ESc101 : Fundamental of Computing

I Semester 2008-09

Lecture 36

- Announcement : Extra sessions for lab test
- Sorting algorithms based on recursion
  - Quick Sort (did in lst class)
  - Merge Sort
- Introduction to Time Complexity

Extra session for lab

On Sunday- 9th November

B1 and B2 :	Nikhi Jain	CS101 2-3:30 PM
B3 and B4 :	Kshitiz Garg	CS102 2-3:30 PM.
B5 and B6 :	Abhinav/Nitin	CS101 3:45-5:15PM
B7 and B8 :	Vikas Mards	CS102 3:45-5:15PM
B9 and B10:	Paras Tikamani	CS101 5:15-6:45PM

Optional extra class

Saturday- 8th November at 5:15 PM in CS101

"Tower of Hanoi" problem

Hints: You can see that for 1 or 2 discs, there is a straight forward solution.

Now for the problem with $n+1$ discs, assume you have a black box method which can transfer n discs from one tower to another. How can you solve the given problem (that is, $n+1$ discs) using that method.

Now try to fill in the details of the recursive method. The final solution (recursive method) is very small....

Quick Sort

Quick Sort : sorting a set S of numbers

- Select an element x.
- Partition the set S of numbers into two parts:
  - $S_{<x}$: the subset consisting of numbers less than x.
  - $S_{>x}$: the subset consisting of numbers greater than x.
- Recursively sort the two sets separately $S_{<x}$ and $S_{>x}$, and *concatenate* them.

Quick Sort : when set is represented as array

```
public static void Qsort(int[] A, int left, int right)
{
    if(left<right)
    {
        int mid = partition(A, left, right);
        Qsort(A,left,mid-1);
        Qsort(A,mid+1,right);
    }
}
```

You can observe that the size of problem corresponding to recursive calls decreases always. Hence the program will eventually terminate.

Merge Sort

Merging two sorted arrays

Problem : Given two sorted arrays A and B, produce another sorted array $C \bigcup A \cup B$.

Merging two sorted arrays

Problem : Given two sorted arrays A and B, produce another sorted array $C \bigcup A \cup B$.

Trivial Solution :

Copy the elements of A and B into C, then sort C

Merging two sorted arrays

Problem : Given two sorted arrays A and B, produce another sorted array $C \bigcup A \cup B$.

Trivial Solution :

Copy the elements of A and B into C, then sort C

Missing Point : A and B are already sorted

Merging two sorted arrays : a better solution

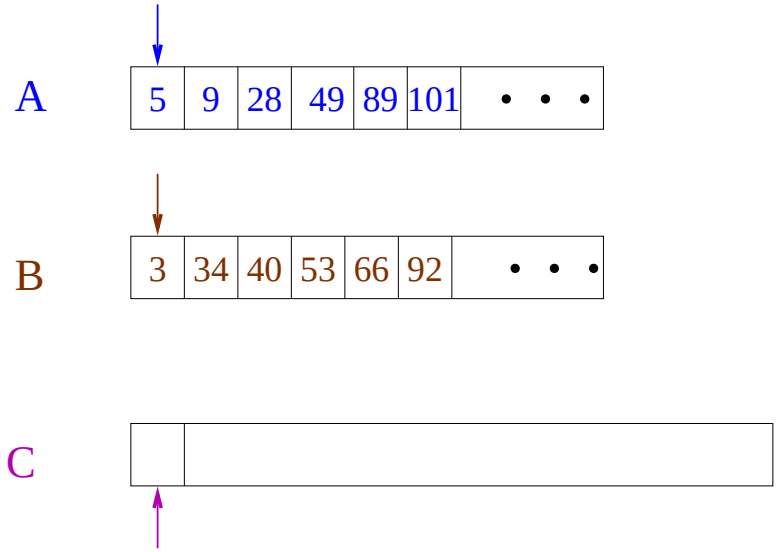
start scanning A and B from left, compare two elements of A and B, copy the smaller one into C and continue ...

Merging two sorted arrays : a better solution

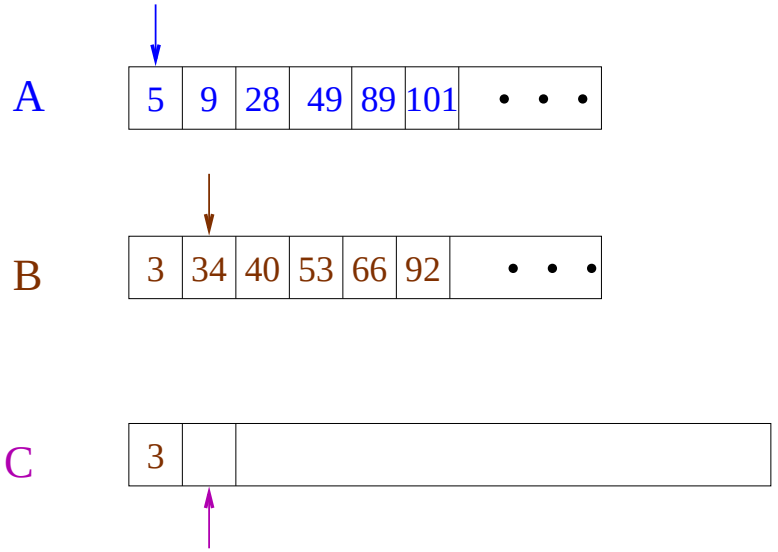
start scanning A and B from left, compare two elements of A and B, copy the smaller one into C and continue ...

Let us consider an example ...

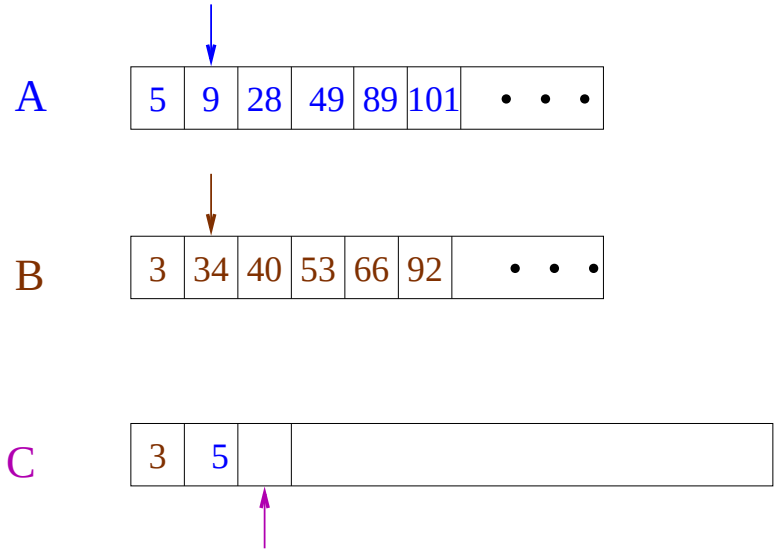
Example : Merging two sorted arrays



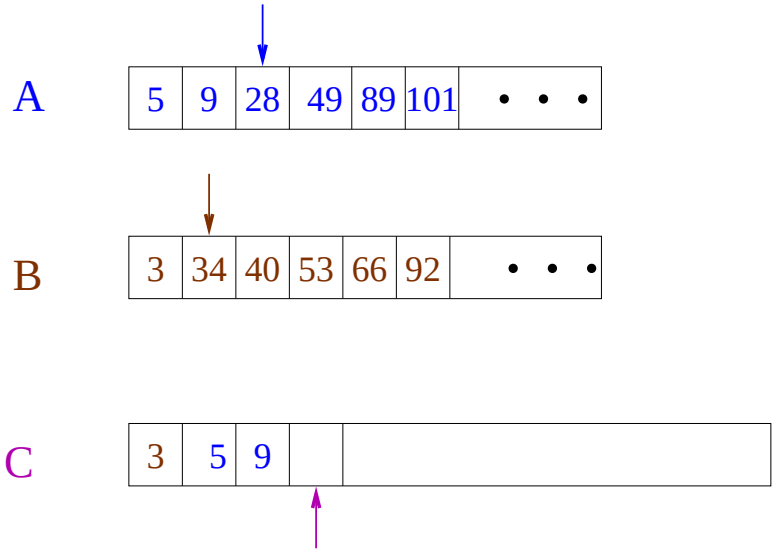
Example : Merging two sorted arrays



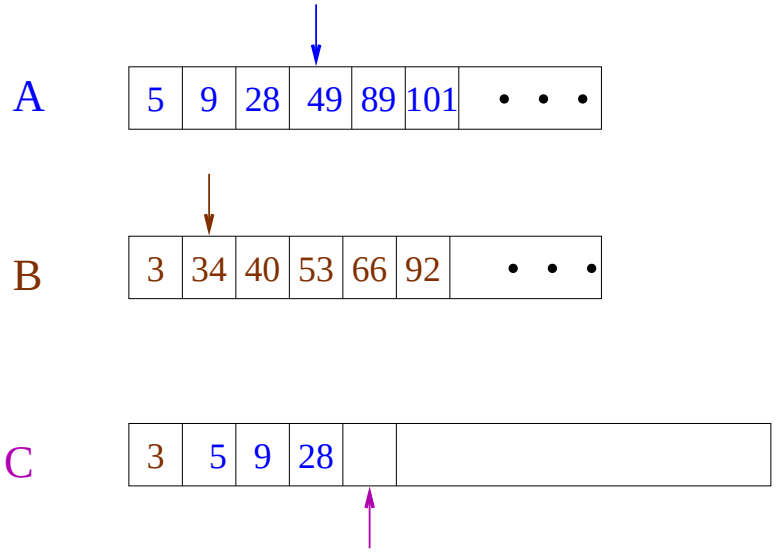
Example : Merging two sorted arrays



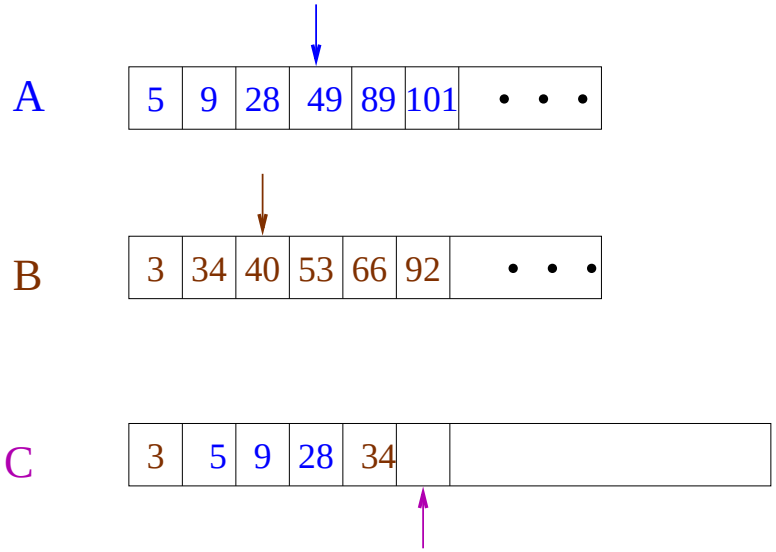
Example : Merging two sorted arrays



Example : Merging two sorted arrays



Example : Merging two sorted arrays



Merge Sort

Key Idea : Merging two sorted arrays is easier than sorting their union.

Merge Sort on an array A

- Sort the first half of array A recursively
- Sort the second half of the array recursively
- merge the two halves.

Merge Sort on an array A

```
public static void mergesort(int[] A, int left, int right)
    {
        if(     ??     )
        {
                    ??          ;
                    ??          ;
                    ??          ;
                    ??          ;
        }
    }
```

Merge Sort on an array A

```
public static void mergesort(int[] A, int left, int right)
    {
        if(left<right)
        {
                    ??          ;
                    ??          ;
                    ??          ;
                    ??          ;
        }
    }
```

Merge Sort on an array A

```
public static void mergesort(int[] A, int left, int right)
    {
        if(left!=right)
        {
            int mid = (left+right)/2;
                      ??            ;
                      ??            ;
                      ??            ;
        }
    }
```

Merge Sort on an array A

```
public static void mergesort(int[] A, int left, int right)
    {
        if(left!=right)
        {
            int mid = (left+right)/2;
            mergesort(A, left, mid);
                      ??           ;
                      ??           ;
        }
    }
```

Merge Sort on an array A

```
public static void mergesort(int[] A, int left, int right)
    {
        if(left!=right)
        {
            int mid = (left+right)/2;
            mergesort(A, left, mid);
            mergesort(A, mid+1, right);
                    ??                ;
        }
    }
```

Merge Sort on an array A

```
public static void mergesort(int[] A, int left, int right)
    {
        if(left!=right)
        {
            int mid = (left+right)/2;
            mergesort(A, left, mid);
            mergesort(A, mid+1, right);
            merge(A,left,mid,right);
        }
    }
```

Merge Sort on an array A

```
public static void mergesort(int[] A, int left, int right)
    {
        if(left!=right)
        {
            int mid = (left+right)/2;
            mergesort(A, left, mid);
            mergesort(A, mid+1, right);
            merge(A,left,mid,right);
        }
    }
```

Convince yourself that each recursive call makes progress, that is, it approaches the base case. This is very important rule that each recursive method must obey.

Recursion Tree

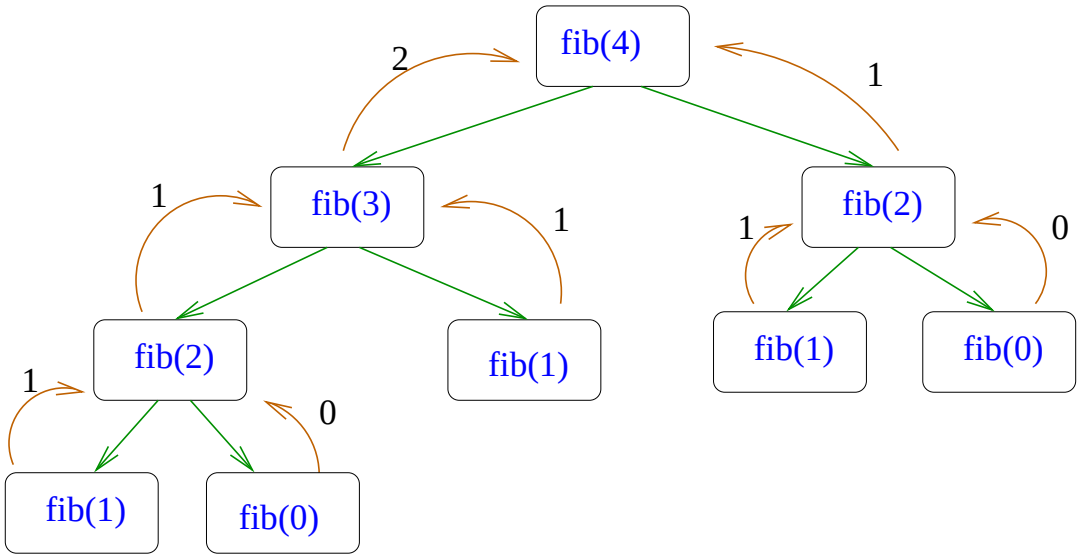
When a method makes two or more recursive calls to itself, it is better to view the execution as a tree.

Recursion Tree for Fibonacci number

```
public static int fib(int n)
{
    if(n==0) return 0;
    else
    {    if(n==1) return 1;
         else return fib(n-1)+fib(n-2);
    }
}
```

Example : n=4

Recursion Tree for Fibonacci number for n = 4



Recursion Tree for Merge sort

```
public static void mergesort(int[] A, int left, int right)
    {
        if(left!=right)
        {
            int mid = (left+right)/2;
            mergesort(A, left, mid);
            mergesort(A, mid+1, right);
            merge(A,left,mid,right);
        }
    }
```

Example : A $= \{99, 7, 5, 1, 67, 11, 4, 2\}$

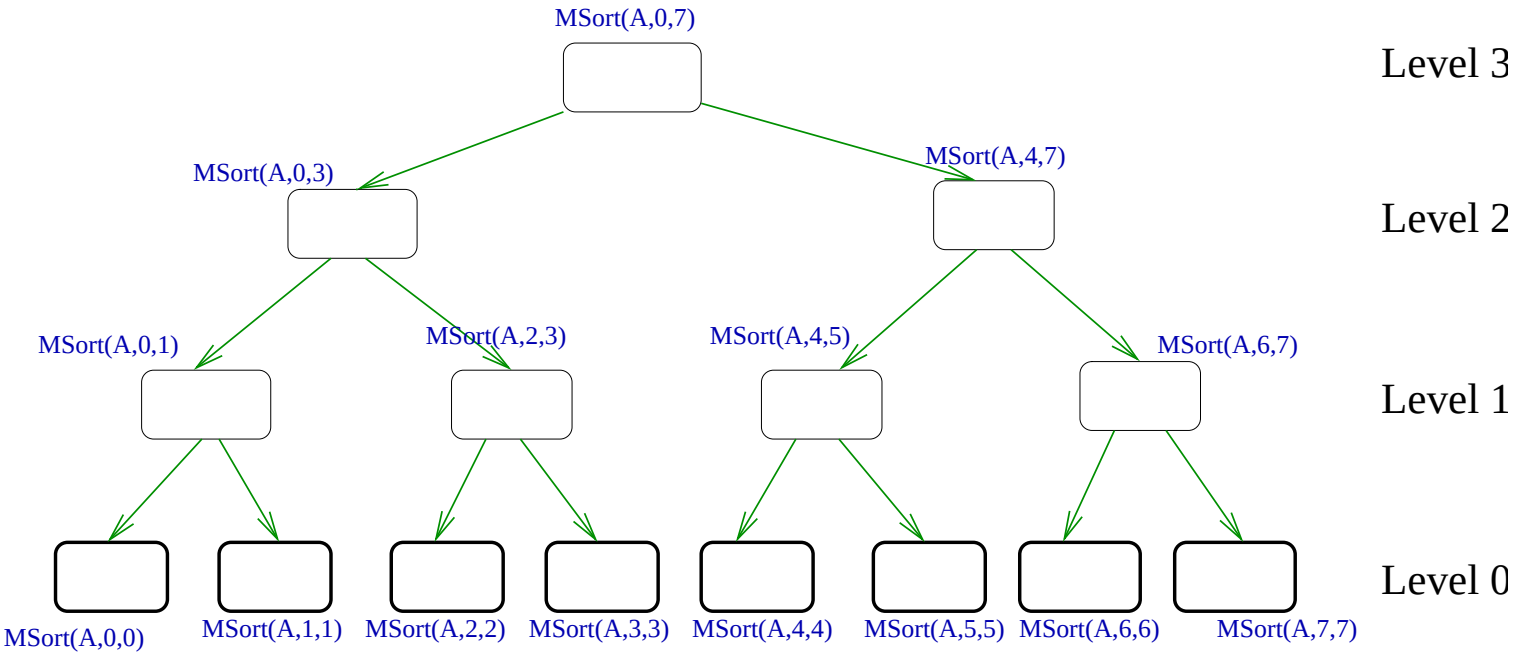
Recursion Tree for Merge sort

for A $= \{99, 7, 5, 1, 67, 11, 4, 2\}$

Note : In the next few slides, for sake of compactness we shall use MSort() to denote mergesort().

We show the status of A as we move up the recursion tree level by level.

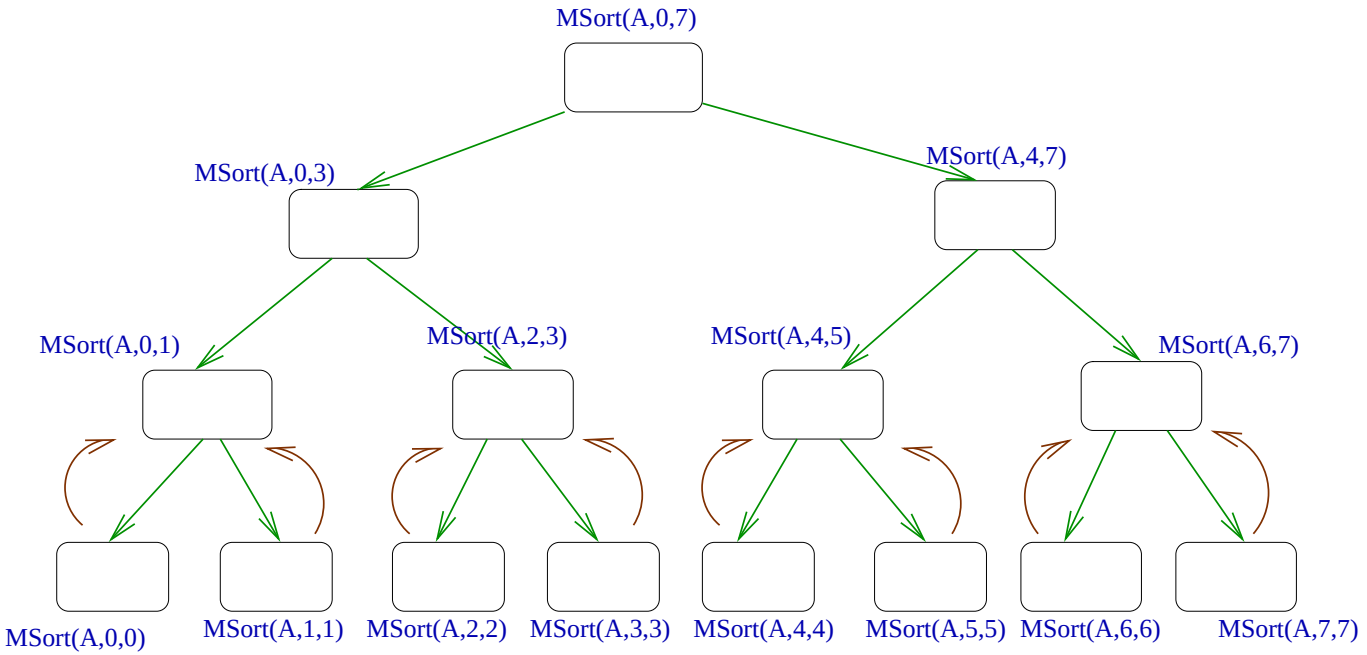
Base case : Level 0 (no processing required)



A

99	7	5	1	67	11	4	2

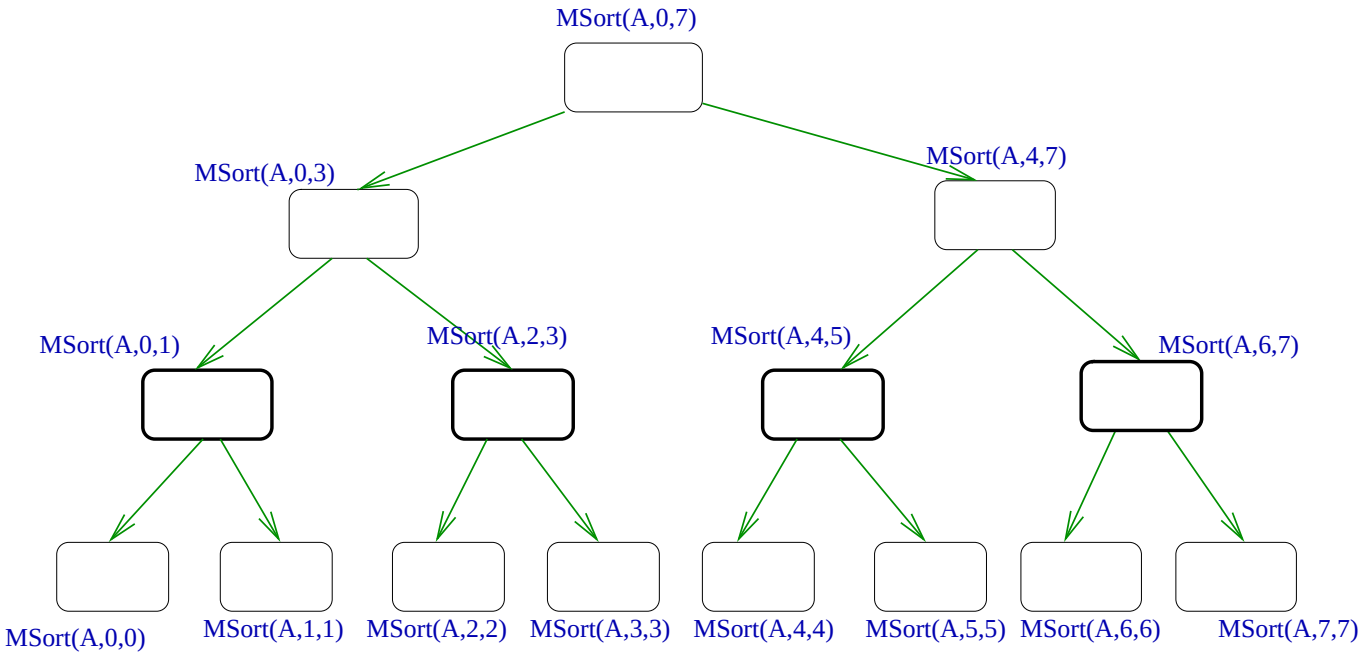
Passing results to Level 1



A

99	7	5	1	67	11	4	2

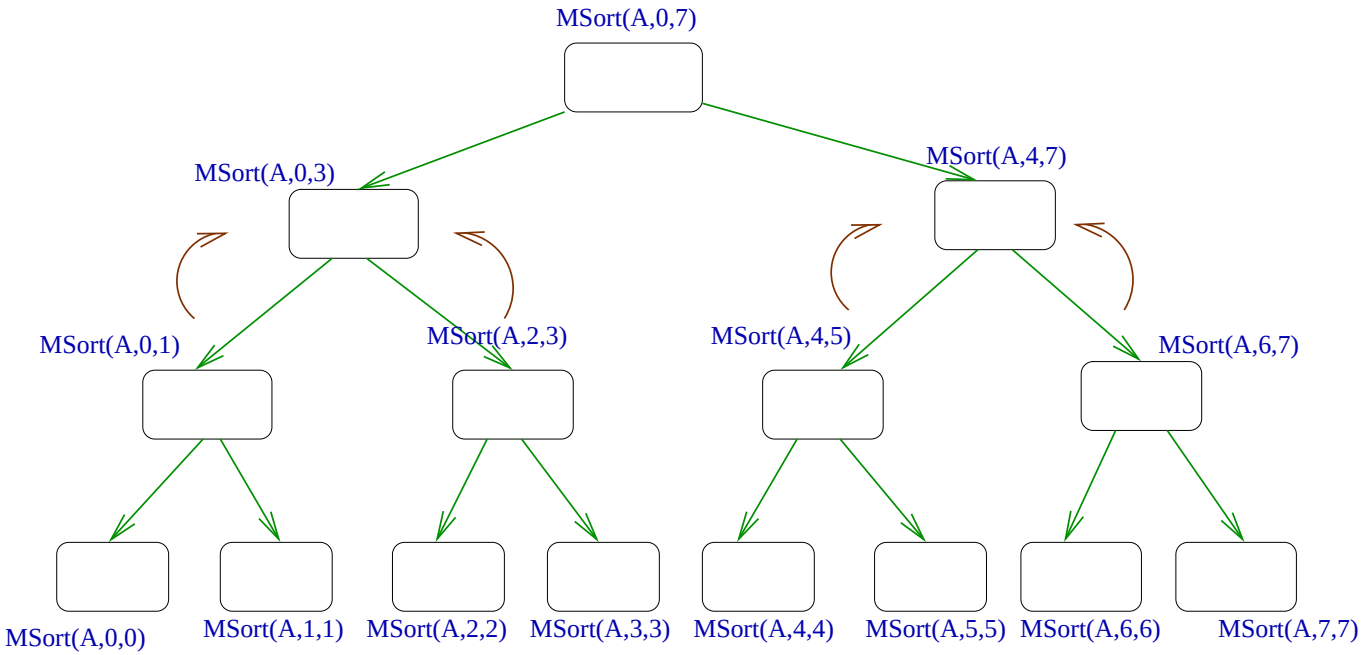
Merging at Level 1



A

7	99	1	5	11	67	2	4

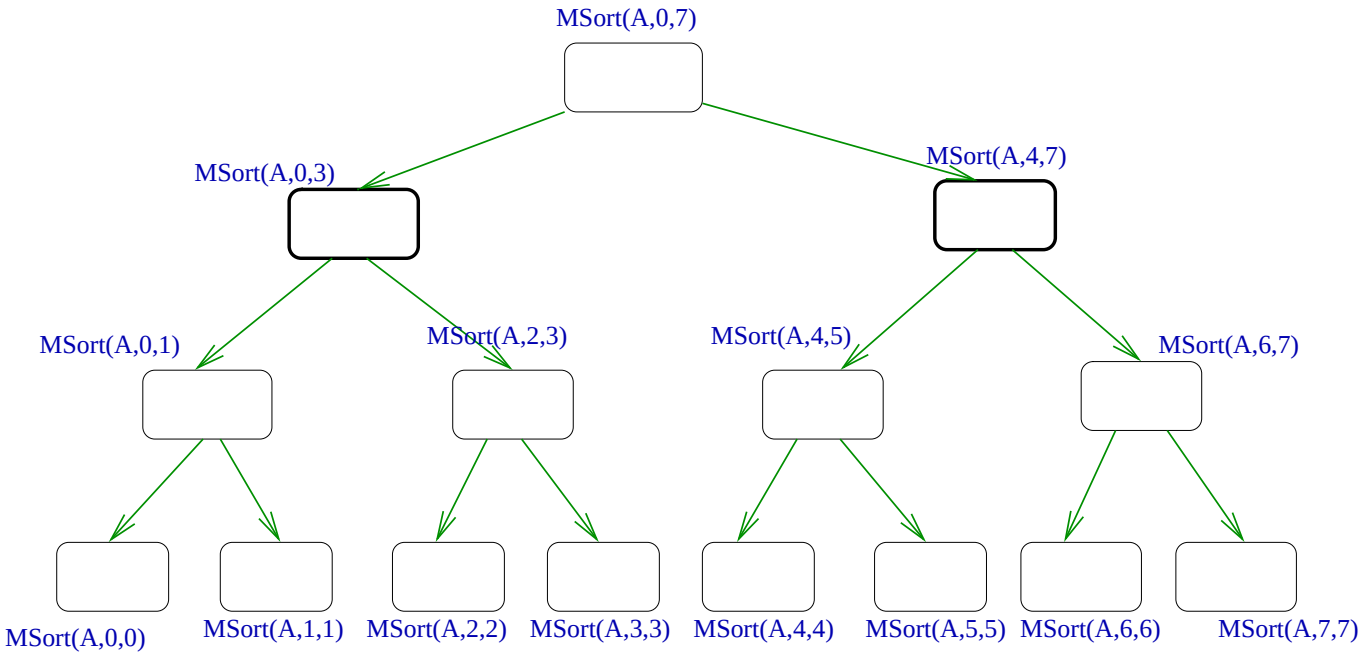
Passing results to Level 2



A

7	99	1	5	11	67	2	4

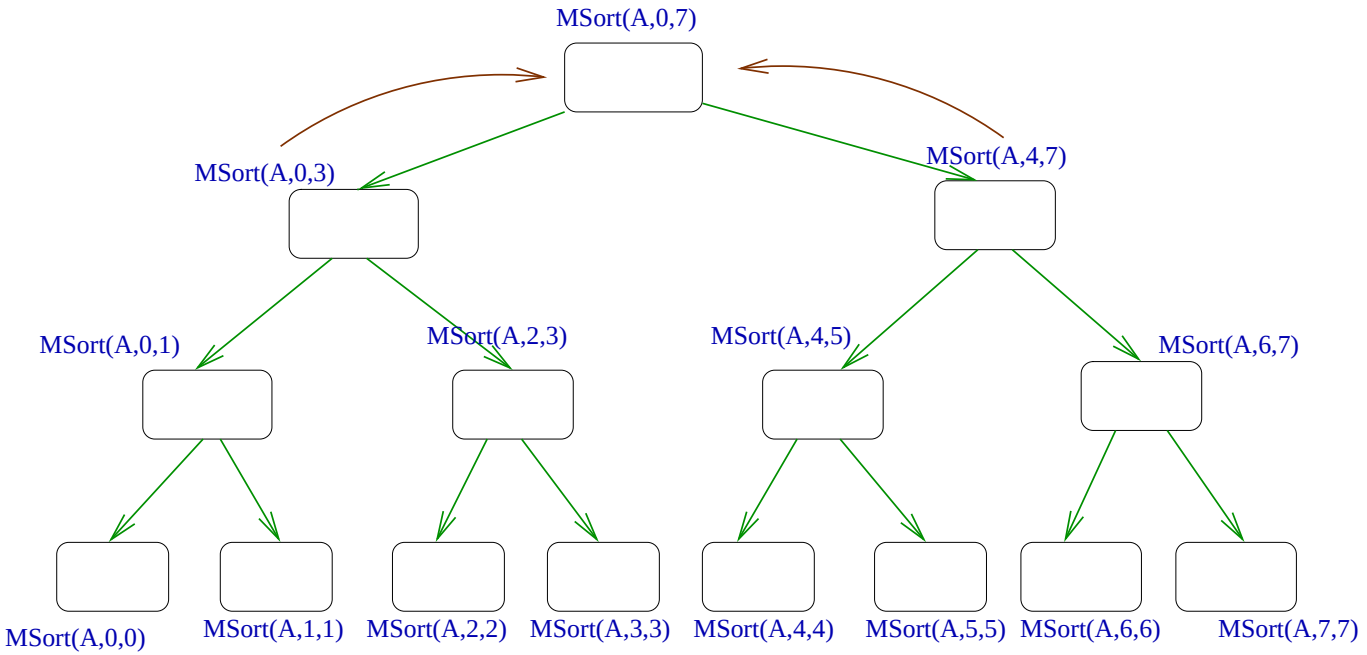
Merging at Level 2



A

1	5	7	99	2	4	11	67

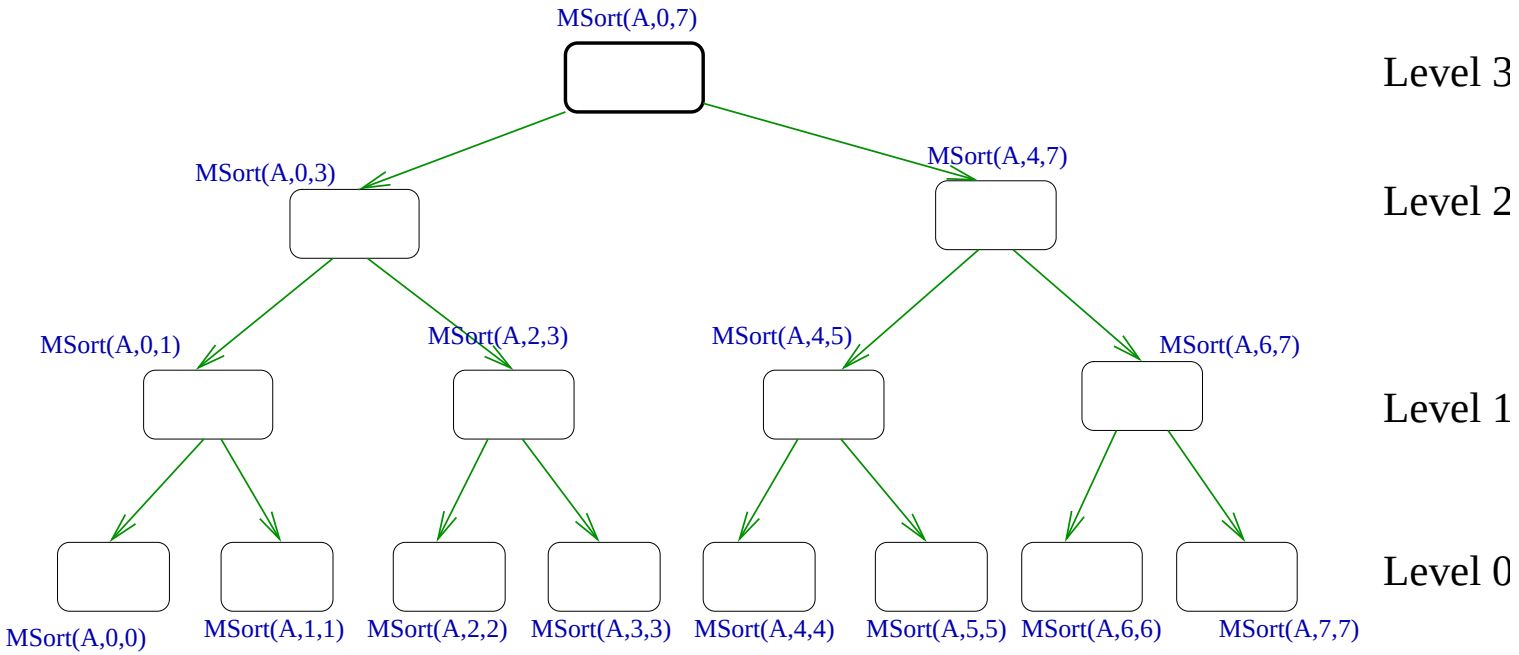
Passing result to Level 3



A

1	5	7	99	2	4	11	67

Merging at Level 3



A

1	2	4	5	7	11	67	99

Recursion Tree for Quick sort

Do it as homework

Comparing Three sorting algorithms

```
for(n=1000,n<20000;n=n+1000)
   {    Generate an array A of size n;
        Fill it with random integers;
        Create copies B and C of array A;
        Execute Quick sort on A and measure time.
        Execute Merge sort on A and measure time.
        Execute SelectionSort on B and measure time.
   }
```

Measuring time taken a method M

```
long start = System.currentTimeMillis();
M();
long stop = System.currentTimeMillis();
System.out.println(stop-start);
```

Note : `System.currentTimeMillis()` returns a long which corresponds to current time in milliseconds.

Comparing Three sorting algorithms

Experimental Observations

- **Quick sort** is more efficient than **merge sort**
- **Merge sort** is more efficient than **Selection Sort**

Please study the program : **Three_sorting_algos.java**

What is the reason for different running times ?

Given that

- all of them has same input and output
- all of them are executed on the same machine

We need to analyze the number of steps/instruction taken by each sorting algorithm

How many steps/instructions are executed by the following loop ?

```
for(int i=1; i<=n; i=i+1)
{
    sum = sum + i;
}
```

Steps =

How many steps/instructions are executed by the following loop ?

```
for(int i=1; i<=n; i=i+1)
{
    sum = sum + i;
}
```

Steps = $1 + 3n + 1$

How many steps/instructions are executed by the following loop ?

```
for(int n=1;n<=m;n=n+1)
{
    for(int i=1; i<=n; i=i+1)
    {
      sum = sum + i;
    }
}
```

Steps =

How many steps/instructions are executed by the following loop ?

```
for(int n=1;n<=m;n=n+1)
{
    for(int i=1; i<=n; i=i+1)
    {
      sum = sum + i;
    }
}
```

Steps = $1 + m + \sum_{n=1}^{n=m}(1 + 3n + 1) + m + 1$