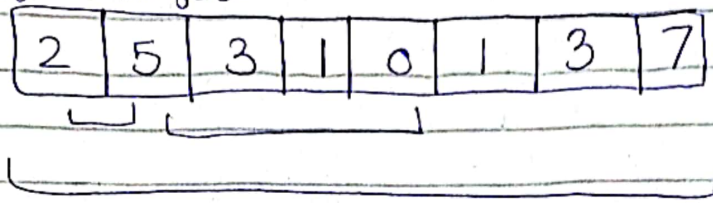
# Maximum Subarray Sum:

For positive integers:

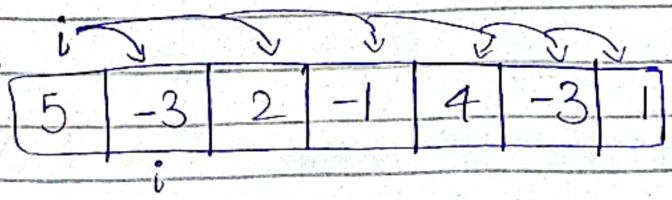



Max Sum = Sum of all elements & here O(n)
So, we've no issue with positive integers.
So, now we consider mixed integers:

For +ve/-ve integers:

e.g.

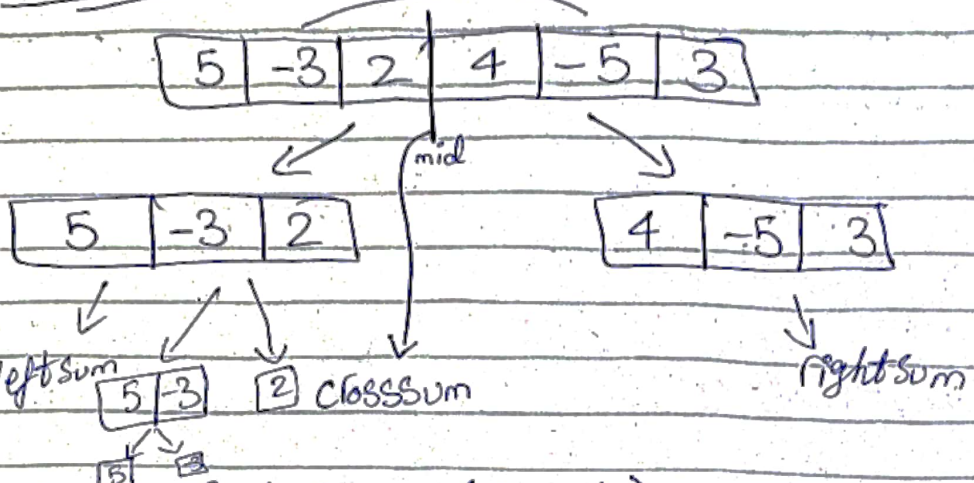

Method 1:




```
MaxSubArraySum(A,n)                    maxSum=5
{                                      currentSum=~~5~~2
                                       (2<5 so no need to update
   maxSum = A[0]
   ~~currentSum = A[0]~~   low=hi=0
   for(i←0 To n-1)
   { currentSum = A[i]
      for(j←i+1 To n-1)
         currentSum = currentSum + A[j]

      if (currentSum > maxSum)
      { maxSum = currentSum
        low = i        //keep track of subarray index
        hi = j         //  "   "   "   "
      }
   return low, hi, maxSum }        O(n²)
```

## Method 2: Based on Divide & Conquer:




$T(n) \leftarrow$ MaxSubArray (A, l, h)

{

if (h > l)

{

mid = $\frac{(h+l)}{2}$

$T(\frac{n}{2})$ leftSum = MaxSubArray (A, l, mid)

$T(\frac{n}{2})$ rightSum = MaxSubArray (A, mid, h)

$O(n)$ crossSum = CrossSubArraySum (A, l, mid, h)

return max (leftSum, rightSum, crosssum)

}

}

∴ For crossSum, move mid se peeche aur age & keep track of currentmax & overallmax & usi waqt keep track of indexes.

```
CrossSubArraySum(A,l,mid,h)
{
    leftsum = 0
    currsum = 0
    for (i ← mid-1  To  l)
    {   currsum = currsum + A[i]

        if (currsum > leftsum)
            leftsum = currsum
    }
    rightsum = 0
    currsum = 0
    for (i ← mid+1  To  h)
    {
        currsum = currsum + A[i]

        if (currsum > rightsum)
            rightsum = currsum
    }
}
```

Recurrence:

$$2T\left(\frac{n}{2}\right) + n$$

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

S1

$$T(n) = 2T\left(\frac{n}{2}\right) + n$$

S2

$$T(n) = 2\left[2T\left(\frac{n}{4}\right) + \frac{n}{2}\right] + n$$

$$= 2^2 T\left(\frac{n}{2^2}\right) + 2n$$

Sk

$$T(n) = 2^k T\left(\frac{n}{2^k}\right) + kn$$

(1) For $\frac{n}{2^k} = 1$

$$2^k = n$$

$$k = \log_2 n$$

$$T(n) = 2^{\log_2 n}(1) + n\log_2 n$$

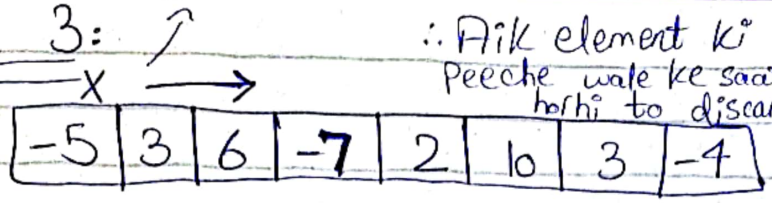
$$= n + n\log n$$

$$T(n) = n\log n$$

Method

As 3-5=-2<3 so start from 3

## Method 3:




∴ Aik element ki value peeche wale ke saath kam hothi to discard previous

maxSum = $-\infty$

| currSum | maxSum |
|---|---|
| -5 ⇒ -5 | -5 ⇒ -5 |
| -2 ⇒ -5,3 | -2 ⇒ -5,3 |
| 4 ⇒ -5,3,6 | |

```
MaxSubArraySum(A,n)
{   maxSum = -∞
    currSum = A[0]
    for(i ← 1 To n-1)
    {
        currSum = currSum + A[i]
        if(currSum < A[i])
        {   currSum = A[i]
            lo = i
        }
        if (currSum > maxSum)
            maxSum = currSum
    }
    return maxSum
}
```

O(n)

# Lower Bound for Comparison based Sorting algorithms:

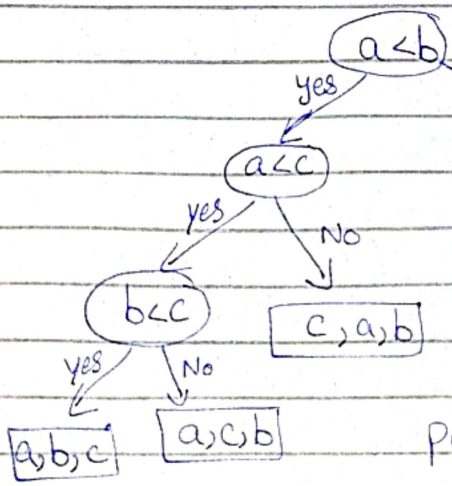
So far, best time complexity of sorting algos was O(nlogn) So comparison based algos are also not better than nlogn

Let a,b,c then permutations:

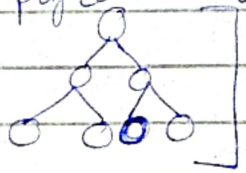
a,b,c
a,c,b
b,a,c
b,c,a
c,a,b
c,b,a

So if n integers then permutations are n!

a,b,c




perfect Binary Tree




Leaves = $2^h$ ≥ permutation

$2^h \geq n!$

$2^{T(n)} \geq n!$

(Har level pr check to h jitna time lge ga)

$T(n) \geq \log n!$

$T(n) \geq \log\left(\sqrt{2\pi n}\left(\frac{n}{e}\right)^n\right)$

Stirling approximation

$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$

$T(n) \geq \log(2\pi n)^{1/2} + \log\left(\frac{n}{e}\right)^n$

$T(n) \geq \frac{1}{2}\log 2\pi n + n\log\left(\frac{n}{e}\right)$ → higher order term

so, $T(n) \geq O(n\log n)$

Hence proved

Comparison based algos for large set of numbers & all types.
But if we've less numbers then instead of comparison based algos we should devise some other algo better than nlogn
i.e.

## Counting Sort (Not general, applied under restrictive circumstances)

The restrictive circumstance here are:

=> if range is defined
say (1 to k) → should be small
(k: max number)

In this technique we've 3 arrays:
1) A[1.....n] original Array
2) B[1....n] Sorted //
3) C[1....k) Rank // (frequency of each element)

| A | 7 | 2 | 4 | 9 | 1 | 0 | 3 | 4 | 5 | 6 | 7 | 3 | 5 | 6 | 7 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

AS range = 9 so C↑ array of size = 9

Initially:

| C | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

C array having count of element at index i ↑ when again reached & increment

C[A[i]] mei C[A[i]]+1 i.e.

| C | 1 | 1 | 1 | 2 | 2 | 2 | 2 | 3 | 0 | 2 |
|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

Update C with commulative frequency

$$C[i] \leftarrow C[i] + C[i-1]$$

| C | 1 | 2 | 3 | 5 | 7 | 9 | 11 | 14 | 14 | 16 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | |

→ 9 se less element 16 hai (inc of 9)

| B | | | | | | | | | 5 | | | | | | 7 | | 9 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | |

∴ Start A pr loop from end

∴ In C, see 5 se. less 9 element hai tou place 5 at index 9 in B array i.e.

$$B[C[A[i]]-1] = A[i]$$

when first 9 placed at 15 index in B so decrement 1 in C for next 9

```
CountingSort(A, n, k)
{
O(k) ← C[1....k], B[1...n]
     for(i←0 To k)
          C[i]←0

O(n) ← for(i←0 To n-1)
          C[A[i]] ← C[A[i]]+1

O(k) ← for(i←1 To k)
          C[i]← C[i]+C[i-1]

O(n) ← for(i←n-1 To 0)
          { B[C[A[i]]-1] = A[i]
               C[A[i]]-- }

}
```

e.g. if we've data to sort as: 11285, 3045925, 300
So here k is v.large & we'll have to make array of that v.large size. so then this technique not useful there.

## Radix Sort:

Let the data is 4583, 3210, 958, 2100, 6987, 5721

It sorts digit by digit starting from least significant digit

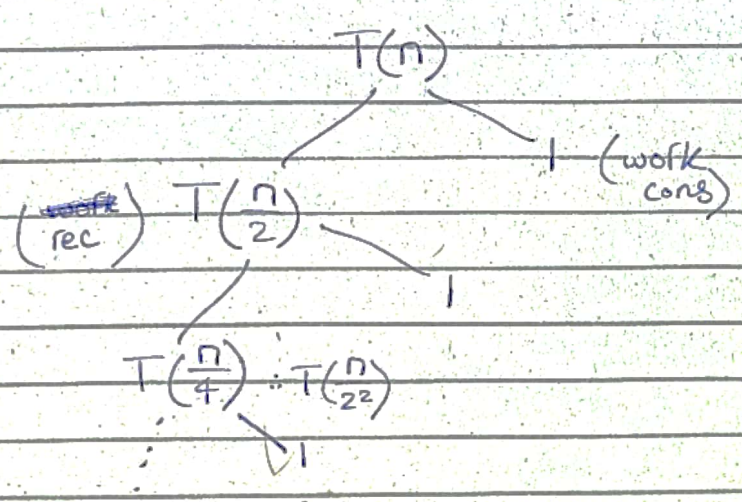
| | | | | |
|---|---|---|---|---|
| 4583 | 3210 | 2100 | 2100 | 958 |
| 3210 | 2100 | 3210 | 3210 | 2100 |
| 958 | ⇒ 5721 | ⇒ 5721 | ⇒ 4583 | ⇒ 3210 |
| 2100 | 4583 | 958 | 5721 | ⇒ 4583 |
| 6987 | 6987 | 4583 | 958 | 5721 |
| 5721 | 958 | 6987 | 6987 | 6987 |

So, inner loop will be 1 to digit number (like ÷ 10 here) Taking modulus & starting from right

## Master Theorem: (For solving recurrence)

$$T(n) = T\left(\frac{n}{2}\right) + 1$$

Tree Method




$k = \log n$ (Total steps)

$T(n) = 1 + 1 + \cdots + k$ time

$T(n) = 1 \log n$

We've recurrence for
→ decreasing function

**Master Theorem (for decreasing function):**

$$T(n) = T(n-1) + f(n)$$

General Form:

$$T(n) = aT(n-b) + f(n)$$

Cases:

1) if $(a == 1)$

$$T(n) = O\left(\frac{n}{b} * f(n)\right)$$

e.g. $T(n) = T(n-1)+1 \Rightarrow O(n)$ $a=1, b=1, f(n)=1$

e.g. $2T(n-1)+1 \Rightarrow O(2^n)$

2) if $(a > 1)$

$$T(n) = O(f(n) * a^{n/b})$$

e.g. $T(n) = 2T(n-1)+1$

$a=2, b=1, f(n)=1$

$T(n) = O(2^n)$ (putting in formula)

3) if $(a < 1)$

$$T(n) = O(f(n))$$