C(A_1)

B_2= RecursiveD&C(A_2)

B =rebuild(B_1, B_2)

Return B

T(n)= contant, Base Case

= aT(f(n))+bT(g(n))+.. +ExtraWork

dosomething(A) findmin, findmid findmax, findmedian

extract(A) ---- divide(A), partition(A)

rebuild(B_1,B_2)– merge, compare

P6. Compute a^n

I/P: a and an intger n

O/P: a^n.

P7. Bisection method: root finding

I/P: A polynomial function f(x), and +ve integer n

O/P:integer root "r" s.t. f(r)=0, where $0 \leq r < n$

e.g. $f(x)=x^2-x-12=0$, $n=8$

$r=4$

Prove via induction

$T(n)=2T(n/2)+ c\,n$

$T(2)=\text{constant}$

Thm: $T(n)=O(n)$ A guess… so need to prove via induction.

Proof: (by induction)

Base Case: $T(2)=\text{constant} = O(1)$

Ind Hyp: Assume statement is true for $k<n$, i.e., $T(k)=O(k)$. [Strong Induction]

Ind Step: To prove theorem for $T(n)$, using ind hyp $T(n/2)=O(n/2)$

$T(n/2) \le dn/2$

$$T(n)= 2T(n/2)+ c\,n$$

$$\le 2d(n/2)+ c\,n \quad \text{(by Hypothesis)}$$

$$=(c+d) = O(n)$$

Prove via induction

$T(n)=2T(n/2)+ c\, n$

$T(2)$=constant

Thm: $T(n)=O(n)$ A guess… so need to prove via induction.

Proof: (by induction)

Base Case: $T(2)$=constant $= O(1)$

Ind Hyp: Assume statement is true for $k<n$, i.e., $T(k)=O(k)$. [Strong Induction]

Ind Step: To prove theorem for $T(n)$, using ind hyp $T(n/2)=O(n/2)$

$T(n/2) \le dn/2$

$$\begin{aligned} T(n) &= 2T(n/2)+ c\, n \\ &\le 2d(n/2)+ c\, n \quad \text{(by Hypothesis)} \\ &=(c+d) = O(n) \end{aligned}$$

can you find the the mistake ?

Prove via induction

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

$T(2)$=constant

Thm: $T(n) \le b\ n$ A guess… so need to prove via induction.

Proof: (by induction)

Base Case: $T(2)$=constant $\le b.2$ [here, constant/2 $\le b$]

Ind Hyp: Assume statement is true for $k<n$, i.e., $T(k) \le b\ k$. [Strong Induction]

Ind Step: To prove theorem for $T(n)$, using ind hyp $T(n/2) \le b\ n/2$

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

$\le bn$

Prove via induction

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

$T(2)=\text{constant}$

Thm: $T(n) \le b\ n$ A guess… so need to prove via induction.

Proof: (by induction)

Base Case: $T(2)=\text{constant} \le b.2=b$ [here, constant/2 $\le b$]

Ind Hyp: Assume statement is true for $k<n$, i.e., $T(k) \le bk$. [Strong Induction]

Ind Step: To prove theorem for $T(n)$, using ind hyp $T(n/2) \le b\ n/2$

$$
\begin{aligned}
T(n) &= 2T(n/2)+ c\ n \\
&\le 2b(n/2)+ c\ n \quad \text{(by Hypothesis)} \\
&= (b+c)n \\
&\ne bn
\end{aligned}
$$

Prove via induction

$T(n)=2T(n/2)+ c\,n$

$T(1)=\text{constant}$

Thm: $T(n) \le b\,n$ A guess… so need to prove via induction

Proof: (by induction)

Base Case: $T(1)=\text{constant} \le b.2=b$ [here, constant/2 $\le b$]

Ind Hyp: Assume statement is true for $k<n$, i.e., $T(k) \le bk$. [Strong Induction]

Ind Step: To prove theorem for $T(n)$, using ind hyp $T(n/2) \le b\,n/2$

$T(n)= 2T(n/2)+ c\,n$

$\le 2b(n/2)+ c\,n$ (by Hypothesis)

$= (b+c)n$

$\ne bn$ **So, the guess is wrong**

Prove via induction

$T(n)=2T(n/2)+ c\, n$

$T(2)=\text{constant}$

Thm: $T(n) \le b\, n \log n$ A guess and we later find 'b'. Need to prove via induction.

Proof: (by induction)

Base Case: $T(2)=\text{constant} \le b.2\log 2=2b$ [here, constant/2 $\le b$]

Ind Hyp: Assume statement is true for $k<n$, i.e., $T(k) \le bk$. [Strong Induction]

Ind Step: To prove theorem for $T(n)$, using ind hyp $T(n/2) \le b\, n/2 \log n/2$

$T(n)= 2T(n/2)+ c\, n$

$\le 2b(n/2) \log n/2 + c\, n$ (by Hypothesis)

$= 2\, b\, n/2 \log (n/2) +cn = bn \log n - bn \log 2+cn = bn \log n - bn +cb$

$=bn\log n-(bn-cn) \le bn\log n$

(if $bn-cn>0$, $b>c$)

Prove via induction

$T(n)=2T(n/2)+ c\, n$

$T(2)=\text{constant}$

Thm: $T(n) \le b\, n \log n$ A guess and we later find ‘b’. Need to prove via induction.

Proof: (by induction)

Base Case: $T(2)=\text{constant} \le b.2\log 2=2b$ [here, constant/2 $\le b$]

Ind Hyp: Assume statement is true for $k<n$, i.e., $T(k) \le bk$. [Strong Induction]

Ind Step: To prove theorem for $T(n)$, using ind hyp $T(n/2) \le b\, n/2 \log n/2$

$T(n)= 2T(n/2)+ c\, n$

$\le 2\, b\, (n/2) \log n/2+ c\, n$ (by Hypothesis)

$= 2\, b\, n/2 \log (n/2) +c\, n = bn \log n - bn \log 2+cn = bn \log n - bn +cb$

$=bn\log n-(bn-cn) \le bn\log n$

b =max{constant/2, c} – this guess will work (if $bn-cn>0$, $b>c$)

P8. Not D&C Problem, but an interesting problem

Let given two sorted array A and B, find number of pairs (i, j) such that A[i] > B[j].

For eg. A= [3,6,7,9] B=[1, 2, 5, 8, 10]

Output= 2+3+3+4=12

Let given two sorted arrays A and B, find the number of pairs (i, j) such that A[i] > B[j].

For eg. A= {3,6,7,9} B={1, 2, 5, 8, 10}

Output= 2+3+3+4=12

Naïve approach: check for each element of A, traverse in B.

$O(n^2)$

Merge and Count

A= [3,6,7,9] B=[1, 2, 5, 8, 10]

Lets Merge to get sorted order

A= [3,6,7,9] B=[1, 2, 5, 8, 10]

Big Sorted Array= [1,2,3,5,6,7,8,9,10]

Merge and Count

A= {3,6,7,9} B={1, 2, 5, 8, 10}

Lets Merge to get sorted order

Big Sorted Array= [1,2,3,5,6,7,8,9,10]

Merge and Count

A= {3,6,7,9} B={1, 2, 5, 8, 10}

Lets Merge to get sorted order

Big Sorted Array= [1,2,3,5,6,7,8,9,10] – element in red came from array B.

Merge and Count

A= {3,6,7,9} B={1, 2, 5, 8, 10}

Lets Merge to get sorted order

Big Sorted Array= [1,2,3,5,6,7,8,9,10] – elements in red came from the array B.

Lets see the status of array A, when an element came from the array B in the big sorted array.

Merge and Count

A= {3,6,7,9} B={1, 2, 5, 8, 10}

Lets Merge to get sorted order

Big Sorted Array= [1,2,3,5,6,7,8,9,10] – element in red came from array B.

Lets see the status of array A, when element entered from B in the big sorted array.

When 1 entered– How many elements are there in array A which are not traversed “fully” by the pointer of A?

Merge and Count

A= {3,6,7,9} B={1, 2, 5, 8, 10}

Lets Merge to get sorted order

Big Sorted Array= [1,2,3,5,6,7,8,9,10] – element in red came from array B.

Lets see the status of array A, when element entered from B in the big sorted array.

When 1 entered– How many elements are there in array A which are not traversed fully by the pointer of A? 4

Merge and Count

A= {3,6,7,9} B={1, 2, 5, 8, 10}

Lets Merge to get sorted order

Big Sorted Array= [1,2,3,5,6,7,8,9,10] – element in red came from array B.

Lets see the status of array A, when element entered from B in the big sorted array.

When 2 entered– How many elements are there in array A which are not traversed fully by the pointer of A? 4

Merge and Count

A= {3,6,7,9} B={1, 2, 5, 8, 10}

Lets Merge to get sorted order

Big Sorted Array= [1,2,3,5,6,7,8,9,10] – element in red came from array B.

Lets see the status of array A, when element entered from B in the big sorted array.

When 5 entered– How many elements are there in array A which are not traversed fully by the pointer of A? 3

Merge and Count

A= {3,6,7,9} B={1, 2, 5, 8, 10}

Lets Merge to get sorted order

Big Sorted Array= [1,2,3,5,6,7,8,9,10] – element in red came from array B.

Lets see the status of array A, when element entered from B in the big sorted array.

When 8 entered– How many elements are there in array A which are not traversed fully by the pointer of A? 1

Merge and Count

A= {3,6,7,9} B={1, 2, 5, 8, 10}

Lets Merge to get sorted order

Big Sorted Array= [1,2,3,5,6,7,8,9,10] – element in red came from array B.

Lets see the status of array A, when element entered from B in the big sorted array.

When 10 entered– How many elements are there in array A which are not traversed fully by the pointer of A? 0

When 1 entered– How many elements are there in array A which are not traversed fully by the pointer of A? 4

When 2 entered– How many elements are there in array A which are not traversed fully by the pointer of A? 4

When 5 entered– How many elements are there in array A which are not traversed fully by the pointer of A? 3

When 8 entered– How many elements are there in array A which are not traversed fully by the pointer of A? 1

When 10 entered– How many elements are there in array A which are not traversed fully by the pointer of A? 0

When 1 entered– How many elements are there in array A which are not traversed fully by the pointer of A? 4

When 2 entered– How many elements are there in array A which are not traversed fully by the pointer of A? 4

When 5 entered– How many elements are there in array A which are not traversed fully by the pointer of A? 3

When 8 entered– How many elements are there in array A which are not traversed fully by the pointer of A? 1

When 10 entered– How many elements are there in array A which are not traversed fully by the pointer of A? 0

12

Merge and Count

Array got sorted and we got the count in O(n)

P9: Counting Inversion

I/P: Given an array A of distinct integers.

O/P: The number of inversions of A is the number of pairs (i,j) of array A with $i<j$ and $A[i] > A[j]$.

P8: Counting Inversion

I/P: Given an array A of distinct integers.

O/P: The number of inversions of A is the number of pairs (i,j) of array A with i<j and A[i] > A[j].

A=[1, 3, 5, 4, 2]

0 + 1 + 2 + 1 + 0 = 4 pairs (3,2), (5,4) (5,2) (4,2)

Application of inversion

My web-series order:

Breaking Bad Prison Break Panchayat Game of Thrones Mirzapur

1 2 3 4 5

Application of inversion

My web-series order:

Breaking Bad	Prison Break	Panchayat	Game of Thrones	Mirzapur
1	2	3	4	5

Friend's web-series order:

Breaking Bad	Panchayat	Mirzapur	Game of Thrones	Prison Break
1	3	5	4	2

Application of inversion

My web-series order:

Breaking Bad Prison Break Panchayat Game of Thrones Mirzapur

1 2 3 4 5

Friend's web-series order:

Breaking Bad, Panchayat, Mirzapur , Game of Thrones, Prison Break

1 3 5 4 2

Naïve approach

```
inversion=0
for i=1 to n-1
    for j=i+1 to n
        if A[i] > A[j] then
            inversion=inversiom +1
Return inversion
```

Naïve approach

```
inversion=0
for i=1 to n-1
    for j=i+1 to n
        if A[i] > A[j] then
            inversion=inversiom +1
Return inversion
```

$O(n^2)$-algorithm.

Can we do better?

Using D&C. Any suggestion?

Can we do better?

Using D&C. Any suggestion?

Divide problem in subproblems of size half and recurse, be smart to combine the solutions of subproblem.

Can we do better?

Using D&C. Any suggestion?

Divide problem in subproblems of size half and recurse, be smart to combine the solutions of subproblem.

problem of size n	
L prob. of size n/2	prob. of size n/2 R

inversions in L = N_L # inversions in L = N_R

Total = $N_L + N_R$ + number of pairs (i,j) s.t. L[i] > R[j]

Can we do better?

Using D&C. Any suggestion?

Divide problem in subproblems of size half and recurse, be smart to combine the solutions of subproblem.

problem of size n

L prob. of size n/2 prob. of size n/2

T(n)=2T(n/2)+ time required to combine solutions of two subproblems

\# inversions in L = N_L # inversions in L = N_R

Total = $N_L + N_R$ + number of pairs (i,j) s.t. L[i] > R[j]

inversions in L = N_L # inversions in L = N_R

Total = N_L + N_R + number of pairs (i,j) s.t. L[i] > R[j]

inversions in L = N_L # inversions in L = N_R

Total = $N_L + N_R +$ $\underbrace{\text{number of pairs } (i,j) \text{ s.t. } L[i] > R[j]}$

This comes from combining solution

inversions in L = N_L # inversions in L = N_R

Total = $N_L + N_R$ + $\underbrace{\text{number of pairs (i,j) s.t. } L[i] > R[j]}$

This comes from combining solution

Naïve way to combine solutions:

for each element in L, check each element of R and find pair which satisfies the inversion property.

inversions in L = N_L # inversions in L = N_R

Total = N_L + N_R + number of pairs (i,j) s.t. L[i] > R[j]

This comes from combining solution

Naïve way to combine solutions:

for each element in L, check each element of R and find pair which satisfies the inversion property. $O(n^2)$

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

inversions in L = N_L # inversions in L = N_R

Total = $N_L + N_R$ + $\underbrace{\text{number of pairs (i,j) s.t. } L[i] > R[j]}$

This comes from combining solution

A smart way to combine solutions:

1. Sort R

2. for each element e in L, do binary search on R and find pair which satisfies the inversion property.

inversions in L = N_L # inversions in L = N_R

Total = $N_L + N_R$ + $\underbrace{\text{number of pairs (i,j) s.t. } L[i] > R[j]}$

This comes from combining solution

A smart way to combine solutions:

1. Sort R

2. for each element e in L, do binary search on R and find pair which satisfies the inversion property. O(nlogn)

$$T(n)=2T(n/2)+O(n\log n)=O(n\log^2 n)$$

inversions in L = N_L # inversions in L = N_R

Total = $N_L + N_R$ + $\underbrace{\text{number of pairs (i,j) s.t. } L[i] > R[j]}$

This comes from combining solution

Even smarter way to combine solutions:

inversions in L = N_L # inversions in L = N_R

Total = $N_L + N_R +$ $\underbrace{\text{number of pairs (i,j) s.t. } L[i] > R[j]}$

This comes from combining solution

Even smarter way to combine solutions:

We need to solve the problem in O(nlogn), and we are traversing both halves, so we cannot go beyond O(n) to do extra work.

inversions in L = N_L # inversions in L = N_R

Total = N_L + N_R + $\underbrace{\text{number of pairs (i,j) s.t. } L[i] > R[j]}$

This comes from combining solution

Even smarter way to combine solutions:

Call Merge and Count routine (Last problem)

$T(n)=2T(n/2)+O(n)=O(n\log n)$